



Best Practices for Running Oracle Databases in Solaris™ Containers

Ritu Kamboj and Fernando Castano

September 2005

©2005 Sun Microsystems, Inc. 4150 Network Circle Santa Clara, CA 95054 U.S.A.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California.

Sun, Sun Microsystems, the Sun logo, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

U.S. Government Rights Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED AS IS AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Unless otherwise licensed, use of this software is authorized pursuant to the terms of the license found at: http://developers.sun.com/berkeley_license.html

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie.

Sun, Sun Microsystems, le logo Sun, et Solaris sont des marques de fabrique ou des marques déposées, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

CETTE PUBLICATION EST FOURNIE EN L'ETAT ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Table of Contents

1	Executive Summary	4
2	Document Scope	4
3	Overview	4
3.1	Solaris Containers	4
3.2	Solaris Zones Partitioning Technology	5
3.3	Solaris Resource Manager	6
3.3.1	Workload Identification	6
3.3.2	Resource Controls	7
3.3.3	CPU and Memory Management	8
4	Oracle License Model for Solaris Containers	8
5	Creating a Container	9
5.1	Requirements	9
5.2	Creating a Resource Pool	10
5.3	Creating a Non-Global Zone	10
6	Special Considerations	11
6.1	Devices in Containers	12
6.2	File Systems in Containers	12
6.3	Backup With Recovery Manager	14
6.4	Volume Management	14
6.5	CPU Visibility	15
7	Oracle Features Not Available in Containers	16
7.1	Oracle RAC	16
7.2	Solaris Dynamic Intimate Shared Memory (DISM)	17
8	Appendix	18
8.1	Appendix 1: Script to Create a Container	18
8.1.1	README.txt	19
8.1.2	setenv.sh	20
8.1.3	zone cmd template.txt	21
8.1.4	pool cmd template.txt	21
8.1.5	create zone cmd.sh	22
8.1.6	create pool cmd.sh	22

8.1.7 create_container.sh	23
8.2 Appendix 2: Setting System V IPC Kernel Parameters	25
9 References	29
10 About the Authors	30

1 Executive Summary

This document provides an overview of Solaris Containers in the Solaris 10 Operating System (OS), with guidelines for running a non-RAC Oracle database in a container. Oracle 9i R2 and 10g R1 databases (RAC and non-RAC) have been certified to run in a global zone. This document concentrates on running a non-RAC Oracle database in a container in the Solaris OS, and it explains in detail the process of creating a non-global zone appropriate for deploying an Oracle database. Additionally, it captures special considerations for running a non-RAC Oracle database in a container in the Solaris OS.

For the remainder of this document a "Solaris Container in the Solaris 10 OS" will be referred to as a "Solaris container" or just a "container" and it will be assumed that it is associated to a non-global zone unless explicitly stated otherwise. Also, a "non-RAC Oracle database" will be referred to simply as an "Oracle database".

2 Document Scope

A new licensing agreement between Sun and Oracle recognizes Solaris 10 capped containers as hard partitions (see references [6] and [7]). The scope of this document is to define a container in the Solaris 10 OS appropriate for running an Oracle database. In addition, it captures limitations and special cases of running Oracle databases in containers.

It is beyond the scope of this document to explain how Solaris Containers technology can be used to consolidate multiple Oracle database instances in separate containers on the same system. See references [1] and [3] for detailed information about using Solaris Containers technology for server consolidation.

3 Overview

This section provides a brief overview of Solaris Containers technology. It also gives an introduction to the Solaris Zones feature and Solaris Resource Manager, which are the two major components of Solaris Containers (for detailed information about these

technologies, see references [2] and [4]).

3.1 Solaris Containers

Solaris Containers are designed to provide a complete, isolated and secure runtime environment for applications. This technology allows application components to be isolated from each other using flexible, software-defined boundaries. Solaris Containers are designed to provide fine-grained control over resources that the application uses, allowing multiple applications to operate on a single server while maintaining specified service levels.

Solaris Containers are a management construct intended to provide a unified model for defining and administering the environment within which a collection of Solaris processes executes. Solaris Containers use Solaris Resource Manager (SRM) features along with Solaris Zones to deliver a virtualized environment that can have fixed resource boundaries for application workloads.

3.2 Solaris Zones Partitioning Technology

Solaris Zones, a component of the Solaris Containers environment, is a software partitioning technology that virtualizes operating system services and provides an isolated and secure environment for running applications. Solaris Zones are ideal for environments that consolidate multiple applications on a single server.

There are two types of zones: global zones and non-global zones. The underlying OS, which is the Solaris instance booted by the system hardware, is called the global zone. There is only one global zone per system, which is both the default zone for the system and the zone used for system-wide administrative control. One or more non-global zones can be created by an administrator of a global zone. Once created, these non-global zones can be administered by individual non-global zone administrators, whose privileges are confined to that non-global zone.

Two types of non-global zones can be created using different root file system models: sparse and whole root. The sparse root zone model optimizes the sharing of objects by only installing a subset of the root packages and using read-only loopback file system to gain access to other files. In this model, by default, the directories `/lib`, `/platform`,

`/sbin`, and `/usr` will be mounted as loopback file systems. The advantages of this model are improved performance due to efficient sharing of executables and shared libraries, and a much smaller disk footprint for the zone itself. The whole root zone model provides for maximum configurability by installing the required packages and any selected optional zones into the private file systems of the zone. The advantages of this model include the ability for zone administrators to customize their zones' file system layout and add arbitrary unbundled or third-party packages.

Solaris Zones provide the standard Solaris interfaces and application environment. They do not impose a new ABI or API. In general, applications do not need to be ported to Solaris Zones. However, applications running in non-global zones need to be aware of non-global zone behavior, in particular:

- All processes running in a zone have a reduced set of privileges, which is a subset of the privileges available in the global zone. Processes that require a privilege not available in a non-global zone can fail to run, or in a few cases fail to achieve full performance (for example, when Oracle is configured to use DISM in a container).

- Each non-global zone has its own logical network and loopback interface. Bindings between upper-layer streams and logical interfaces are restricted such that a stream may only establish bindings to logical interfaces in the same zone. Likewise, packets from a logical interface can only be passed to upper-layer streams in the same zone as the logical interface.
- Non-global zones have access to a restricted set of devices. In general, devices are shared resources in a system. Therefore, restrictions within zones are put in place so that security is not compromised.

3.3 Solaris Resource Manager

By default, the Solaris OS provides all workloads running on the system equal access to all system resources. This default behavior of the Solaris OS can be modified by Solaris Resource Manager, which provides a way to control resource usage.

SRM provides the following functionality:

- A method to classify a workload, so the system knows which processes belong to a given workload.
- The ability to measure the workload to assess how much of the system resources the workload is actually using.
- The ability to control the workloads so they do not interfere with one another and also get the required system resources to meet predefined service-level agreements.

SRM provides three types of workload control mechanisms:

- The *constraint mechanism*, which allows the Solaris system administrator to limit the resources a workload is allowed to consume.
- The *scheduling mechanism*, which refers to the allocation decisions that accommodate the resource demands of all the different workloads in an under-committed or over-committed scenario.
- The *partitioning mechanism*, which ensures that pre-defined system resources are assigned to a given workload.

3.3.1 Workload Identification

Projects:

Projects are a facility that allow the identification and separation of workloads. A workload can be composed of several applications and processes belonging to several different groups and users. The identification mechanism provided by projects serves as a *tag* for all the processes of a workload. This identifier can be shared across multiple machines through the project name service database. The location of this database can be in files, NIS, or LDAP, depending on the definition of *projects* database source in the `/etc/nsswitch.conf` file. Attributes assigned to the projects are used by the resource control mechanism to provide a resource administration context on a per-project basis.

Tasks:

Tasks provide a second level of granularity in identifying a workload. A task collects a group of processes into a manageable entity that represents a workload component. Each login creates a new task that belongs to the project, and all the processes started during that login session belong to the task. The concept of projects and tasks has been incorporated in several administrative commands like `ps`, `pgrep`, `pkill`, `prstat`, and `cron`.

3.3.2 Resource Controls

Resource usage of workloads can be controlled by placing bounds on resource usage. These bounds can be used to prevent a workload from over-consuming a particular resource and interfering with other workloads. The Solaris Resource Manager provides a resource control facility to implement constraints on resource usage.

Each resource control is defined by the following three values:

- Privilege level
- Threshold value
- Action that is associated with the particular threshold

The privilege level indicates the privilege needed to modify the resource. It must be one of the following three types:

- Basic, which can be modified by the owner of the calling process

- Privileged, which can be modified only by privileged (superuser) callers
- System, which is fixed for the duration of the operating system instance

The threshold value on a resource control constitutes an enforcement point where actions can be triggered. The specified action is performed when a particular threshold is reached. Global actions apply to resource control values for every resource control on the system. Local action is taken on a process that attempts to exceed the control value. There are three types of local actions:

- none: No action is taken on resource requests for an amount that is greater than the threshold.
- deny: Deny resource requests for an amount that is greater than the threshold.
- signal: Enable a global signal message action when the resource control is exceeded.

For example, `task.max-lwp=(privileged, 10, deny)` would tell the resource control facility to deny more than 10 lightweight processes to any process in that task.

3.3.3 CPU and Memory Management

SRM enables the end user to control the available CPU resources and physical memory consumption of different workloads on a system by providing *Fair Share Scheduler (FSS)* and *Resource Capping Daemon* facilities, respectively.

Fair Share Scheduler

The default scheduler in the Solaris OS provides every process equal access to CPU resources. However, when multiple workloads are running on the same system one workload can monopolize CPU resources. Fair Share Scheduler provides a mechanism to prioritize access to CPU resources based on the importance of the workload.

With FSS the importance of a workload is expressed by the number of shares the system administrator allocates to the project representing the workload. Shares define the relative importance of projects with respect to other projects. If project A is deemed twice as important as Project B, project A should be assigned twice as many shares as project B.

It is important to note that FSS only limits CPU usage if there is competition for CPU resources. If there is only one active project on the system, it can use 100% of the system CPUs resources, regardless of the number of shares assigned to it.

Resource Capping Daemon

The resource capping daemon (`rcapd`) can be used to regulate the amount of physical memory that is consumed by projects with resource caps defined. The `rcapd` daemon repeatedly samples the memory utilization of projects that are configured with physical memory caps. The sampling interval is specified by the administrator. When the system's physical memory utilization exceeds the threshold for cap enforcement and other conditions are met, the daemon takes action to reduce the memory consumption of projects with memory caps to levels at or below the caps.

Please note that the `rcapd` daemon cannot determine which pages of memory are shared with other processes or which are mapped multiple times within the same process. Hence, it is not recommended that shared memory-intensive applications, like Oracle databases, run under projects that use `rcapd` to limit physical memory usage.

4 Oracle License Model for Solaris Containers

Oracle now recognizes capped Solaris 10 Containers as licensable entities, known as hard partitions. Oracle customers running Oracle in a Solaris 10 environment can now license only the CPUs or cores that are in a capped Solaris container as described below.

Oracle licensing policy defines hard partitioning as "a physical subset of a server that acts like a self-contained server" (for more details see reference [2]). The following example shows how an 8-processor system can be partitioned into a 3-processor sub-system using Solaris Containers technology in the Solaris 10 OS.

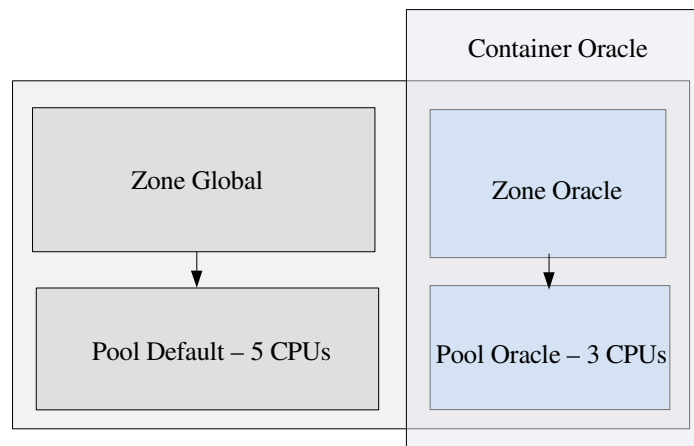


Figure 1: Example of Container for Oracle in the Solaris 10 OS

To create a Solaris 10 container that fits the licensing requirements set by Oracle, the Solaris system administrator needs to create a resource pool with the desired number of CPUs or cores and bind a zone to this resource pool. The license is driven by the number of CPUs or cores in this pool.

5 Creating a Container

This section provides instructions for creating a Solaris 10 container appropriate for installing and running Oracle. These instructions have been followed in the sample

scripts documented in Appendix 1, which provide a convenient way of creating such containers.

5.1 Requirements

1. Ensure that the file system in which the root directory for the container will be placed has at least 6 GB of physical disk space. This will be enough to create the container and install Oracle.
2. Identify the physical interface that will be used to bring up the virtual interface for the container. To find the physical interfaces available in the global container execute the command:

```
/usr/sbin/ifconfig -a
```

Examples of common interfaces are `ce0`, `bge0` or `hme0`.

3. Obtain an IP address and a hostname for the container. This IP address must be in the same subnet as the IP assigned to the physical interface selected in the previous step.
4. Ensure that the netmask for the IP address of the container can be resolved in the global container according to the databases used in the `/etc/nsswitch.conf` file. If this is not the case, update the file `/etc/netmasks` in the global container with the netmask desired for the subnet to which the IP address belongs.
5. Determine the number of CPUs to be reserved for the container. To find the number of CPUs available in the default pool execute the command `poolstat`. The default pool will indicate the number of CPUs available. Keep in mind that the default pool must always have at least one CPU.

5.2 Creating a Resource Pool

The resource pool can be created by the root user in the global container following these steps:

1. Enable the pool facility with this command:

```
pooladm -e
```

2. Use the file `pool_cmd_template.txt` provided in the appendix as a template to create a command file. Replace the strings `PSET_NAME`, `NUM_CPUS_MIN`,

NUM_CPUS_MAX and POOL_NAME with appropriate values. This command file has instructions to create a resource pool and a processor set (`pset`), and then to associate the `pset` to the resource pool.

3. If the default configuration file `/etc/pooladm.conf` does not exist, create it by executing this command:

```
pooladm -s /etc/pooladm.conf
```

4. Create the resource pool by executing this command:

```
poolcfg -f pool_commands.txt
```

where `pool_commands.txt` is the file you created two steps before.

5. Instantiate the changes made to the static configuration by executing the command:

```
pooladm -c
```

6. Validate your configuration by executing this command:

```
pooladm -n
```

5.3 Creating a Non-Global Zone

Once the resource pool is available, a non-global zone can be created and bound to it. For the purposes of installing and running Oracle the non-global zone can have either a whole root model or a sparse root model. Unless it conflicts with specific requirements we recommend using the sparse root model. A non-global Solaris zone can be created as follows:

1. Create as root a directory where the root of the non-global zone will be placed (for example, `/export/home/myzone`) and set the access permissions to 700. The name of this directory should match the name of the zone (`myzone`, in this example).
2. Unless special instructions are added, the directory `/usr` in the container will be a loopback file system (`lofs`). This means that the container will mount in read-only mode `/usr` from the global container into `/usr` of its own file system's tree. By default, the Oracle installer requires root to create files in `/usr/local` directory. Since `/usr` will be read-only in the container, we will create a special mount point in `/usr/local` to allow root to create such files in the container. Check if the directory `/usr/local` exists in the global container,

and if it is not present, create it.

3. Create a directory in the global container to mount `/usr/local` in the container. The recommendation is to create it in `/opt/<ZONE_NAME>/local`.
4. Use the `zone_cmd_template.txt` file in Appendix 1 as a model to create a command file to create the zone and bind it to the resource pool previously created. Replace the strings `ZONE_DIR`, `ZONE_NAME`, `POOL_NAME`, `NET_IP`, and `NET_PHYSICAL` with appropriate values. Note that this template file expects the special mount point for `/usr/local` to be found in `/opt/<ZONE_NAME>/local`. In this file, the command `create` is used to create a sparse root. Replacing this command with `create -b` would create a whole root. Also, in this file the zone is bound to the pool with the command `set pool=POOL_NAME`. Another way to bind a pool to a zone is to use the command `poolbind(1M)` once the pool and the zone have been created.
5. Create the zone by executing as root this command:

```
zonecfg -z <ZONE_NAME> -f zone_commands.txt
```

where `zone_commands.txt` is the file you create in the previous step.

6. Install the zone by executing as root this command:

```
zoneadm -z <ZONE_NAME> install
```

7. Boot the zone with this command:

```
zoneadm -z <ZONE_NAME> boot
```

8. Finish the configuration of your container with this command:

```
zlogin -C <ZONE_NAME>
```

6 Special Considerations

In this section we point out some special considerations when running an Oracle database inside a container.

6.1 Devices in Containers

To guarantee that security and isolation are not compromised, certain restrictions regarding devices are placed on non-global zones. Here are these restrictions:

- By default, only a restricted set of devices (which consist primarily of pseudo

devices) such as `/dev/null`, `/dev/zero`, `/dev/poll`, `/dev/random`, and `/dev/tcp`, are accessible in the non-global zone.

- Devices that expose system data like `dtrace`, `kmem`, and `ksyms` are not available in non-global zones.
- By default, physical devices are also not accessible by containers.

The zone administrator can make physical devices available to non-global zones. It is the administrator's responsibility to ensure that the security of the system is not compromised by doing so, mainly for two reasons:

- 1) Placing a physical device into more than one zone can create a covert channel between zones.
- 2) Global zone applications that use such a device risk the possibility of compromised data or data corruption by a non-global zone.

The global zone administrator can use the `add device` sub-command of `zonecfg` to include additional devices in non-global zone. For example to add the block device `/dev/dsk/c1t1d0s0` to the non-global zone, the administrator executes the following commands:

```
zonecfg -z my-zone
zonecfg:my-zone> add device
zonecfg:my-zone:device> set match=/dev/dsk/c1t1d0s0
zonecfg:my-zone:device> end
zonecfg:my-zone>exit
zoneadm -z my-zone reboot
```

All the slices of `/dev/dsk/c1t1d0` could be added to the non-global zone by using `/dev/dsk/c1t1d0*` in the `match` command. This same procedure can be used for character devices (also known as raw devices) or any other kind of device.

If you plan to install an Oracle database in a non-global zone by using Oracle installation CDs, you will need to make the CD-ROM device visible to the non-global zone. To do this you can either loopback mount the `/cdrom` directory from the global zone or export the physical device (which is discouraged). For details about how to gain access to the CD-ROM device from a non-global zone, see reference [8].

6.2 File Systems in Containers

Each zone has its own file system hierarchy, rooted at a directory known as zone root. Processes in the zones can access only files in the part of the hierarchy that is located under the zone root.

Here we present four different ways of mounting a file system from a global zone to a non-global zone:

1. Create a file system in a global zone and mount it in a non-global zone as a loopback file system (lofs).

- Log in as global zone administrator.
- Create a file system in global zone

```
global# newfs /dev/rdisk/clt0d0s0
```

- Mount the file system in the global zone

```
global#mount /dev/dsk/clt0d0s0 /mystuff
```

- Add the file system of type lofs to the non-global zone

```
global#zonecfg -z my-zone
zonecfg:my-zone> add fs
zonecfg:my-zone:fs>set dir=/usr/mystuff
zonecfg:my-zone:fs> set special=/mystuff
zonecfg:my-zone:fs>set type=lofs
zonecfg:my-zone:fs>end
```

2. Create a file system in the global zone and mount it to the non-global zone as UFS.

- Log in as global zone administrator.
- Create a file system in the global zone

```
global# newfs /dev/rdisk/clt0d0s0
```

- Add the file system of type ufs to the non-global zone

```
global# zonecfg -z my-zone
zonecfg:my-zone>add fs
zonecfg:my-zone:fs> set dir=/usr/mystuff
zonecfg:my-zone:fs>set special=/dev/dsk/clt0d0s0
zonecfg:my-zone:fs>set raw=/dev/rdisk/clt0d0s0
zonecfg:my-zone:fs>set type=ufs
zonecfg:my-zone:fs>end
```

3. Export a device from a global zone to a non-global zone and mount it from the non-global zone.

- Log in as global zone administrator.
- Export a raw device to the non-global zone

```
global # zonecfg -z my-zone
zonecfg:my-zone> add device
zonecfg:my-zone:device> set match=/dev/rdisk/c1t0d0s0
zonecfg:my-zone:device>end
zonecfg:my-zone>add device
zonecfg:my-zone:device>set match=/dev/dsk/c1t0d0s0
zonecfg:my-zone:device>end
```

- Log in as root in non-global zone.
- Create a file system in the non-global zone:

```
my-zone# newfs /dev/rdisk/c1t0d0s0
```

- Mount the file system in the non-global zone:

```
my-zone# mount /dev/dsk/c1t0d0s0 /usr/mystuff
```

4. Mount a UFS file system directly into the non-global zone's directory structure. This assumes that the device is already made available to the non-global zone.

- Log in as non-global zone administrator.
- Mount the device in non-global zone:

```
my-zone#mount /dev/dsk/c1t1d0s0 /usr/mystuff
```

6.3 Backup With Recovery Manager

Recovery manager (`rman`) is an Oracle utility available in Oracle database version 8.0 and higher used to back up, restore, and recover database files. To store backup on tape, RMAN requires a media manager. A media manager is third-party software that manages sequential media such as tape drives to back up and restore database files. Although `rman` works fine on non-global zones, the media managers may not be certified to work on non-global zones. This would potentially imply that currently the `rman` utility may only be used to back up and restore database files on disks. The backup taken by `rman` on disks can however be copied over to tape devices from the global zone.

6.4 Volume Management

Volume managers are often third-party products, so their usability in containers cannot be generalized. As of this writing, volume managers cannot be installed or managed from containers. Currently, the recommended approach is to install and manage volume managers from global containers. Once the devices, file systems, and volumes are created in the global container, they can be made available to the container by using `zonecfg` subcommands.

In the case of VERITAS, the following features are **not** supported in containers:

- Admin `ioctl`s
- Administration commands
- VERITAS Volume Manager software
- VERITAS Storage Migrator (VSM)
- VERITAS File System (VxFS)/VERITAS Federated Mapping Service (VxMS)
- Quick I/O and CQIO
- Cluster File System

The following VERITAS features are supported in non-global zones:

- VERITAS file systems
- Access to VxFS in the global zone from the non-global zone of a `lofs` file system
- Access to ODM files from non-global zones
- Concurrent I/O with files from non-global zones

VERITAS commands will be accessible in non-global zones, but they have been modified to ensure they are not executed in a non-global zone. When VERITAS Volume Manager commands detect non-global zone execution, the following error message will be presented:

VxVM command_xxx ERROR msg_id: Please execute this operation in global zone.

In a similar way, Solaris Volume Manager (SVM) should also be installed and managed from global zones in containers. Once the storage has been configured in the desired way from the global zone, the metadevices can be made available to the non-global zones.

6.5 CPU Visibility

Users in a container can expect a virtualized view of the system with respect to CPU visibility when the zone is bound to a resource pool. In these cases the zone will only see those CPUs associated with the resource pool it is bound to.

The Oracle database application mainly calls `pset_info(2)` and `sysconf(3c)` with the `_SC_NPROCESSORS_ONLN` argument to procure the number of CPUs it has available. Based on this number, Oracle will size internal resources and create threads for different purposes (for example, parallel queries and number of readers). These calls will return the expected value (that is, the number of CPUs in the resource pool) if the zone is bound to a resource pool with a pset associated with it. Table 1 shows the interfaces that have been modified in the Solaris OS and that will return the expected value in this scenario.

INTERFACE	TYPE
<code>p_online(2)</code>	System Call
<code>processor_bind(2)</code>	System Call
<code>processor_info(2)</code>	System Call
<code>pset_list(2)</code>	System Call
<code>pset_info(2)</code>	System Call
<code>pset_getattr(2)</code>	System Call
<code>pset_getloadavg(3c)</code>	System Call
<code>getloadavg(3c)</code>	System Call
<code>sysconf(3c)</code>	System Call
<code>_SC_NPROCESSORS_CONF</code>	<code>sysconf(3c)</code> arg
<code>_SC_NPROCESSORS_ONLN</code>	<code>sysconf(3c)</code> arg

INTERFACE	TYPE
<code>pbind(1M)</code>	Command
<code>psrset(1M)</code>	Command
<code>psrinfo(1M)</code>	Command
<code>mpstat(1M)</code>	Command
<code>vmstat(1M)</code>	Command
<code>iostat(1M)</code>	Command
<code>sar(1M)</code>	Command

Table 1: List of CPU-Related Solaris 10 Interfaces That Are Container-Aware

In addition to these interfaces, certain kernel statistics (*kstats*) are used commonly by tools such as `psrinfo(1M)` and `mpstat(1M)`, to retrieve information about the system. All consumers of these *kstats* will only see information for a *pset* in the pool bound to the zone.

7 Oracle Features Not Available in Containers

This section covers some of the Oracle database features that are not available in non-global zones in containers.

7.1 Oracle RAC

An Oracle RAC installation is composed of several nodes, shared storage, and a private interconnect. A container cannot be used as an Oracle RAC node, mainly for the following two reasons.

Cluster manager: One of the software components that must be present in a RAC installation is a cluster manager. The cluster manager is responsible for isolating failed systems from the shared storage, avoiding corruption of data, and enabling communication between nodes in the cluster via a private interconnect. In order to use containers as RAC nodes the cluster manager would need to become capable of managing containers. As of this writing no cluster solution is capable of using non-global containers as cluster members or nodes.

Oracle RAC VIP: Another limitation to running Oracle RAC in a container is the Virtual IP. Oracle RAC uses a virtual interface in each node to distribute client connections to all the active nodes in the cluster. When a node (node A) leaves the cluster, one of the other active members of the cluster (node B) will take over its virtual IP address by bringing up an extra virtual interface with the IP address used by node A. In this way node B will service all new connections sent to node A until node A re-joins the cluster. For security reasons, a container does not have the privileges required for managing virtual interfaces. Therefore, the VIP mechanism used by Oracle RAC conflicts with the container security limitations.

7.2 Solaris Dynamic Intimate Shared Memory (DISM)

This section provides a brief overview of DISM and explains why it is not supported in containers.

DISM provides shared memory that is dynamically resizable. A process that makes use of a DISM segment can lock and unlock parts of a memory segment while the process is running. By doing so, the application can dynamically adjust to the addition of physical memory to the system or prepare for the removal of it.

By default, Oracle uses intimate shared memory (ISM) instead of standard System V shared memory on the Solaris OS. When a shared memory segment is made into an ISM segment, it is mapped using large pages and the memory for the segment is locked (that is, it cannot be paged out). This greatly reduces the overhead due to process context switches, which improves Oracle's performance linearity under load. For more details about Oracle and DISM, see reference [5].

ISM has been a feature of the Solaris OS since Solaris 2.2, and it is enabled by default in

Solaris 2.4 and above. ISM is used by default by Oracle from 7.2.2 onwards. Prior to Oracle 7.2.2, ISM could be turned on in Oracle by adding the directive `use_ism=true` in the `init.ora` file. Oracle uses ISM by default and recommends its use. ISM is supported in containers.

ISM certainly has benefits over standard System V shared memory. However, its disadvantage is that ISM segments cannot be resized. To change the size of an ISM database buffer cache, the database must be shut down and restarted. DISM overcomes this limitation as it provides shared memory that is dynamically resizable. DISM has been supported in Oracle databases since Oracle 9i. Oracle uses DISM instead of ISM if the maximum SGA size set by the `sga_max_size` parameter in `init.ora` is larger than the sum of its components (that is, `db_cache_size`, `shared_pool_size`, and other smaller SGA components).

In ISM, the kernel locks and unlocks memory pages. However, in DISM the locking and unlocking of memory pages is done by the Oracle process `ora_dism_<${ORACLE_SID}>`. In order to lock memory, a process needs the `proc_lock_memory` privilege. This privilege is not available in a Solaris non-global zone. Although Oracle comes up when DISM is invoked in a non-global zone, it is not able to lock SGA memory in Solaris non-global zones. This is likely to cause performance to degrade, therefore **Oracle should not be configured to use DISM in non-global zones.**

8 Appendix

8.1 Appendix 1: Script to Create a Container

The scripts documented in this Appendix can be used to create a container appropriate for installing and running non-RAC instances of an Oracle database. These scripts do not represent the only way in which the container can be created. They are provided as sample code and should be modified to fit specific requirements and constraints.

These scripts will first create a resource pool and a processor set (`pset`) resource with a minimum and maximum number of CPUs in it (both values specified by the user). These new resources will be configured in the default configuration file `/etc/pooladm.conf` (see `pooladm(1M)` for details). Once created, the `pset` will be associated with the resource pool. Next, a sparse root zone will be created with the root directory, IP address, and physical interface provided by the user. A special mount point for `/usr/local` will be created in `/opt/<zone_name>/local` to facilitate the Oracle installation, since `/usr/local` is the default directory for the installation of some of the Oracle utilities. Once created, the zone will be bound to the resource pool. The combination of the zone bound to the resource pools will define the container in which Oracle can be installed and executed. This container (and all processes running in it) will only have access to the CPUs associated to the resource pool. To use these scripts, save all the files in the same directory and follow these steps:

1) Edit the file `setenv.sh` with appropriate values for the following variables:

- `ZONE_NAME`: hostname for the zone
- `ZONE_DIR`: directory for the root directory of the zone
- `NET_IP`: IP address for the zone
- `NET_PHYSICAL`: physical interface in which the virtual interface for the zone will be created
- `NUM_CPUS_MAX`: maximum number of CPUs in the `pset` resource
- `NUM_CPUS_MIN`: minimum number of CPUs in the `pset` resource

2) Issue the following command from the global container:

```
./create_container.sh
```

3) Configure this container by executing the following command from global container:

```
zlogin -C <zone_name>
```

The files composing this scripts are presented and explained next.

8.1.1 README.txt

This file describes how a container is created when these scripts are used.

It also gives some tips about how to execute some common operations such as giving the zone access to raw devices or removing resource pools.

The scripts in this directory can be used to create a container suitable for installing and running non-RAC instances of Oracle database. These scripts do not represent the only way in which you can create an appropriate container for Oracle; depending on your requirements and constraints you can modify these scripts to fit your needs.

1) creating a container for Oracle

These scripts will first create a resource pool and a processor set (pset) resource with a minimum and maximum number of CPUs in it (both values specified by the user). These new resources will be configured in the default configuration file /etc/pooladm.conf (see pooladm(1M) for details). Once created, the pset will be associated with the resource pool. Next, a sparse root zone will be created with the root directory, IP and interface provided by the user. A special mount point for /usr/local will be created in /opt/<zone_name>/local to facilitate the oracle installation, since /usr/local is the default directory for the installation of some of the oracle utilities. Once created, the zone will be bound to the resource pool. The combination of the zone bound to the resource pools will define the container in which Oracle can be installed and used. This non-global container (and all processes running in it) will only have access to the CPUs associated to the resource pool. To use these scripts follow these steps:

- a) edit the file setenv.sh with appropriate values for:
 - ZONE_NAME: hostname for the zone
 - ZONE_DIR: directory for the root directory of the zone
 - NET_IP: IP for the zone
 - NET_PHYSICAL: physical interface in which the virtual interface for the zone will be created
 - NUM_CPUS_MAX: maximum number of CPUs in the pset resource
 - NUM_CPUS_MIN: minimum number of CPUs in the pset resource
- b) from the global container run ./create_container.sh
- c) Once the container has been created run "zlogin -C <zone_name>" from the global container to finish configuring the zone.

2) giving your container access to raw devices

If you need to give your container access to a raw device follow this example once the container has been created (these commands must be issued from the global container):

```
zonecfg -z my-zone
zonecfg:my-zone> add device
zonecfg:my-zone:device> set match=/dev/rdisk/c3t40d0s0
zonecfg:my-zone:device> end
zonecfg:my-zone> exit
zonecfg -z my-zone halt
zonecfg -z my-zone boot
```

3) giving your container access to a file system

If you need to give your container access to a file system created in the global container follow this example once the non-global

container has been created:

```
global# newfs /dev/rdisk/clt0d0s0
global# zonecfg -z my-zone
zoncfg:my-zone> add fs
zoncfg:my-zone> set dir=/usr/mystuff
zoncfg:my-zone> set special=/dev/dsk/clt0d0s0
zoncfg:my-zone> set raw=/dev/rdisk/clt0d0s0
zoncfg:my-zone> set type=ufs
zoncfg:my-zone> end
zonecfg -z my-zone halt
zonecfg -z my-zone boot
```

4) to remove pool resources previously created by operating directly on the kernel (see poolcfg(1M) for details) use these commands:

```
poolcfg -d -c 'destroy pool my_pool'
poolcfg -d -c 'destroy pset my_pset'
```

5) to uninstall and delete a previously created zone use these commands:

```
zoneadm -z $ZONE_NAME halt
zoneadm -z $ZONE_NAME uninstall -F
zonecfg -z $ZONE_NAME delete -F
```

8.1.2 setenv.sh

This file is where the user defines the parameters to create the container.

```
#!/usr/bin/sh

#host name for the zone
ZONE_NAME=myzone
#directory where to place root dir for the zone
ZONE_DIR=/export/home
#IP for the zone (make sure netmask can be resolved for this IP according to
# the databases defined in nsswitch.conf)
NET_IP=129.146.182.199
#interface used by the zone
NET_PHYSICAL=bge0
#min and max CPUs for the pool bound to the zone
NUM_CPUS_MIN=1
NUM_CPUS_MAX=1

# do not make changes beyond this point
POOL_NAME=pool_${ZONE_NAME}
PSET_NAME=ps_${ZONE_NAME}
export ZONE_NAME ZONE_DIR NET_IP NET_PHYSICAL
export POOL_NAME PSET_NAME NUM_CPUS_MIN NUM_CPUS_MAX
```

8.1.3 zone_cmd_template.txt

This file contains a template set of commands to create the zone. After replacing some strings by user-defined values, it will be used to create the zone.

```
create
set zonepath=ZONE_DIR/ZONE_NAME
set autoboot=true
set pool=POOL_NAME
add net
set address=NET_IP
set physical=NET_PHYSICAL
end
add fs
set dir=/usr/local
set special=/opt/ZONE_NAME/local
set type=lofs
end
verify
commit
```

8.1.4 pool_cmd_template.txt

This file contains a template set of commands to create the resource pool and the pset resource, and to associate them. After replacing some strings with user-defined values, it will be used to create the zone.

```
create pset PSET_NAME ( uint pset.min = NUM_CPUS_MIN ; uint pset.max
=NUM_CPUS_MAX)
create pool POOL_NAME
associate pool POOL_NAME ( pset PSET_NAME )
```

8.1.5 create_zone_cmd.sh

This script will use the `sed` utility to create a command file that creates the zone. It replaces the user-given parameters in the zone command template file. It is called by `create_container.sh`.

```
#!/bin/sh
# Copyright (c) 2005 Sun Microsystems, Inc. All Rights Reserved.
#
# SAMPLE CODE
# SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT
# THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS
# OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
# IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
# FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
# SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED
# BY LICENSEE AS A RESULT OF USING, MODIFYING OR
# DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

echo $ZONE_DIR > /tmp/ZP.$$
REG_ZONE_DIR=`sed 's\\/\\/\\\\\\\\\\\\/g' /tmp/ZP.$$`
rm -rf /tmp/ZP.$$ > /dev/null

sed -e "
/ZONE_DIR/ {
    s/ZONE_DIR/$REG_ZONE_DIR/
}
/ZONE_NAME/ {
    s/ZONE_NAME/$ZONE_NAME/
}
/NET_IP/ {
    s/NET_IP/$NET_IP/
}
/NET_PHYSICAL/ {
    s/NET_PHYSICAL/$NET_PHYSICAL/
}
/POOL_NAME/ {
    s/POOL_NAME/$POOL_NAME/
}
./ {
    p
    d
}"
```

8.1.6 create_pool_cmd.sh

This script will use the `sed` utility to create a command file that creates the resources. It replaces the user-given parameters in the pool command template file. It is called by

create_container.sh.

```
#!/bin/sh
# Copyright (c) 2005 Sun Microsystems, Inc. All Rights Reserved.
#
# SAMPLE CODE
# SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT
# THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS
# OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
# IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
# FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
# SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED
# BY LICENSEE AS A RESULT OF USING, MODIFYING OR
# DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

sed -e "
/NUM_CPUS_MIN/ {
    s/NUM_CPUS_MIN/$NUM_CPUS_MIN/g
}
/NUM_CPUS_MAX/ {
    s/NUM_CPUS_MAX/$NUM_CPUS_MAX/g
}
/POOL_NAME/ {
    s/POOL_NAME/$POOL_NAME/
}
/PSET_NAME/ {
    s/PSET_NAME/$PSET_NAME/
}
./ {
    p
    d
}"
```

8.1.7 create_container.sh

This is the main script. It will use the parameters given in `setenv.sh` to create the container.

```
#!/usr/bin/ksh
# Copyright (c) 2005 Sun Microsystems, Inc. All Rights Reserved.
#
# SAMPLE CODE
# SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT
# THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS
# OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
# IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
# FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
# SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED
# BY LICENSEE AS A RESULT OF USING, MODIFYING OR
# DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

# script to create a container to run oracle RDBMS.
# to use this script follow the instructions in the README.txt file
```

```

# located in this directory.

. ./setenv.sh

# 1)..... validate setenv.sh values

#zone path exists?
if [ ! -d $ZONE_DIR/$ZONE_NAME ]
then
  mkdir -p $ZONE_DIR/$ZONE_NAME
  if [ $? = 1 ]
  then
    echo ERROR: could not create root directory
    exit 1
  fi
fi
chmod 700 $ZONE_DIR/$ZONE_NAME

#zone already exists?
zonecfg -z $ZONE_NAME info > /tmp/z.$$ 2>&1
cat /tmp/z.$$ | grep "No such zone" > /dev/null 2>&1
if [ $? -eq 1 ]
then
  echo "ERROR: zone $ZONE_NAME already exists. IF you want to remove it do:"
  echo "use zoneadm -z $ZONE_NAME halt"
  echo "use zoneadm -z $ZONE_NAME uninstall -F"
  echo "use zonecfg -z $ZONE_NAME delete -F"
  exit 1
fi
rm -rf /tmp/z.$$ > /dev/null 2>&1

#pset already created?
pooladm -e
pooladm | grep pset | grep $PSET_NAME > /dev/null 2>&1
if [ $? -eq 0 ]
then
  echo "ERROR: pset $PSET_NAME already exists. Please choose a different \
pset name "
  exit 1
fi

#/usr/local directory exists?
if [ ! -d /usr/local ]
then
  mkdir /usr/local
fi

#special mnt point for /usr/local exists?
if [ ! -d /opt/$ZONE_NAME/local ]
then
  mkdir -p /opt/$ZONE_NAME/local
fi

# 2)..... pool creation
./create_pool_cmd.sh < pool_cmd_template.txt > /tmp/pool_commands.txt
pooladm -e # enable facility
#check for default config file exists, if not there
#create one with active configuration
if [ ! -f /etc/pooladm.conf ]

```

```

then
    pooladm -s /etc/pooladm.conf
fi
poolcfg -f /tmp/pool_commands.txt          # configure
pooladm -n > /tmp/pool.out 2>&1          # validate
if [ ! $? -eq 0 ]
then
    echo ERROR: invalid pool configuration.  please see /tmp/pool.out
    exit 1
fi
#instantiate config at /etc/pooladm.conf
pooladm -c

# 3)..... zone creation
./create_zone_cmd.sh < zone_cmd_template.txt > /tmp/zone_commands.txt
zonecfg -z $ZONE_NAME -f /tmp/zone_commands.txt
echo $ZONE_NAME was configured with this information:
echo -----
zonecfg -z $ZONE_NAME info
echo -----
zoneadm -z $ZONE_NAME install
zoneadm -z $ZONE_NAME boot

echo "to finish configuring your container please run: zlogin -C $ZONE_NAME"

```

8.2 Appendix 2: Setting System V IPC Kernel Parameters

Prior to the Solaris 10 OS, the System V IPC resources, consisting primarily of shared memory, message queues, and semaphores, were set in the `/etc/system` file. This implementation had the following shortcomings:

- Relying on `/etc/system` as an administrative mechanism meant reconfiguration required a reboot.
- A simple typo in setting the parameter in `/etc/system` could lead to hard-to-track configuration errors.
- The algorithms used by the traditional implementation assumed statically-sized data structures.
- There was no way to allocate additional resources to one user without allowing all users those resources. Since the amount of resources was always fixed, one user could have trivially prevented another from performing its desired allocations.
- There was no good way to observe the values of the parameters.
- The default values of certain tunables were too small.

In the Solaris 10 OS, all these limitations were addressed. The System V IPC implementation in the Solaris 10 OS no longer requires changes in the `/etc/system` file.

Instead, it uses the resource control facility, which brings the following benefits:

- It is now possible to install and boot an Oracle instance without needing to make changes to `/etc/system` file (or to resource controls in most cases).
- It is now possible to limit use of the System V IPC facilities on a per-process or per-project basis (depending on the resource being limited), without rebooting the system.

- None of these limits affect allocation directly. They can be made as large as possible without any immediate effect on the system. (Note that doing so would allow a user to allocate resources without bound, which **would** have an effect on the system.)
- Implementation internals are no longer exposed to the administrator, thus simplifying the configuration tasks.
- The resource controls are fewer and are more verbosely and intuitively named than the previous tunables.
- Limit settings can be observed using the common resource control interfaces, such as `prctl(1)` and `getrctl(2)`.
- Shared memory is limited based on the total amount allocated per project, not per segment. This means that an administrator can give a user the ability to allocate a lot of segments **and** large segments, without having to give the user the ability to create a lot of large segments.
- Because resource controls are the administrative mechanism, this configuration can be persistent using `project(4)` and be made via the network.

In the Solaris 10 OS, the following changes were made:

- Message headers are now allocated dynamically. Previously all message headers were allocated at module load time.
- Semaphore arrays are allocated dynamically. Previously semaphore arrays were allocated from a `seminfo_semmns` sized `vmem` arena, which meant that allocations could fail due to fragmentation.
- Semaphore undo structures are dynamically allocated per-process and per-semaphore array. They are unlimited in number and are always as large as the semaphore array they correspond to. Previously there were a limited number of per-process undo structures, allocated at module load time. Furthermore, the undo structures each had the same, fixed size. It was possible for a process to not be able to allocate an undo structure, or for the process's undo structure to be full.
- Semaphore undo structures maintain their undo values as signed integers, so no semaphore value is too large to be undone.
- All facilities were used to allocate objects from a fixed size namespace, and were allocated at module load time. All facility namespaces are now resizable, and will grow as demand increases.

As a consequence of these changes, the following related parameters have been removed. If these parameters are included in the `/etc/system` file on a Solaris system, the parameters are ignored (see Table 2).

<i>Parameter Name</i>	<i>Brief Description</i>
<code>semsys:seminfo_semmns</code>	Maximum number of System V semaphores on the system
<code>semsys:seminfo_semmnu</code>	Total number of undo structures supported by the System V semaphore system
<code>semsys:seminfo_semmap</code>	Number of entries in semaphore map
<code>semsys:seminfo_semvmx</code>	Maximum value a semaphore can be set to
<code>semsys:seminfo_semaem</code>	Maximum value that a semaphore's value in an undo structure can be set to
<code>semsys:seminfo_semusz</code>	The size of the undo structure
<code>shmsys:shminfo_shmseg</code>	Number of segments, per process
<code>shmsys:shminfo_shmmin</code>	Minimum shared memory segment size
<code>msgsys:msginfo_msgmap</code>	The number of entries in a message map
<code>msgsys:msginfo_msgssz</code>	Size of the message segment
<code>msgsys:msginfo_msgseg</code>	Maximum number of message segments
<code>msgsys:msginfo_msgmax</code>	Maximum size of System V message

Table 2: System Parameters No Longer Needed
in the Solaris 10 OS

As described above, many `/etc/system` parameters are removed simply because they are no longer required. The remaining parameters have more reasonable defaults, enabling more applications to work out-of-the-box without requiring these parameters to be set.

Table 3 describes the default value of the remaining `/etc/system` parameters.

<i>Resource Control</i>	<i>Obsolete Tunable</i>	<i>Old Default Value</i>	<i>New Default Value</i>
<code>process.max-msg-qbytes</code>	<code>msginfo_msgmnb</code>	4096	65536
<code>process.max-msg-messages</code>	<code>msginfo_msgtql</code>	40	8192
<code>process.max-sem-ops</code>	<code>seminfo_semopm</code>	10	512
<code>process.max-sem-nsems</code>	<code>seminfo_semmsl</code>	25	512
<code>project.max-shm-memory</code>	<code>shminfo_shmmax</code>	0x800000	1/4 of physical memory
<code>project.max-shm-ids</code>	<code>shminfo_shmmni</code>	100	128
<code>project.max-msg-ids</code>	<code>msginfo_msgmni</code>	50	128
<code>project.max-sem-ids</code>	<code>seminfo_semmni</code>	10	128

Table 3: Default Values for System Parameters in the Solaris 10 OS

Setting System V IPC Parameters for Oracle Installation

Table 4 identifies the values recommended for `/etc/system` parameters by the Oracle Installation Guide and the corresponding Solaris resource controls.

<i>Parameter</i>	<i>Oracle Recommendation</i>	<i>Required in Solaris 10 OS</i>	<i>Resource Control</i>	<i>Default Value</i>
SEMMNI (<code>semsys:seminfo_s emmni</code>)	100	Yes	<code>project.max- sem-ids</code>	128

SEMMNS (semsys:seminfo_s emmns)	1024	No	N/A	N/A
SEMMSL (semsys:seminfo_s emmsl)	256	Yes	project.max- sem-nsems	512
SHMMAX(shmsys:sh minfo_shmmax)	4294967295	Yes	project.max- shm-memory	1/4 of physical memory
SHMMIN (shmsys:shminfo_s hmmmin)	1	No	N/A	N/A
SHMMNI (shmsys:shminfo_s hmmni)	100	Yes	project.max- shm-ids	128
SHMSEG (shmsys:shminfo_s hmseg)	10	No	N/A	N/A

Table 4: Recommended Values for System Parameters When Running Oracle 10g

Since the default values are higher than Oracle recommended values, the only resource control that might need to be set is `project.max-shm-memory`. The following section details the process of setting a particular value using resource control.

Using Resource Control Commands to Set System V IPC Parameters

The `prctl` command can be used to view and change value of resource control. The `prctl` command is invoked with the `-n` option to display the value of a certain resource control. The following command displays the value of `max-file-descriptor` resource control for the specified process:

```
prctl -n process.max-file-descriptor <pid>
```

The following command updates the value of `project.cpu-shares` in the `project group.staff`:

```
prctl -n project.cpu-shares -v 10 -r -l project group.staff
```

9 References

[1] *Consolidating Applications with Solaris 10 Containers*, Sun Microsystems, 2004
http://www.sun.com/datacenter/consolidation/solaris10_whitepaper.pdf

[2] *Solaris 10 System Administrator Collection -- System Administration Guide: Solaris Containers-Resource Management and Solaris Zones*, Sun Microsystems, 2005
<http://docs.sun.com/app/docs/doc/817-1592>

[3] *Solaris Containers--What They Are and How to Use Them*, Menno Lageman, Sun BluePrints OnLine, 2005
<http://www.sun.com/blueprints/0505/819-2679.html>

[4] Solaris Zones section on BigAdmin System Administration Portal, Sun Microsystems
<http://www.sun.com/bigadmin/content/zones>

[5] *Dynamic Reconfiguration and Oracle 9i Dynamically Resizable SGA*, Erik Vanden Meersch and Kristien Hens, Sun BluePrints OnLine, 2004
<http://www.sun.com/blueprints/0104/817-5209.pdf>

[6] Oracle Pricing with Solaris 10 Containers

<http://www.sun.com/third-party/global/oracle/consolidation/solaris10.html>

[7] Oracle's Partitioning document

<http://www.oracle.com/corporate/pricing/partitioning.pdf>

[8] *Bringing Your Application Into the Zone*, Paul Lovvik and Joseph Balenzano, Sun Developer Network article, 2005

http://developers.sun.com/solaris/articles/application_in_zone.html

10 About the Authors

Ritu Kamboj is a staff engineer with Sun Microsystems' Market Development Engineering group. She works in the Oracle group helping Oracle to adopt Sun's technology and improve performance on Sun's hardware.

Fernando Castano joined Sun in October 2000. Since then he has been working as a member of technical staff in the Market Development Engineer organization. Currently he works in the Oracle group helping Oracle adopt Sun's technology and improve performance in Sun's hardware.