



## Red Hat Reference Architecture Series

# Resource Management using Control Groups in Red Hat® Enterprise Linux 6

## Case Study: Database Performance

<b>OLTP Workload</b>
<b>Oracle 10g</b>
<b>Red Hat® Enterprise Linux 6</b>
<b>Intel Nehalem EX</b>

Version 1.0  
November 2010





## Resource Management using Control Groups in Red Hat Enterprise Linux 6 Case Study: Database Performance

1801 Varsity Drive™  
Raleigh NC 27606-2072 USA  
Phone: +1 919 754 3700  
Phone: 888 733 4281  
Fax: +1 919 754 3701  
PO Box 13588

Research Triangle Park NC 27709 USA

Linux is a registered trademark of Linus Torvalds. Red Hat, Red Hat Enterprise Linux and the Red Hat "Shadowman" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group.

Intel, the Intel logo, Xeon and Itanium are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

© 2010 by Red Hat, Inc. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

The information contained herein is subject to change without notice. Red Hat, Inc. shall not be liable for technical or editorial errors or omissions contained herein.

Distribution of modified versions of this document is prohibited without the explicit permission of Red Hat Inc.

Distribution of this work or derivative of this work in any standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from Red Hat Inc.

The GPG fingerprint of the security@redhat.com key is:  
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E



# Table of Contents

1 Executive Summary.....	4
2 Introduction.....	5
2.1 Definitions.....	6
2.1.1 Hierarchy.....	6
2.1.2 Subsystem.....	6
3 Testbed Configuration.....	8
3.1 Hardware.....	8
3.2 Software.....	8
3.3 Storage.....	8
3.4 Control Groups.....	9
3.4.1 Creation.....	10
3.4.2 Cpuset.....	11
3.4.2.1 NUMA Pinning.....	11
4 Usage Scenarios.....	12
4.1 Application Consolidation.....	12
4.2 Performance Optimization.....	15
4.3 Dynamic Resource Management.....	17
4.3.1 Single Instance.....	18
4.3.2 Multi Instance.....	19
4.4 Application Isolation.....	22
5 References.....	27



# 1 Executive Summary

Previously known as container groups, task control groups (cgroups) provide a method of allocating memory, processor and I/O resources across process groups, whether those process groups are applications or virtual guests.

For the purposes of this document, all testing was performed on bare metal without virtualization. Also, this document focuses on using the cpuset and memory subsystems to manage CPU and memory as well as to pin cgroups to specific Non-Uniform Memory Access (NUMA) nodes.

The following benefits of cgroups are demonstrated in four use cases using a database workload:

1. **Application Consolidation** - Cgroups may be used to consolidate multiple applications onto larger servers by providing the user control over the level of resources allocated to each application.
2. **Performance Optimization** – Judicious mapping of cgroups to system resources can result in significant performance improvements. E.g., If cgroups are used to pin each application to the memory and CPUs in a separate NUMA node, it results in reduced memory latency and better overall performance.
3. **Dynamic Resource Management** – Often certain applications, in an application mix running on a server, require additional resources at specific times. Cgroups may be used dynamically alter the resources allocated to each application, as needed, for optimal system performance.
4. **Application Isolation** – Confining each application to its own cgroup ensures that an attempt by a rogue application to hog a system resource (e.g., CPU or memory) does not impact other applications running in other cgroups.



## 2 Introduction

The saturation of one or more system resources is most commonly the cause of poor system performance. In the standard multi user environment, system resources are controlled by the operating system (OS) and are shared via a scheduling algorithm.

Resource control is a method of allocating the resources of a system in a controlled manner. Red Hat Enterprise Linux 6 provides new ways of grouping tasks and dividing system resources into groups for improved performance.

The use of cgroups provides an administrator a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specific behavior. The user can monitor any cgroups they configure, deny cgroups access to certain resources, and even reconfigure their cgroups dynamically on a running system. The control group configuration service (`cgconfig`) can be configured to start at system boot and reestablish any predefined cgroups so they remain persistent after reboot.

Control groups provide:

- **Resource limits:** set memory or file system cache limits on specific groups
- **Isolation:** groups can be given separate namespaces so that each group does not detect the processes, network connections, or files belonging to others
- **Control:** freezing groups or checkpointing and restarting
- **Prioritization:** different groups may receive more I/O throughput or CPU cycles than others
- **Accounting:** measure the resources used by specific systems

In using control groups, a set of tasks can be associated with a set of parameters for one or more subsystems. For example, an application in a particular control group is given a user specified share of the resources of that system. These shares are minimums and not maximums, meaning if one group is allocated 20% of the CPU processing power of a server and another group is allocated 80% but is not using the 80%, the other group is able to utilize the remaining CPUs.

Not only do cgroups provide capabilities much like that of `numactl` and `taskset` to pin tasks to NUMA nodes or to specific CPUs, they can also be used to dynamically manage resources such as disk I/O, memory, and network bandwidth. Although `taskset` is also able to set CPU affinity on the fly, cgroups can provide the `numactl` functionality dynamically as well. Note that even though memory can be reallocated in mid test, it is neither recommended nor supported at this time.

Although control groups were written as the low-level mechanism in Linux for containers and virtual machines, they are not restricted to virtual machines. They can manage the resources and performance of ordinary processes as well.



## 2.1 Definitions

### 2.1.1 Hierarchy

A *cgroup hierarchy* is a tree of cgroups that has zero or more subsystems attached to it. Each hierarchy presents itself to the user as a virtual file system or *cgroup*, in which each directory is a single cgroup containing various control files and descendant cgroups as subdirectories. So within this file system, each directory defines a new group and as such, groups can be arranged to form an arbitrarily nested hierarchy by creating new sub-directories.

### 2.1.2 Subsystem

Simplified, a subsystem (or resource controller) is something that acts upon a group of tasks (i.e., processes). It is a module that makes use of the task grouping facilities provided by cgroups to treat groups of tasks in specific ways. A subsystem is typically a resource controller that schedules a resource or applies limits per cgroup, but in reality could be anything that wants to act on a group of processes (e.g., a virtualization subsystem). An example would be the *cpuset* subsystem, which limits which processors on which tasks can run and for how long.

Subsystems are also known as resource controllers. The terms *resource controller* or simply *controller* are often seen in control group literature such as man pages or kernel documentation. Both of these terms are synonymous with subsystem and are used because a subsystem typically schedules a resource or applies a limit to the cgroups in the hierarchy to which it is attached. For instance, the CPU resource controller allocates processing resources to a cgroup on a proportional basis, and with interactive, real-time tasks, each task are assigned a specific number of CPU cycles.

Cgroups are similar to processes in as much as they are hierarchical and child cgroups inherit certain attributes from their parent cgroup. The fundamental difference is that many different hierarchies of cgroups can exist simultaneously on a system. If the Linux process model is a single tree of processes, then the cgroup model is one or more separate, unconnected trees of tasks (i.e., processes).

Multiple separate hierarchies of cgroups are necessary because each hierarchy is attached to one or more subsystems. A subsystem represents a single resource, such as CPU time or memory.

Each of the subsystems described are implemented by mounting one or more subsystems as virtual file systems.

The available control group subsystems in Red Hat Enterprise Linux 6 and their functions are:

- **blkio** - this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.)
- **cpu** - this subsystem uses the scheduler to provide cgroup tasks access to the CPU
- **cpuacct** - this subsystem generates automatic reports on CPU resources used by tasks in a cgroup



- **cpuset** - this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup
- **devices** - this subsystem allows or denies access to devices by tasks in a cgroup
- **freezer** - this subsystem suspends or resumes tasks in a cgroup
- **memory** - this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic reports on memory resources used by those tasks
- **net\_cls** - this subsystem tags network packets with a class identifier that allows the Linux traffic controller to identify packets originating from a particular cgroup task
- **ns** — the *namespace* subsystem

**NOTE:** This document focuses on using the *cpuset* and *memory* cgroup subsystems to manage both CPU and memory as well as to restrict (aka: pin) cgroups to specific NUMA nodes.



## 3 Testbed Configuration

The following configuration details describe the testbed used to test control groups with an Oracle OLTP workload.

### 3.1 Hardware

Server	Specifications
Intel Nehalem EX	Quad Socket, 32 CPU (32 cores) Intel® Xeon® CPU X7560 @2.27GHz 128GB RAM (32GB per NUMA node)
	1 x 72 GB SAS 15K internal disk drive
	2 x QLogic ISP2432-based 4Gb FC HBA

**Table 1: Hardware Configuration**

### 3.2 Software

Software	Version
Red Hat Enterprise Linux (RHEL)	6 (2.6.32-71.el6 kernel)
Oracle	10.2.0.4

**Table 2: Software Configuration**

### 3.3 Storage

Hardware	Specifications
1 x HP StorageWorks HSV200 Fibre Channel Storage Array [28 x 136GB 15K RPM SCSI disks]	Controller Firmware Version: 6110
	Software Version: CR0ECAxc3p-6110
	Disk Firmware Version: HP03
1 x HP StorageWorks 2/32 SAN Switch	Kernel: 2.4.19 Fabric OS: v4.1.1 BootProm: 3.2.4

**Table 3: Storage Hardware**





## 3.4 Control Groups

This section includes the control group configurations used during testing. Reference the **Red Hat Enterprise Linux 6 Resource Management Guide** for complete details regarding control groups.

The cgroup configuration file (*/etc/cgconfig.conf*) content used for this effort defines four groups, test1 through test4, to control the resources of four oracle database instances.

```
mount {
    cpuset = /cgroup/cpuset;
    cpu    = /cgroup/cpu;
    cpuacct = /cgroup/cpuacct;
    memory = /cgroup/memory;
    devices = /cgroup/devices;
    freezer = /cgroup/freezer;
    net_cls = /cgroup/net_cls;
    blkio  = /cgroup/blkio;
}

group test1 {
    perm {
        task {
            uid = oracle;
            gid = dba;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpuset {
        cpuset.cpus=0-31;
        cpuset.mems=0-3;
    }
}

group test2 {
    perm {
        task {
            uid = oracle2;
            gid = dba;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpuset {
        cpuset.cpus=0-31;
        cpuset.mems=0-3;
    }
}
```



```
}
group test3 {
    perm {
        task {
            uid = oracle3;
            gid = dba;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpuset {
        cpuset.cpus=0-31;
        cpuset.mems=0-3;
    }
}
group test4 {
    perm {
        task {
            uid = oracle4;
            gid = dba;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpuset {
        cpuset.cpus=0-31;
        cpuset.mems=0-3;
    }
}
```

The specified CPUs and NUMA nodes, defined by *cpuset.cpus* and *cpuset.mems* respectively, default to using all available resources and were modified dynamically during testing. Users with preferences regarding the resource levels they wish to allocate to their application(s) can configure cgroups accordingly in their configuration file.

### 3.4.1 Creation

The following `cgcreate` commands were used to create the four control groups, `test1` through `test4`, for use in testing.

```
# cgcreate -t oracle:dba -g cpuset:test1
# cgcreate -t oracle2:dba -g cpuset:test2
# cgcreate -t oracle3:dba -g cpuset:test3
# cgcreate -t oracle4:dba -g cpuset:test4
```



## 3.4.2 Cpuset

The *cpuset* subsystem assigns individual CPUs and memory nodes to control groups. The changes to cpusets were performed before each single instance test as well as dynamically during multi instance using the *cgset* command to set subsystem parameters from a user account with permission to modify the specific control group.

The subsystem parameters modified in this testing were *cpuset.cpus* (specifies the CPUs that tasks in the cgroup are permitted to use) and *cpuset.mems* (specifies the NUMA or memory nodes that tasks in this cgroup are permitted to use).

### 3.4.2.1 NUMA Pinning

The NUMA configuration for the Nehalem EX server used in testing.

```
# numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28
node 0 size: 32649 MB
node 0 free: 21762 MB
node 1 cpus: 1 5 9 13 17 21 25 29
node 1 size: 32768 MB
node 1 free: 26818 MB
node 2 cpus: 2 6 10 14 18 22 26 30
node 2 size: 32768 MB
node 2 free: 25832 MB
node 3 cpus: 3 7 11 15 19 23 27 31
node 3 size: 32768 MB
node 3 free: 30561 MB
node distances:
node  0  1  2  3
  0: 10 21 21 21
  1: 21 10 21 21
  2: 21 21 10 21
  3: 21 21 21 10
```

Using the CPU list per NUMA node as output above by *numactl*, the following commands were used to pin the four cgroups to the CPUs and memory of the four NUMA nodes.

```
# cgset -r cpuset.cpus='0,4,8,12,16,20,24,28' test1
# cgset -r cpuset.mems='0' test1
# cgset -r cpuset.cpus='1,5,9,13,17,21,25,29' test2
# cgset -r cpuset.mems='1' test2
# cgset -r cpuset.cpus='2,6,10,14,18,22,26,30' test3
# cgset -r cpuset.mems='2' test3
# cgset -r cpuset.cpus='3,7,11,15,19,23,27,31' test4
# cgset -r cpuset.mems='3' test4
```



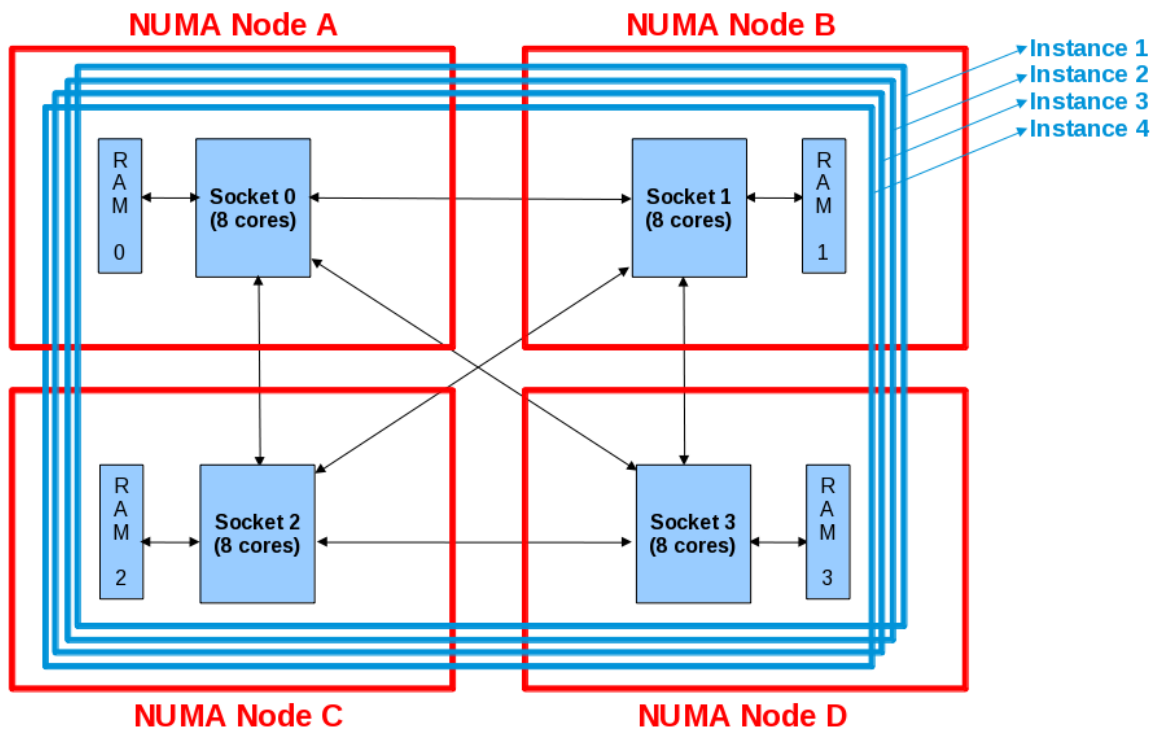
# 4 Usage Scenarios

A series of tests were performed with an Oracle database executing an OLTP workload. The objective of these tests was to understand the effects of adjusting the *memory* and *cpusets* cgroup subsystems and characterize any performance impact.

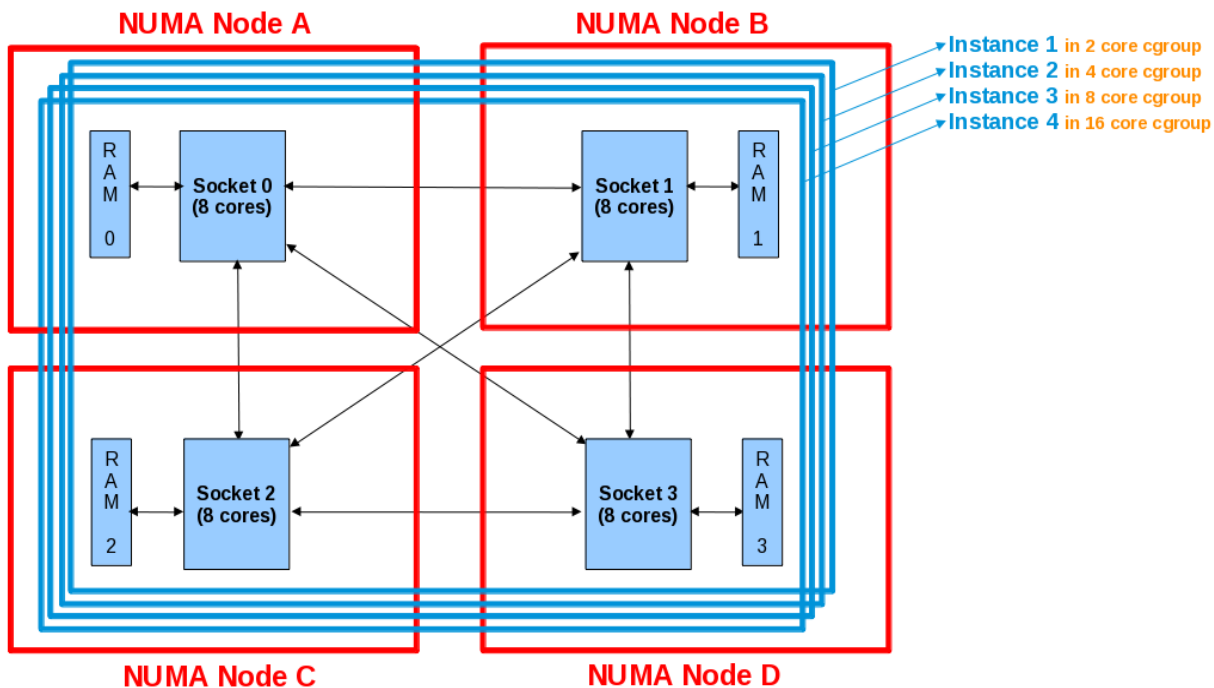
## 4.1 Application Consolidation

Cgroups can be used to consolidate multiple applications onto larger servers by providing the user control over the level of resources allocated to each application. This test did not pin cgroups to NUMA nodes in order to be able to allocate CPUs existing in one NUMA node to applications executing elsewhere.

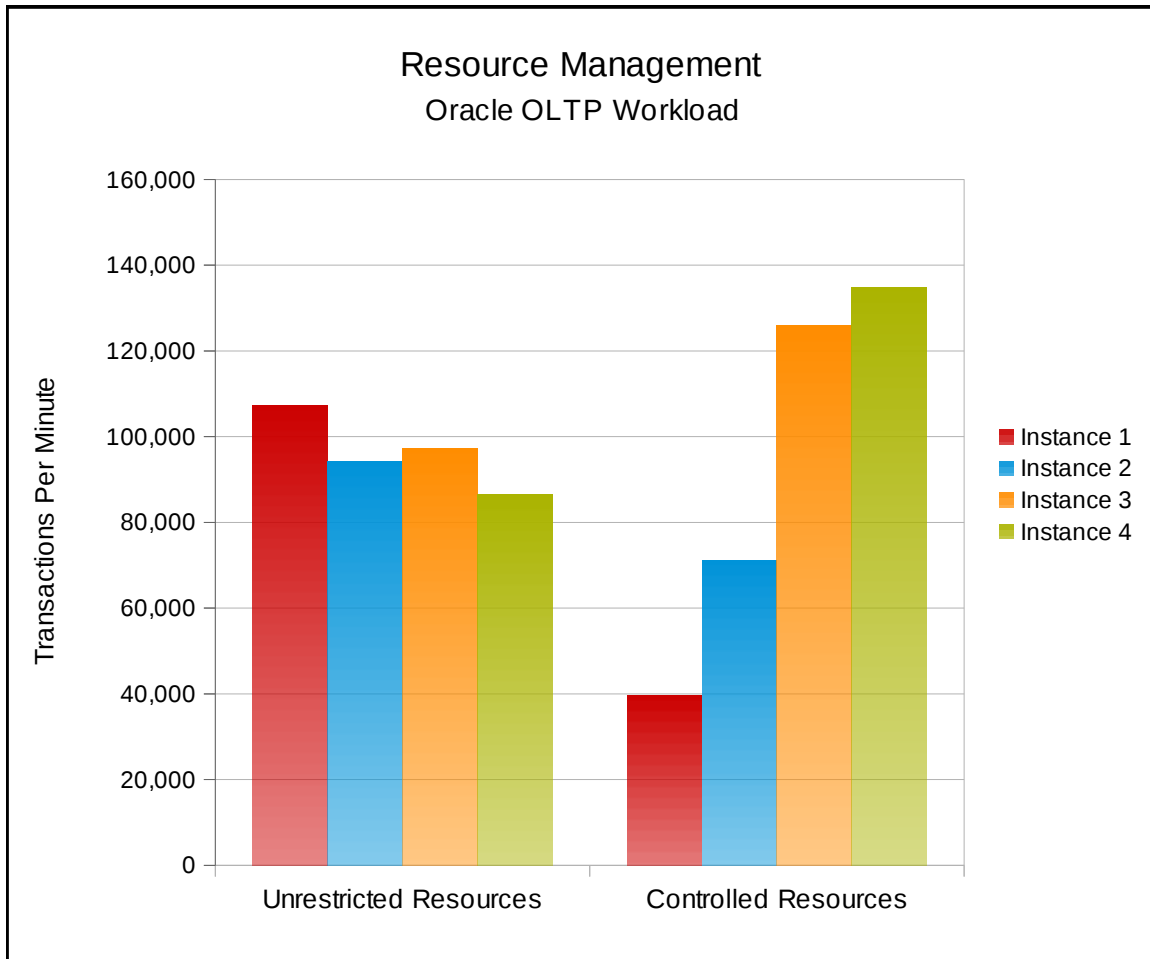
Figures 1 and 2 illustrate the test configurations comparing unrestricted instances with no defined cgroups to using cgroups to allocate 2, 4, 8, and 14 CPUs to cgroups 1 through 4 respectively.



**Figure 1: Instances Unpinned, No Cgroups**



**Figure 2: Instances Unpinned, Cgroups with 2,4,8,16 CPUs**



**Figure 3: Application Consolidation Results**

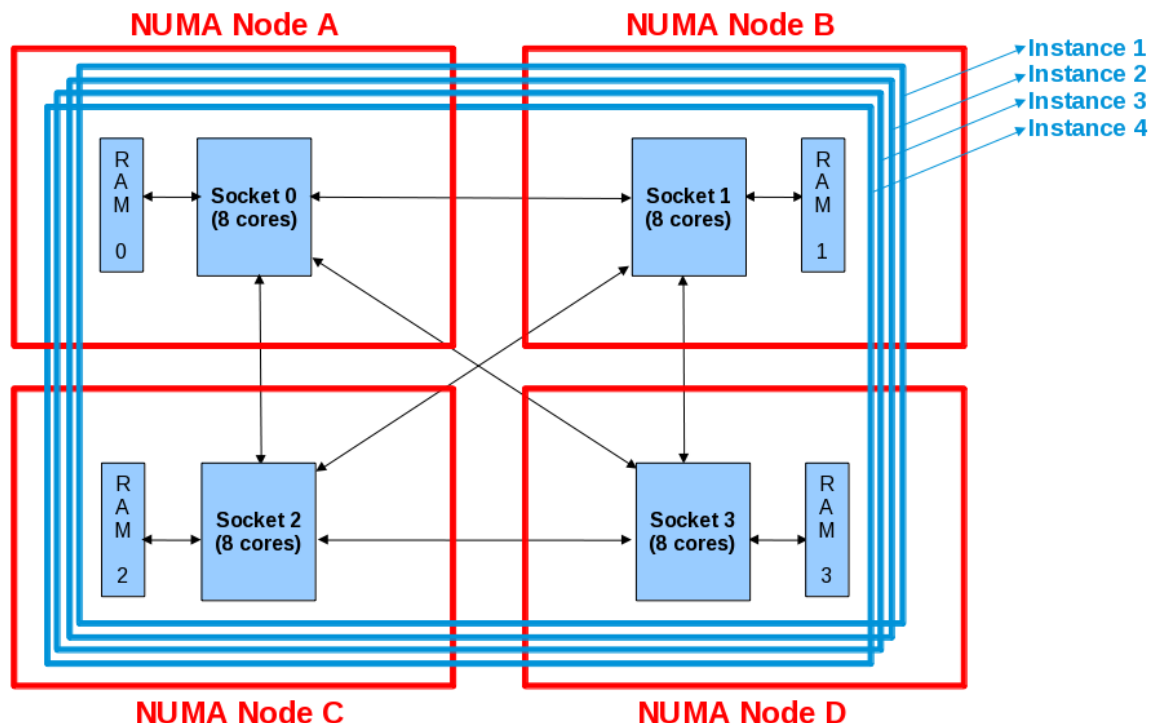
Figure 3 graphs the performance of each database instance with and without CPU resource management. The first test allowed the instances free reign to all system resources while the second allocated 2, 4, 8, and 14 CPUs to cgroups 1 through 4, respectively. This is typically done to accommodate applications or processes that require additional processing power, whether it be always or at specific times. In this case, database instances 3 and 4 are able to take advantage of the extra CPUs and reflect improved performance by 23% and 36%, respectively.

A common problem faced when consolidating applications on a larger system is that applications tend to acquire resources as required and as a result their performance can be uneven and unpredictable as observed in the unrestricted resources data in Figure 3. The ability to govern the resources allocated to each application provides system administrators the necessary control to determine how much of each resource an application is allowed which brings more predictability into the picture.



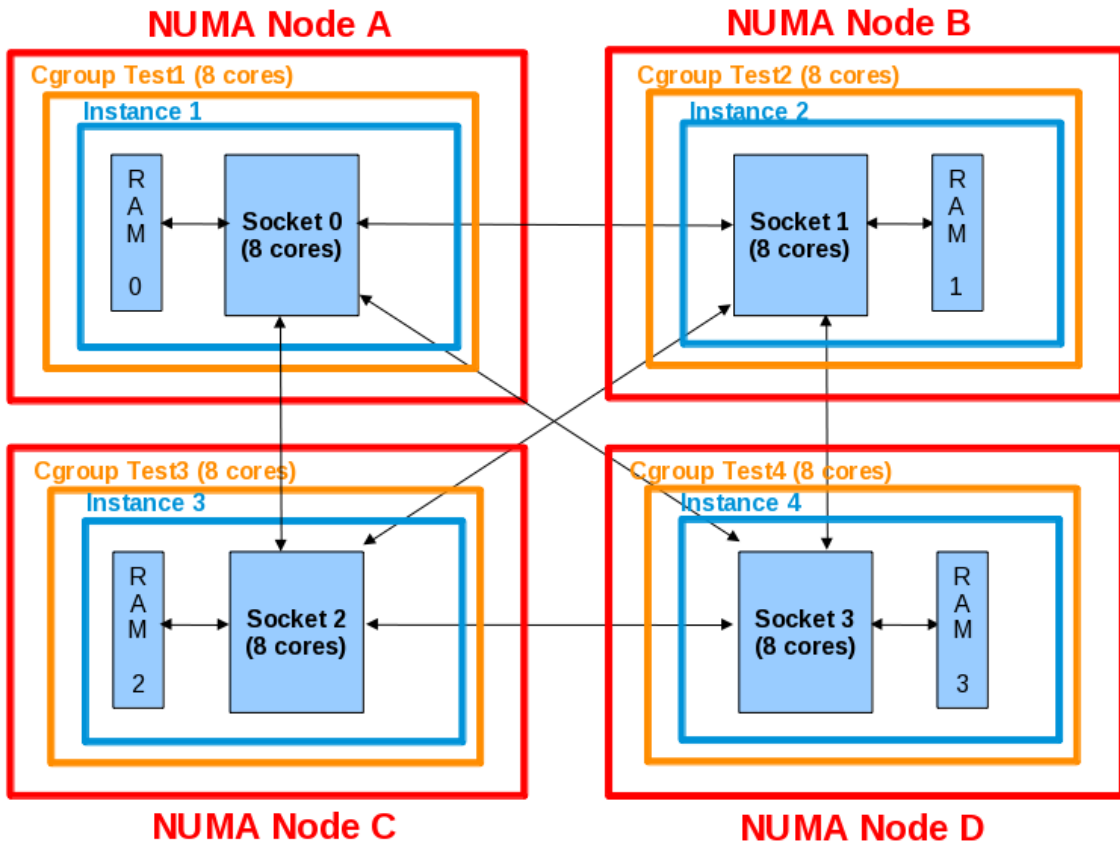
## 4.2 Performance Optimization

In this test, *cpuset* was used to govern the resources used in a multi instance database workload. This test required running four database instances, each executing an OLTP workload with a System Global Area (SGA) of 24GB. The same test was performed with and without NUMA pinning using cgroups as illustrated in Figures 4 and 5. The default is no policy (all instances share all resources).



**Figure 4: Instances Unpinned, No Cgroups**

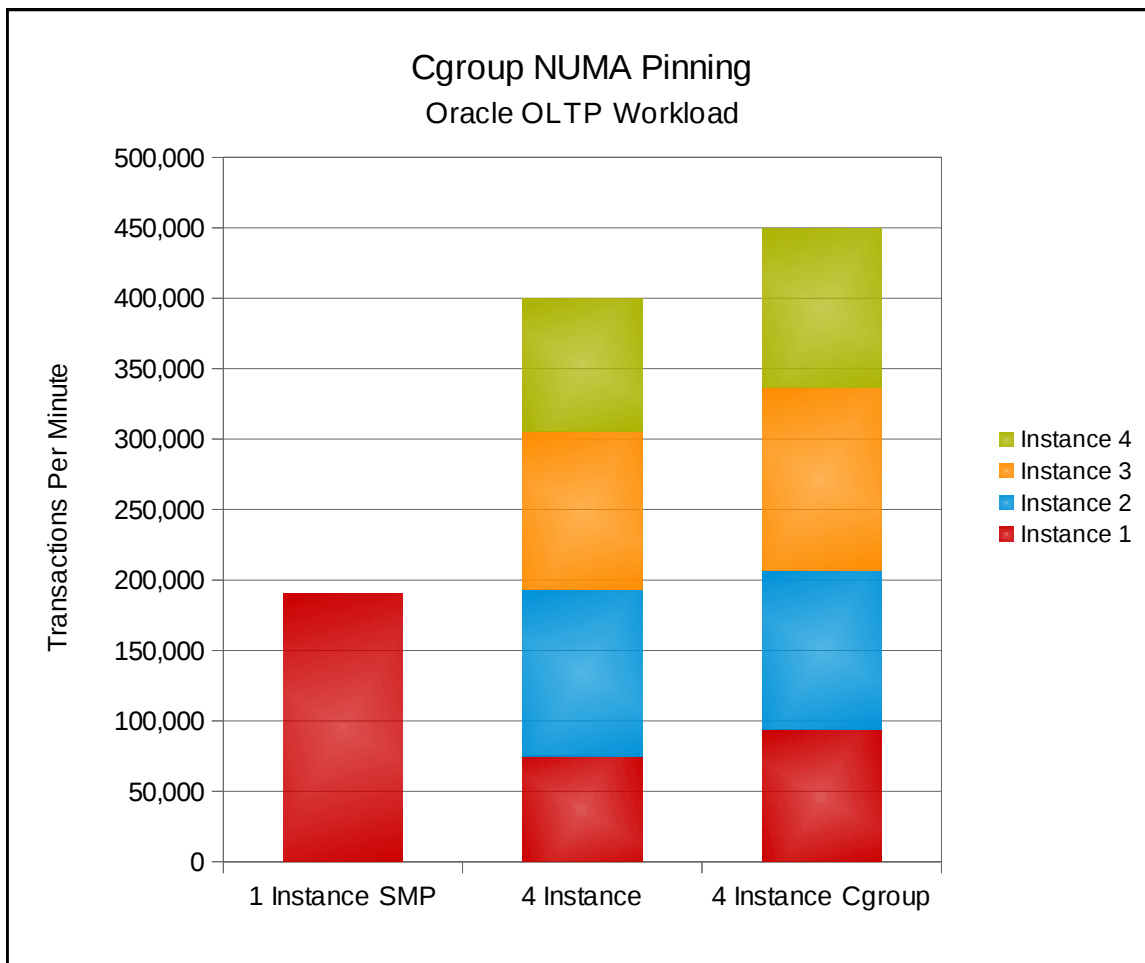
NUMA pinning locks a specific task or tasks to a NUMA node so its CPU and memory allocations are always local, avoiding the lesser bandwidth of cross-node memory transfers.



**Figure 5: NUMA Pinned Instances using Cgroups**

The multi instance testing was executed with both of the NUMA configurations described with the following results.





**Figure 6: Results of NUMA Pinning with Cgroups**

Figure 6 compares the single instance scalability limits versus those when consolidating applications for better hardware utilization. While the advantages of multi instance over single instance are obvious, cgroups may be used to take advantage of NUMA layout and improve the server consolidation experience. Each instance using only memory and CPUs local to each NUMA node reduces latency and improves the performance of other instances as much as 12.5%.

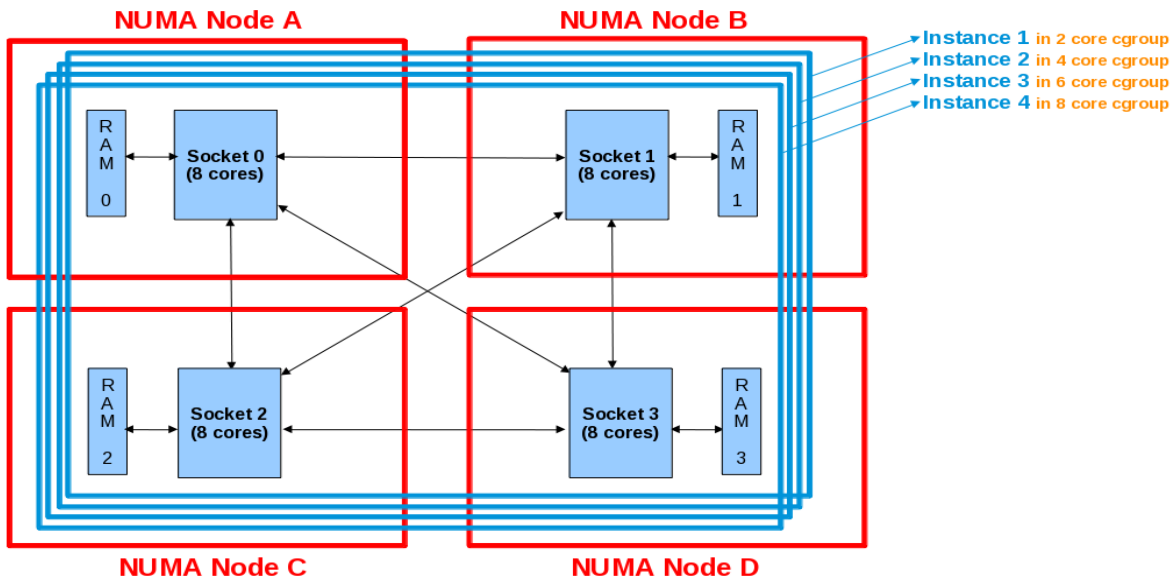
### **4.3 Dynamic Resource Management**

Because cgroups can be used to isolate applications from each other and govern the resources available to them, they are particularly useful in situations where multiple applications are pooled together on a larger server and certain applications require additional resources at specific times.



### 4.3.1 Single Instance

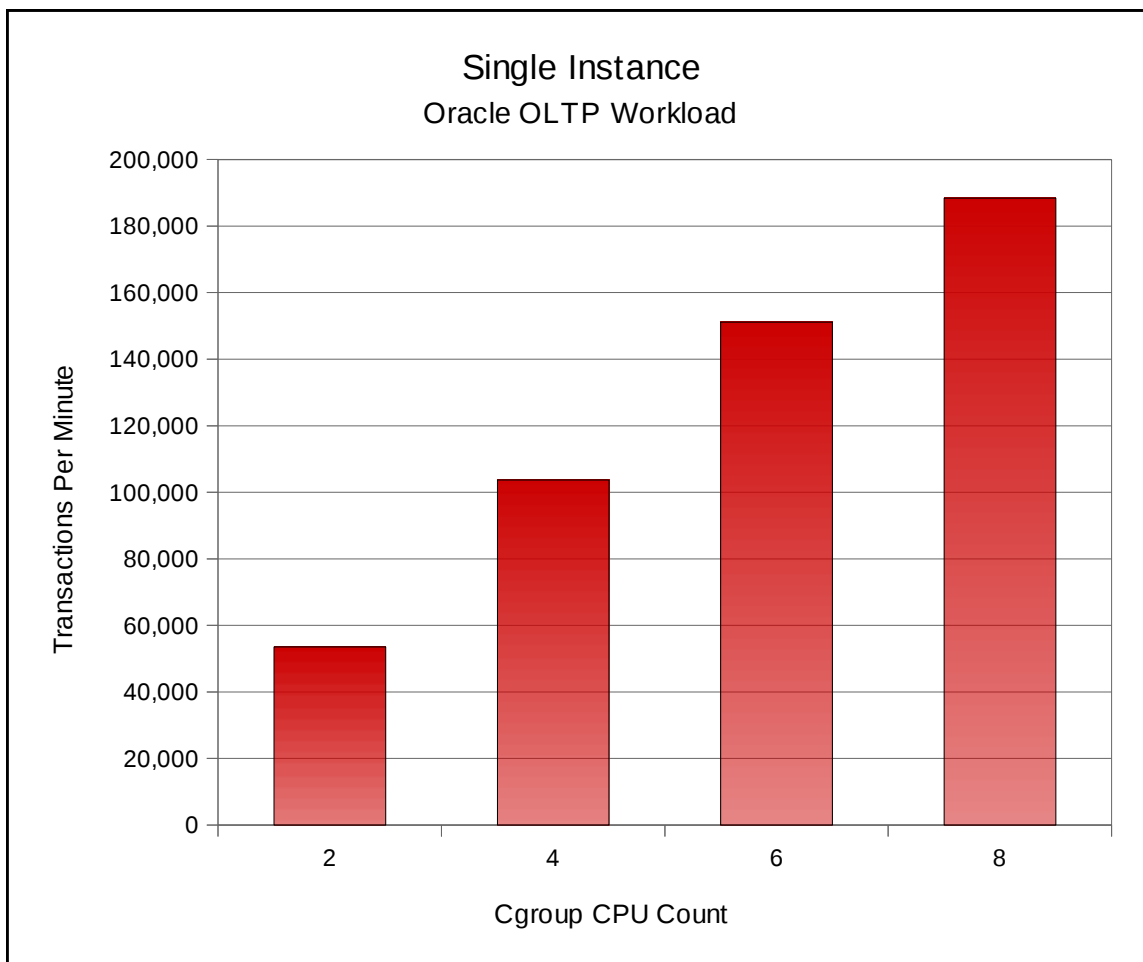
An OLTP workload was run using a 96GB SGA in the different cgroups to characterize the effects of CPU resource control. The four cgroups used were configured with 2, 4, 6, and 8 CPUs as illustrated in Figure 7.



**Figure 7: Instances Unpinned, Varying Cgroup CPU Counts**

The following commands were used to configure the cgroup CPU counts for this testing.

```
# cgset -r cpuset.cpus='0-1' test1
# cgset -r cpuset.cpus='0-3' test1
# cgset -r cpuset.cpus='0-5' test1
# cgset -r cpuset.cpus='0-7' test1
```



**Figure 8: Resource Management, Single Instance Results**

The results of the test show how cgroups can be used to scale workloads. By placing workloads in different cgroups with varying processing power, the user is able to allocate the amount of resources utilized by the workload.

### 4.3.2 Multi Instance

In this test, two instances were executed in two different cgroups. Instance 1 ran in a cgroup with four CPUs while Instance 2 ran in a cgroup with 64 CPUs as depicted in Figures 9 and 10.

The following commands were used to configure the cgroup CPU counts for this testing.

```
# cgset -r cpuset.cpus='0-3' test1
# cgset -r cpuset.cpus='4-35' test2
# sleep 600
# cgset -r cpuset.cpus='0-31' test1
# cgset -r cpuset.cpus='32-35' test2
```

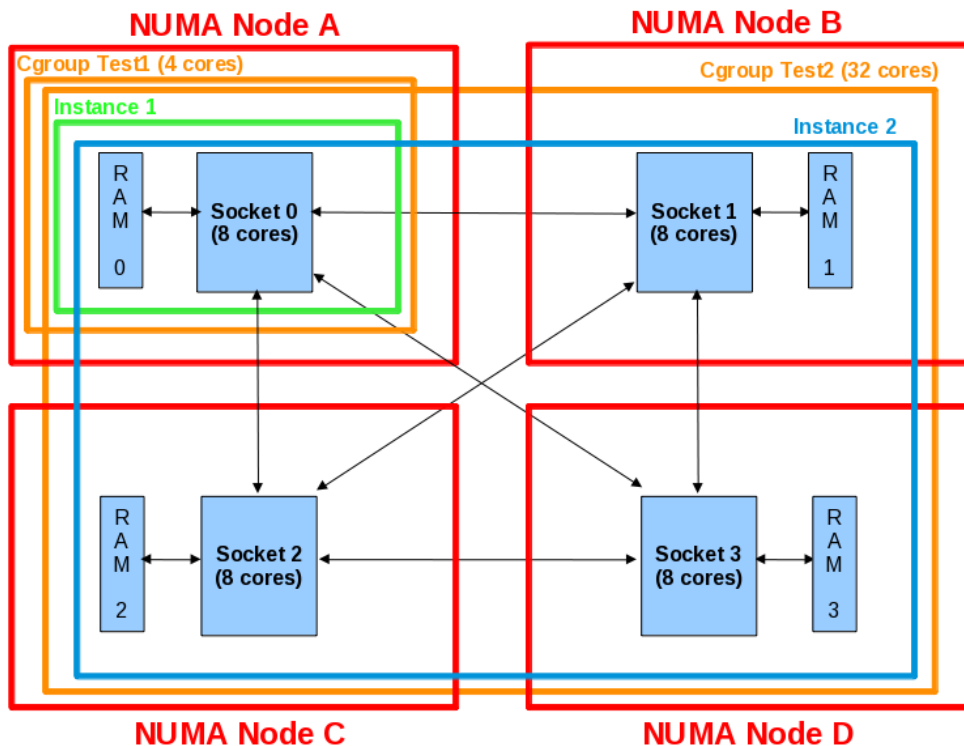


Figure 9: Dynamic Resource Management, 4 CPU vs. 32 CPU

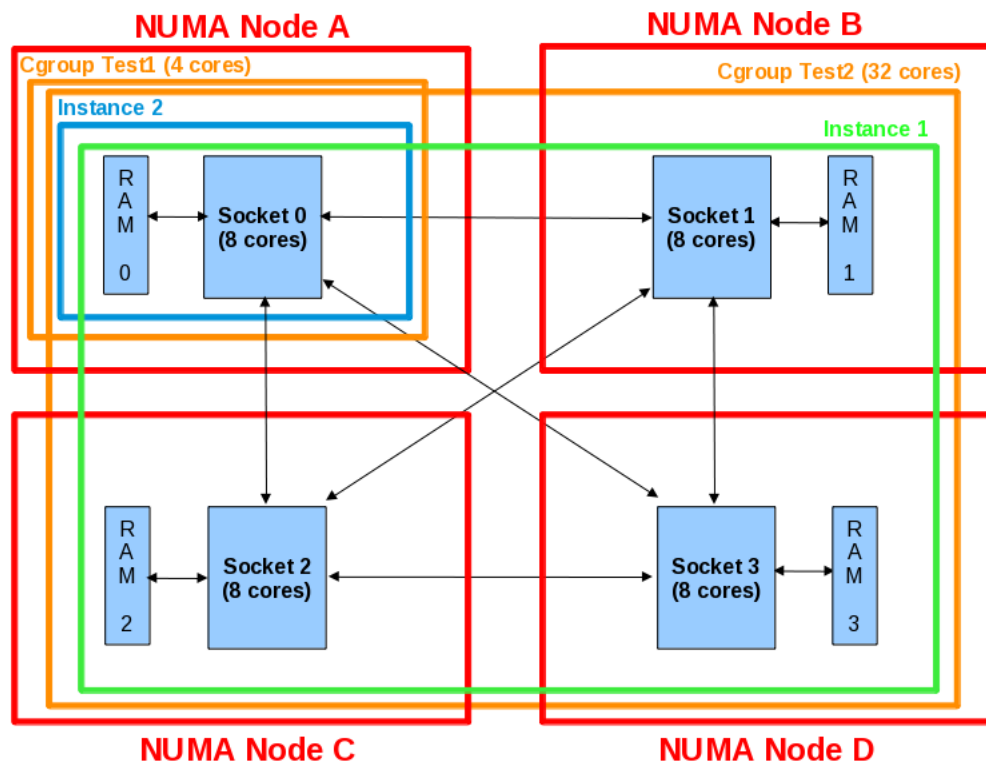
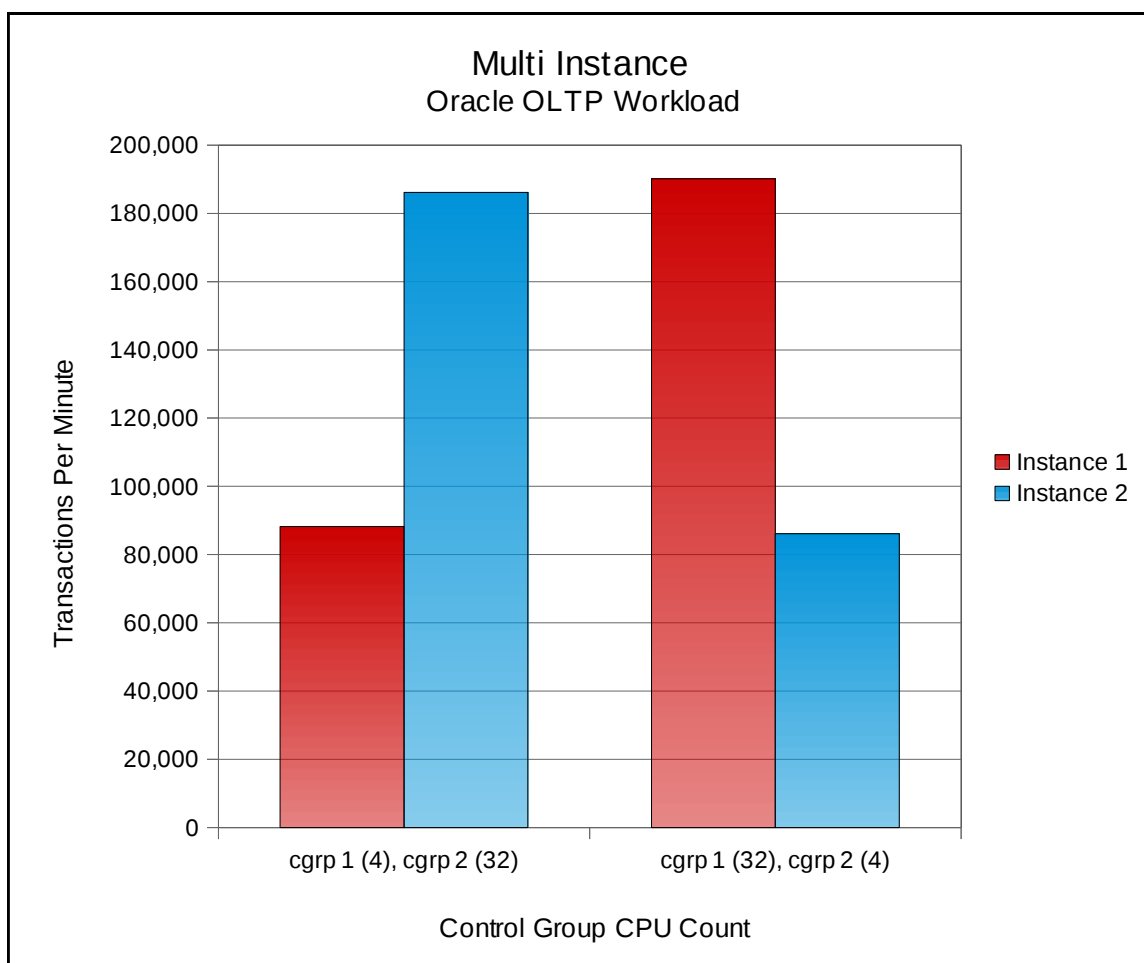


Figure 10: Dynamic Resource Management, 32 CPU vs. 4 CPU



During the test, the CPU counts of each group were switched to highlight the ability to dynamically govern the processing power of any given cgroup and its applications.



**Figure 11: Dynamic Resource Management, Multi Instance Results**

Figure 11 graphs the transaction rate of Instance 1 at approximately 88K TPM and Instance 2 at 186K. During the run, the amount of CPUs in cgroup 1 was increased to 64 while the CPUs in cgroup 2 were reduced to four. As the resources were modified, the transaction rate of the instances reflected the change with Instance 1 then running at 190K TPM and instance 2 closer to 87K.

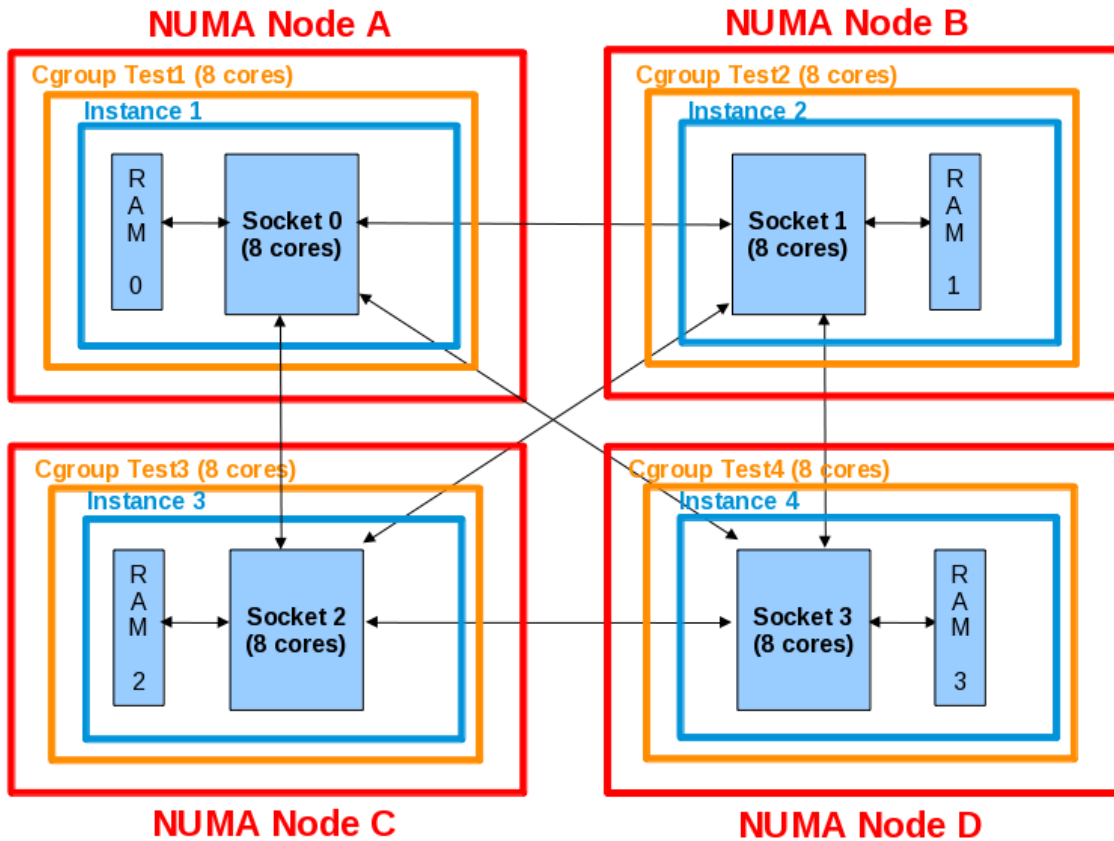
Dynamic modification of application performance is also very useful in environments where some applications must perform additional tasks (e.g., system backups, batch processing, etc.).



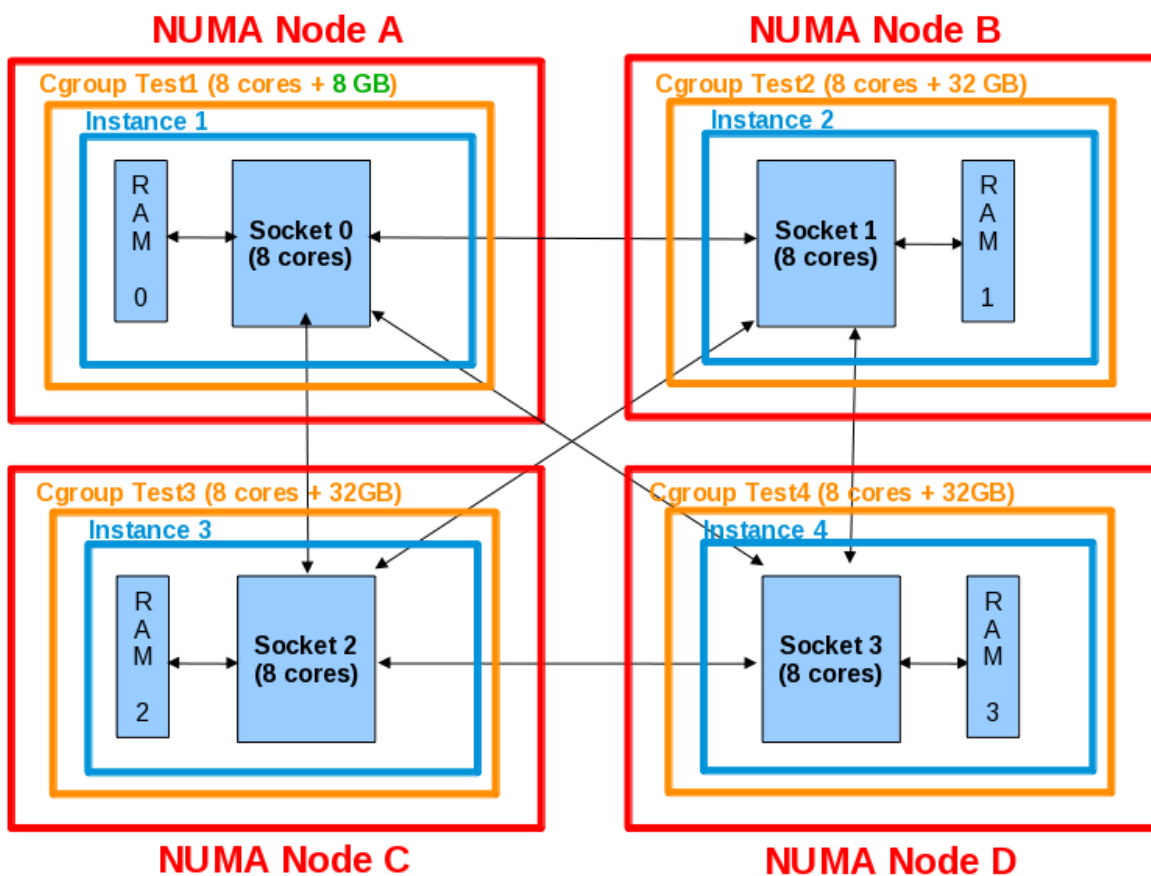
## 4.4 Application Isolation

Cgroups provide process isolation that eliminates the adverse affect misbehaving applications can have on system resources and other applications.

Figures 12 and 13 illustrate how cgroups were used to confine each database instance to a NUMA node and to reduce the memory for Instance 1.



**Figure 12: Database Instances Pinned to NUMA Nodes**



**Figure 13: Instance 1 Reduced Memory to Induce Swapping**

This test executed the same OLTP workload with each of the four database instances pinned to its own NUMA node. As listed in Table 4, the first cgroup was configured to limit the memory of database instance 1 to 8GB while the second cgroup allowed each instance the use of all the resources within that NUMA node. The memory reduction in cgroup 1 demonstrates the resource isolation that cgroups are able to provide.

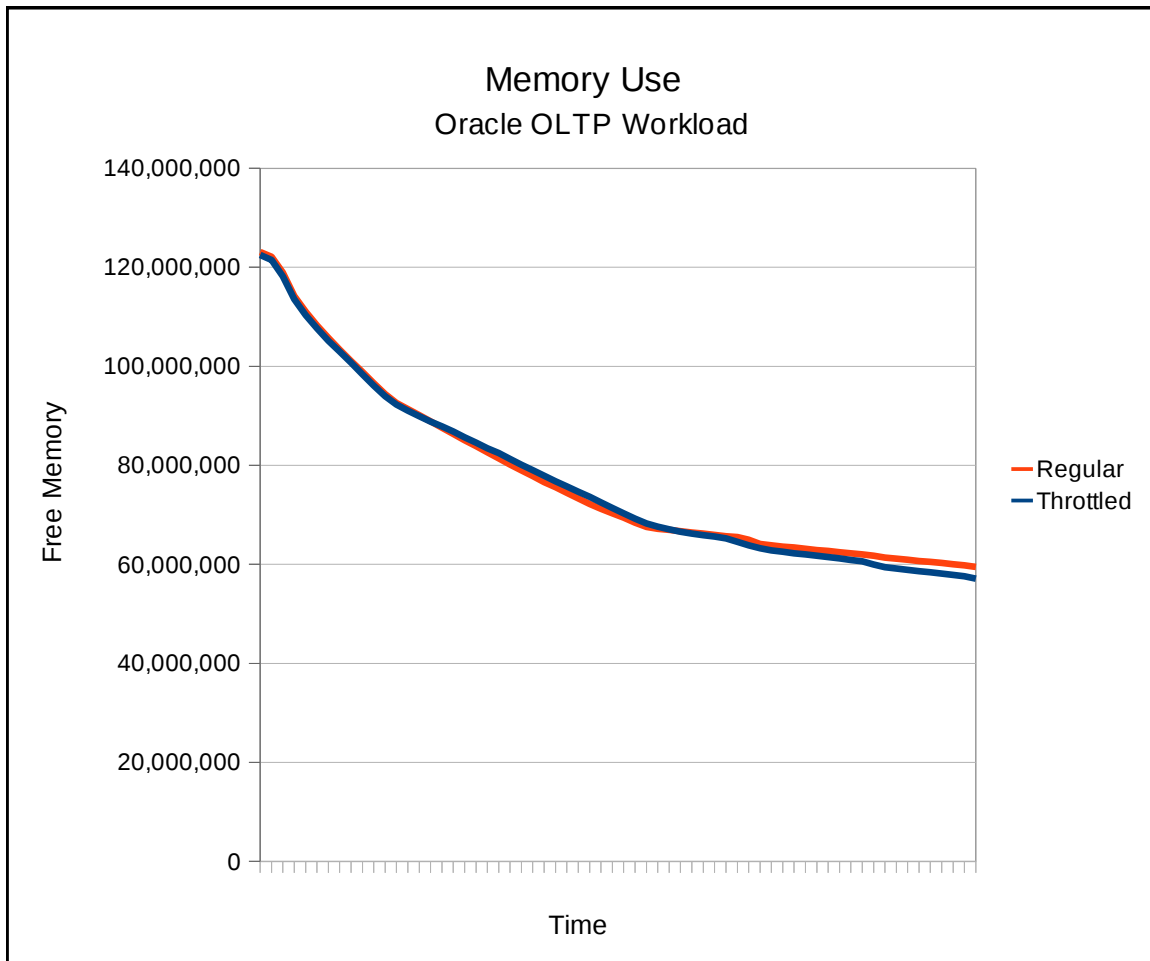
The following command was used to reduce the memory resource for cgroup 1.

```
# cgset -r memory.limit_in_bytes=8589934592 test1
```



Cgroup	NUMA Node	DB Instance	Memory (GB)
1 [test1]	0	1	8
	1	2	32
	2	3	32
	3	4	32
2 [test2]	0	1	32
	1	2	32
	2	3	32
	3	4	32

**Table 4: Isolation Cgroup Configuration**



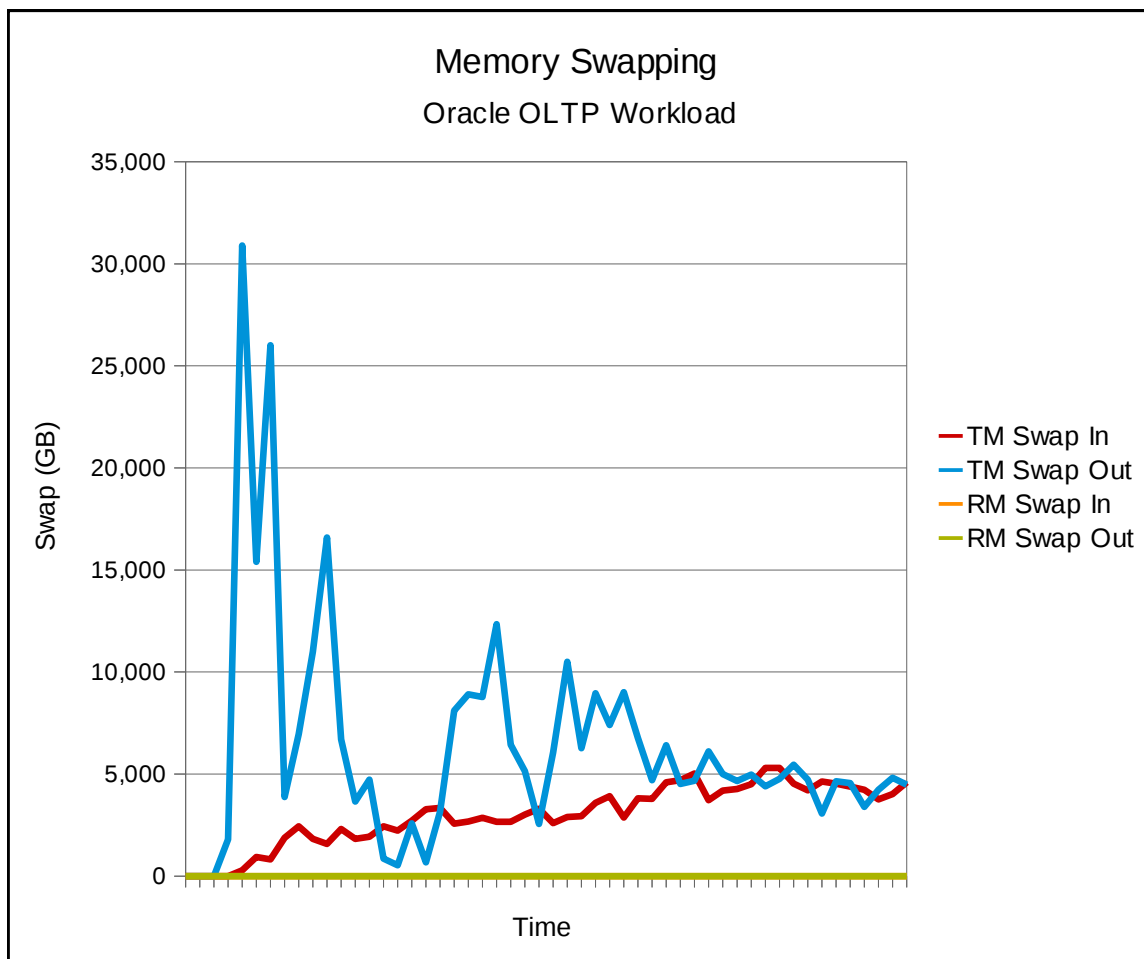
**Figure 14: Isolated Application Memory Usage**

Figure 14 graphs the free memory on the system during both the regular and reduced (throttled) memory tests. The results indicate that each test consumed the same amount of



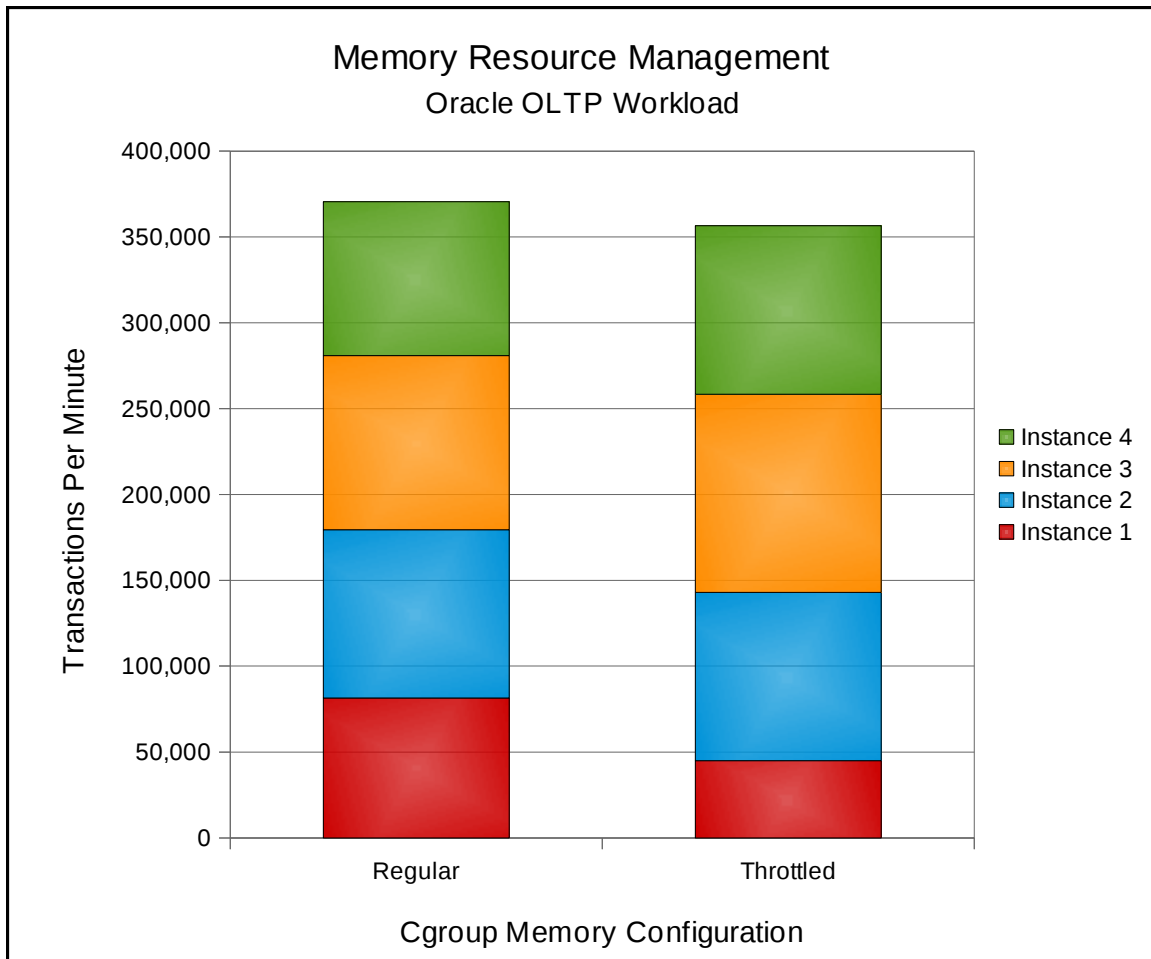


memory and that there was a minimum of 60GB of free memory available throughout the test period.



**Figure 15: Isolated Application Swapping**

Figure 15 verifies how when testing with the throttled memory (TM) of cgroup 1, database instance 1 exceeded the available memory and forced swapping to occur even though free memory was available on the system, as observed in Figure 14. The same test using other cgroups with unrestricted, or regular memory (RM), generated no swapping.



**Figure 16: Cgroup Memory Resource Management Results**

The resulting data graphed in Figure 16 verifies that even though the application in cgroup test1 exceeded the allocated memory which lead to swapping and reduced performance, it did not affect the performance of the other database instances which were able to make use of the additional memory not used by instance 1 and gained up to 12% in performance.

In this configuration, the available memory can be used by other applications. More importantly, misbehaving or rogue applications can not monopolize system resources (e.g., a typo in the SGA size can no longer wield enough force to cripple a system), bringing more critical applications down with them.



## 5 References

1. Red Hat Enterprise Linux 6 Resource Management Guide  
[http://docs.redhat.com/docs/en-US/Red\\_Hat\\_Enterprise\\_Linux/6/html/Resource\\_Management\\_Guide/index.html](http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/index.html)
2. Linux Kernel Source Documentation - Cgroups  
<http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt>