OCFS2

A CLUSTER FILE SYSTEM FOR LINUX

USER'S GUIDE FOR RELEASE 1.6

Sunil Mushran

September 2010

OCFS2: A Cluster File System for Linux – User's Guide for Release 1.6

Copyright © 2008, 2010 Oracle. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts.

The GNU Free Documentation License is available at the website, http://www.gnu.org/licenses/fdl.txt.

TABLE OF CONTENTS

PREFACE

The aim of this guide is to be a starting point for users looking to use the OCFS2 file system. New users should be able to jump to the Getting Started chapter to learn how to configure, format and mount the volume. Users looking to upgrade from an earlier release should refer to the chapter on New Features and review the sections on compatibilities and defaults. Oracle database users, in addition to the above, should also review the chapter titled Oracle RDBMS.

All users should note that this document is best consumed along with the man pages for the various OCFS2 tools. While this document explains the usage of the tools, it does not list all the available options. Users should supplement this guide with the man pages.

This guide has been written specifically for the users of OCFS2 Release 1.6, which is currently only available for the Oracle Linux distribution. However, users of other distributions can also make use of this guide, as OCFS2 Release 1.6 is just a snapshot of the file system in the mainline Linux kernel. The information included is current as of Linux kernel version 2.6.36.

I. INTRODUCTION

OCFS2 is a **file system**. It allows users to store and retrieve data. The data is stored in files that are organized in a hierarchical directory tree. It is a **POSIX compliant** file system that supports the standard interfaces and the behavioral semantics as spelled out by that specification.

It is also a **shared disk cluster** file system, one that allows multiple nodes to access the same disk at the same time. This is where the fun begins as allowing a file system to be accessible on multiple nodes opens a can of worms. What if the nodes are of different architectures? What if a node dies while writing to the file system? What data consistency can one expect if processes on two nodes are reading and writing concurrently? What if one node removes a file while it is still being used on another node?

Unlike most shared file systems where the answer is fuzzy, the answer in OCFS2 is very well defined. It behaves on all nodes exactly like a local file system. If a file is removed, the directory entry is removed but the inode is kept as long as it is in use across the cluster. When the last user closes the descriptor, the inode is marked for deletion.

The data consistency model follows the same principle. It works as if the two processes that are running on two different nodes are running on the same node. A read on a node gets the last write irrespective of the IO mode used. The modes can be buffered, direct, asynchronous, splice or memory mapped IOs. It is fully **cache coherent**.

Take for example the newly added *REFLINK* feature that allows a user to create multiple write-able snapshots of a file. This feature, like all others, is fully cluster-aware. A file being written to on multiple nodes can be safely reflinked on another. The snapshot created is a point-in-time image of the file that includes both the file data and all its attributes (including extended attributes).

It is a **journaling** file system. When a node dies, a surviving node transparently replays the journal of the dead node. This ensures that the file system metadata is always consistent. It also defaults to **ordered data** journaling to ensure the file data is flushed to disk before the journal commit, to remove the small possibility of stale data appearing in files after a crash.

It is **architecture** and **endian neutral**. It allows concurrent mounts on nodes with different processors like x86, x86_64, IA64 and PPC64. It handles little and big endian and 32-bit and 64-bit architectures.

It is **feature rich**. It supports *indexed directories*, *metadata checksums*, *extended attributes*, *POSIX ACLs*, *quotas*, *REFLINKs*, *sparse files*, *unwritten extents* and *inline-data*.

It is fully **integrated** with the mainline Linux kernel. The file system was merged into the mainline kernel in early 2006. It is available with almost all Linux distributions. The file

system is on-disk **compatible** across all of them.

It is **quickly installed**. The file system is part of Oracle's Unbreakable Enterprise Kernel. All the user needs to do is install that kernel and the *ocfs2-tools* rpm package.

It is **easily configured**. The cluster stack configuration involves editing two files, one that holds information about the nodes in the cluster, and the other, cluster timeouts.

It is very **efficient**. The file system consumes very little resources. It is widely used to store virtual machine images in the management domain of Oracle VM that is configured to use as little as 512MB of RAM.

There you have it. OCFS2 is an efficient, easily configured, quickly installed, fully integrated and compatible, feature-rich, architecture and endian neutral, cache coherent, ordered data journaling, POSIX-compliant, shared disk cluster file system.

II. OVERVIEW

OCFS2 is a general-purpose shared-disk cluster file system for Linux capable of providing both high performance and high availability. As it provides local file system semantics, it can be used with almost all applications. Cluster-aware applications can make use of cache-coherent parallel I/Os from multiple nodes to scale out applications easily. Other applications can make use of the file system facilities to fail-over running application in the event of a node failure.

The file system is currently being used in virtualization (Oracle VM) in both the management domain, to host virtual machine images, and in the guest domain, to allow Linux guests to share a file system. It is also being used in database clusters (Oracle RAC), middleware clusters (Oracle E-Business Suite), appliances (SAP's Business Intelligence Accelerator), etc.

Some of the notable features of the file system are:

- **Variable Block and Cluster sizes**
  Supports block sizes ranging from 512 bytes to 4 KB and cluster sizes ranging from 4 KB to 1 MB (increments in power of 2).

- **Extent-based Allocations**
  Tracks the allocated space in ranges of clusters making it especially efficient for storing very large files.

- **Optimized Allocations**
  Supports sparse files, inline-data, unwritten extents, hole punching, reflinks and allocation reservation for higher performance and efficient storage.

- **Indexed Directories**
  Allows efficient access to millions of objects in a directory.

- **Metadata Checksums**
  Detects silent corruption in inodes and directories.

- **Extended Attributes**
  Supports attaching an unlimited number of name:value pairs to the file system objects like regular files, directories, symbolic links, etc.

- **Advanced Security**
  Supports ***POSIX ACLs*** and ***SELinux*** in addition to the traditional file access permission model.

- **Quotas**
  Supports user and group quotas.

- **Journaling**
  Supports both ordered and writeback data journaling modes to provide file system consistency in the event of power failure or system crash.

- **Endian and Architecture neutral**
  Supports a cluster of nodes with mixed architectures. Allows concurrent mounts on nodes running 32-bit and 64-bit, little-endian (x86, x86_64, ia64) and big-endian (ppc64) architectures.

- **In-built Cluster-stack with DLM**
  Includes an easy to configure, in-kernel cluster-stack with a distributed lock manager.

- **Buffered, Direct, Asynchronous, Splice and Memory Mapped I/Os**
  Supports all modes of I/Os for maximum flexibility and performance.

- **Comprehensive Tools Support**
  Provides a familiar EXT3-style tool-set that uses similar parameters for ease-of-use.

History

The OCFS2 file system development began in 2003. The goals for the project were to provide raw-like I/O throughput for the database, be POSIX compliant, and provide near local file system performance for meta-data operations. An additional goal was to submit the file system for merging into the mainline Linux Kernel. At that time the Linux kernel did not have a cluster file system.

The file system was accepted into the mainline Linux kernel in January 2006. The 2.6.16 kernel, released that March, included the file system.

OCFS2 Release 1.2 was released in April 2006. It targeted the Enterprise Linux distributions, *SLES9* from Novell and *RHEL4* from Red Hat. The file system was made available for the x86, x86_64, ia64, ppc64, and s390x architectures.

OCFS2 Release 1.4 was released in July 2008. It was available on all three Enterprise Linux distributions, namely, Oracle Linux, Red Hat's EL and Novell's SLES. The new features in that release included sparse files, unwritten extents, inline-data, and shared writeable mmap.

OCFS2 Release 1.6 is the latest release of the file system and is available on Oracle Linux 5 with the Unbreakable Enterprise Kernel. The new features in this release include REFLINKs, indexed directories, metadata checksums, extended attributes, quotas, POSIX ACLs, and allocation reservations.

Development

The OCFS2 development began as project in the Linux Kernel development group in Oracle Corporation. However, since its inclusion in the mainline Linux kernel, it has attracted patch submissions from over 100 developers including developers from other companies, notably, Novell.

In order to satisfy the needs of our users, who want a stable file system on a Linux distribution of their choice, and the developers, who want a consistent environment for developing new features, the development group follows few basic ground rules:

1. All new features are first included in the mainline Linux kernel tree.
2. All bug fixes are applied to all active kernel trees.

Active kernel trees include the currently supported Enterprise kernels, the current mainline tree and the stable kernel trees. The stable trees are tracked by most Linux distributions.

The source of the file system is made available with the Linux kernel and can be downloaded from http://kernel.org/. The source for file system for the Oracle Linux kernel is available at http://oss.oracle.com/el5/SRPMS-updates/.

The source of the OCFS2 file system is available under the GNU General Public License (GPL) version 2.

Support

The support for the file system is included as part of Oracle Linux support contract.

The OCFS2 development community also provides email support for all users via the *ocfs2-users@oss.oracle.com* mailing list.

III. NEW FEATURES

OCFS2 Release 1.6 comes with a slew of new features that includes new functionalities for the user and the system administrator, improved performance, and fixes to some problems. Most of the new features require explicit activation, details for which are provided in the chapter titled *File System Features*.

The new features added since the OCFS2 Release 1.4 are as follows:

1. Extended Attributes

This is a new functionality that allows a user to associate *name:value* pairs to file system objects like regular files, directories, symbolic links, etc. OCFS2 allows associating an unlimited number of attributes per inode. The attribute names can be up to 255 bytes in length, terminated by the first NUL character. While it is not required, printable names (ASCII) are recommended. The attribute values can be up to 64 KB of arbitrary binary data.

The example below shows attaching a user attribute, *location*, with a value, *New York*, to a jpg file.

```
$ setfattr –n user.location –v “New York” DSC00134.jpg
$ getfattr –d DSC00134.jpg
# file: DSC00134.jpg
  user.location=”New York”
```

This feature entails an on-disk change and can be activated by enabling the **xattr** file system feature.

More information on extended attributes can be found in the man page for *attr(5)*.

2. POSIX Access Control Lists

This feature is aimed at the both the user and the system administrator as it allows them to assign fine-grained discretionary access rights for files and directories. This security scheme is a lot more flexible than the traditional file access permissions that imposes a strict user-group-other model. With POSIX ACLs, one can assign multiple named users specific permissions to an object.

The example below shows file owner *sunil* assigning the read privilege to user *joel*, the write privilege to user *tao* and the execute privilege to user *tristan*.

```
$ setfacl -m u:joel:r setup.sh
$ setfacl -m u:tao:w setup.sh
$ setfacl -m u:tristan:x setup.sh
```

```
$ getfacl setup.sh
# file: setup.sh
# owner: sunil
# group: sunil
user::rw-
user:joel:r--
user:tao:-w-
user:tristan:--x
group::r--
mask::rwx
other::r--
```

This feature requires the file system feature **xattr** to be enabled.

More information on POSIX ACLs can be found in the man page for *acl(5)*.

3. Indexed Directories

This feature allows a user to perform quick lookups of a directory entry in a very large directory. It also results in faster creates and unlinks and thus provides better overall performance.

This feature entails an on-disk change and can be activated by enabling the **indexed-dirs** file system feature.

4. Metadata Checksums

This feature makes the file system compute and validate the checksums of meta-data objects, like inodes and directories, to ensure meta-data integrity. It also stores an error correction code capable to fixing single bit errors.

This feature entails an on-disk change and can be activated by enabling the **metaecc** file system feature.

5. REFLINK

The feature allows a user to create multiple write-able snapshots of regular files. It is called REFLINK because it looks and feels more like a (hard) link than a traditional snapshot. Like a link, it is a regular user operation, subject to the security attributes of the inode being reflinked and not to the super user privileges typically required to create a snapshot. Like a link, it operates within a file system. But unlike a link, it links the inodes at the data extent level allowing each reflinked inode to grow independently as and when written to. Up to *four billion* inodes can share a data extent.

The example shows a file of size 1 GB being reflinked in a fraction of a second.

```
$ ls -1
total 1024000
-rw-r--r-- 1 marcos marcos 1048576000 Sep 16 16:38 myfile

$ time reflink myfile myfile-ref

real    0m0.012s
user    0m0.000s
sys     0m0.004s

$ ls -l
total 2048000
-rw-r--r-- 1 marcos marcos 1048576000 Sep 16 16:38 myfile
-rw-r--r-- 1 marcos marcos 1048576000 Sep 17 10:48 myfile-ref
```

This feature entails an on-disk change and can be activated by enabling the **refcount** file system feature.

The *reflink(1)* utility is available with the reflink rpm package. More information on this feature is available in the chapter titled Notes.

6. User and Group Quotas

This feature allows a system administrator to setup usage quotas on a user and group basis by using the standard utilities like quota(1), setquota(8), quotacheck(8), and quotaon(8) to utilize this feature.

This feature entails an on-disk change and can be activated by enabling the **usrquota** and **grpquota** file system features.

7. Allocation Reservation

File contiguity plays an important role in file system performance. When a file is fragmented on disk, reading and writing to the file involves many seeks, leading to lower throughput. Contiguous files, on the other hand, minimize seeks, allowing the disks to perform IO at the maximum rate.

A simple scheme to reduce fragmentation is over-allocation. In this scheme, the file system grows the file in large chunks, say, 5% of the current file size. While this scheme helps in reducing fragmentation, it does so at the cost of space wastage. The over allocated space is typically released when the file is deleted. Some file systems over allocate but release the extra space when the file is no longer in use. This is definitely better than release-on-delete, but is also inefficient as the space is unavailable for some duration.

A better scheme is referred to as *reservation*. As the name suggests, the file system merely reserves a window in the bitmap for all extending files allowing each to grow as contiguously as possible. As this extra space is not actually allocated, it is available for use by other files if the need arises.

The allocation reservation feature is automatically enabled in this release. More information on this feature is available in the chapter titled Notes.

8. JBD2 Support

JBD2 is the new journal block device that will allow the file system to grow beyond 16TB.

This feature is automatically enabled in this release.

9. Discontiguous Block Group

Most file systems pre-allocate space for inodes during format. OCFS2 dynamically allocates this space when required.

However, this dynamic allocation has been problematic when the free space is very fragmented, because the file system required the inode and extent allocators to grow in contiguous fixed-size chunks.

The discontiguous block group feature takes care of this problem by allowing the allocators to grow in smaller, variable-sized chunks.

This feature entails an on-disk change and can be activated by enabling the **discontig-bg** file system feature.

File System Compatibility

OCFS2 Release 1.6 is fully compatible with OCFS2 Release 1.4. A node with the new release can join a cluster of nodes running the older file system.

OCFS2 Release 1.6 is on-disk compatible with OCFS2 Release 1.2. A node with the new release can mount a volume used by the older release.

Tools Compatibility

The latest version of ocfs2-tools supports all existing versions of the file system.

New File System Defaults

The OCFS2 1.6 does not automatically enable any new on-disk feature. Users looking to use a new feature will need to enable it with *tunefs.ocfs2(8)* or *mkfs.ocfs2(8)*.

IV. FILE SYSTEM FEATURES

File system features refer to all features added after the 1.0 release. They entail an on-disk format change. The OCFS2 utilities, mkfs.ocfs2(8) and tunefs.ocfs2(8) stamp the enabled features in the super block. When a volume is later mounted, the file system software compares the enabled features with the ones it knows about. If it does not understand a feature, it fails to mount the volume. The same applies to utilities like fsck.ocfs2(8). They only operate on volumes having features they fully understand.

OCFS2 uses this to provide on-disk compatibility across releases. When a newer file system mounts an older volume, it restricts the functionality to the set of features enabled on disk. Newer features are never auto-enabled. The user needs to specifically enable them.

Most features can also be disabled. Users wishing to revert to an older file system version may need to disable a feature. This topic is covered in more detail in a later section.

Feature Categories

The file system features are split into three categories: Compat, Incompat and RO Compat.

Compat, or compatible, is a feature that the file system does *not* need to fully understand to safely read/write to the volume. An example of this is the *backup-super* feature that added the capability to backup the super block in multiple locations in the file system. As the backup super blocks are typically not read nor written to by the file system, an older file system can safely mount a volume with this feature enabled.

Incompat, or incompatible, is a feature that the file system needs to fully understand to read/write to the volume. Most features fall under this category.

RO Compat, or read-only compatible, is a feature that the file system needs to fully understand to write to the volume. Older software can safely read a volume with this feature enabled. An example of this would be user and group quotas. As quotas are manipulated only when the file system is written to, older software can safely mount such volumes in read-only mode.

Compat Features

- **backup-super**
  This compat feature indicates that the volume has backups of the super block. *mkfs.ocfs2(8)*, by default, makes up to 6 backup copies of the super block at offsets 1GB, 4GB, 16GB, 64GB, 256GB and 1TB, depending on the size of the volume. These backups can be useful in disaster recovery. Refer to the *Administration* chapter for recovering using the backup super blocks.

- **strict-journal-super**
  This compat feature indicates that it is using version 2 of the JBD super block. Mind you, not JBD2, but version 2 of the JBD super block that is used in both JBD and JBD2. This feature is enabled by default.

Incompat Features

- **local**
  This incompat feature is enabled when the user wishes to mount the volume without a cluster stack. It is also referred to as a local mount.

- **sparse**
  A *sparse file* refers to a file with holes. If a user creates a file and writes one byte at offset 1MB, the sparse feature allows it to skip allocating (and zeroing) space for the first 1MB. That space is allocated only when written to. Reading a hole gets all zeroes. This feature allows the file system to be efficient in terms of both performance and space usage.

- **inline-data**
  This incompat feature allows the file system to store small files and directories in the inode block itself. Data is transparently moved out to an extent when it can no longer fit inside the block. The inline size depends on the block size. In a 4KB file system, the inline size is 3896 bytes. This feature is also referred to as data-in-inode.

- **extended-slotmap**
  The slot map is a system file used to map mounted nodes to system file resources. This incompat feature allows a large range of possible node numbers and is useful for user-space cluster stacks like cman and pacemaker. It should be noted that support for such stacks is not yet available in this release. We plan on enabling support for cman and pacemaker in an upcoming release of OCFS2.

- **metaecc**
  This feature enables the file system to compute and validate the checksums for all metadata blocks, like inodes and directories. It also computes an error correction code capable of fixing single bit errors. This incompat feature allows for better metadata integrity.

- **refcount**
  This feature enables the creation of reference counted (*refcount*) trees that are required to support reflinks. A refcount tree is used to store the usage count of shared extents.

- **xattr**
  Extended attributes are name:value pairs that can be associated with objects within a file system. In OCFS2, the names can be up to 255 bytes in length, terminated by the first NUL byte. While it is not required, printable names (ASCII) are recommended. The value can be up to 64KB of arbitrary binary data. Attributes can be associated with all types of inodes. Regular files, directories, symbolic links, device nodes, etc. This feature is also required to enable extended security facilities like POSIX ACLs and SELinux.

- **indexed-dirs**
  This feature allows the file system to create indexes in directories to significantly improve the directory entry lookup performance. Faster lookup results in faster create and unlink, which leads to better overall performance.

- **discontig-bg**
  This feature allows the file system to grow the inode and the extent allocators even when there is no large contiguous free chunk available. The allocators are grown in smaller, discontiguous chunks.

RO Compat Features

- **unwritten**
  This feature is needed to make the file system support the fallocate(2) system call that allows users to instantly pre-allocate large extents within a file. OCFS2 marks such extents with a special flag to skip the expensive data initialization step. Reads and writes to a pre-allocated region resemble those to a hole, except that a write will not fail due to lack of space allocation. This feature requires sparse file support to be enabled.

- **usrquota**
  The feature allows the file system to track the amount of space and number of inodes (files, directories, symbolic links) each user owns in order to limit the usage by each user. See the man page for quota(1) for more details.

- **grpquota**
  This feature is similar to usrquota with the difference that it tracks the usage for groups.

Enabling and Disabling Features

The format utility, mkfs.ocfs2(8), allows a user to enable and disable specific features using the –fs-features option. The features are provided as a comma separated list. The enabled features are listed as is. The disabled features are prefixed with **no**.

The example below shows the file system being formatted with sparse disabled and inline-data enabled.

```
$ mkfs.ocfs2 --fs-features=nosparse,inline-data /dev/sda1
```

After formatting, the users can toggle features using the tune utility, tunefs.ocfs2(8). This is an offline operation. The volume needs to be umounted across the cluster.

The example below shows the sparse feature being enabled and inline-data disabled.

```
$ tunefs.ocfs2 --fs-features=sparse,noinline-data /dev/sda1
```

Care should be taken before enabling and disabling features. If the user plans to use the volume with older file system software, do not enable features that are not supported by that version. One should not rely on the feature disable to always succeed.

An example of would be disabling the sparse feature; this requires filling every hole. The operation will only succeed if the file system has enough free space.

This is not to discourage users from disabling features. It is merely to make users aware of challenges that could be encountered in the process.

The tune utility can be used to query on-disk features. The following example shows the features on a recently created volume.

```
$ tunefs.ocfs2 -Q "Compat: %M\nIncompat: %H\nRO Compat: %O\n" /dev/sda1
Compat: backup-super strict-journal-super
Incompat: local sparse inline-data xattr indexed-dirs refcount
RO Compat: unwritten
```

Feature Compatibility

Say one tries to mount a volume with an incompatible feature. What happens then? How does one detect the problem? How does one know the name of that incompatible feature?

To begin with, one should look for error messages in dmesg. Mount failures that are due to an incompatible feature will always result in an error message like the following.

ERROR: couldn't mount because of unsupported optional features (200).

Here the file system is unable to mount the volume due to an unsupported optional feature. That means that that feature is an Incompat feature. By referring to the table below, one can then deduce that the user failed to mount a volume with the *xattr* feature enabled. (The value in the error message is in hexadecimal.)

Another example of an error message due to incompatibility is as follows.

ERROR: couldn't mount RDWR because of unsupported optional features (1).

Here the file system is unable to mount the volume in the RW mode. That means that that feature is a RO Compat feature. Another look at the table and it becomes apparent that the volume had the *unwritten* feature enabled.

In both cases, the user has the option of disabling the feature. In the second case, the user has the choice of mounting the volume in the RO mode.

Feature	Category	Hex Value	Version
backup-super	Compat	1	1.2
strict-journal-super	Compat	2	1.4
local	Incompat	8	1.2
sparse	Incompat	10	1.4
inline-data	Incompat	40	1.4
extended-slotmap	Incompat	100	1.6
xattr	Incompat	200	1.6
indexed-dirs	Incompat	400	1.6
metaecc	Incompat	800	1.6
refcount	Incompat	1000	1.6
discontig-bg	Incompat	2000	1.6
unwritten	RO Compat	1	1.4
usrquota	RO Compat	2	1.6
grpquota	RO Compat	4	1.6

V. GETTING STARTED

The OCFS2 software is split into two components, namely, kernel and user-space. The kernel component includes the core file system and the cluster stack. The user-space component provides the utilities to format, tune, mount and check the file system.

Software Packaging

The kernel component of OCFS2 1.6 is bundled with the *Unbreakable Enterprise Kernel* that is available for *Oracle Linux 5*. Unlike the previous release, the kernel component of OCFS2 is not packaged separately, but provided as part of the kernel rpm package. This is ideal: upgrading the kernel automatically upgrades the file system.

The user-space component includes two packages: ocfs2-tools (CLI) and ocfs2console (GUI). These packages are specific to a distribution and architecture only and have no kernel dependency. For example, ocfs2-tools-1.6.3-2.el5.x86_64.rpm is for the OL5/x86_64 platform regardless of the kernel version in use.

DOWNLOAD AND INSTALL

OCFS2 1.6 is only available with the *Unbreakable Enterprise Kernel*. Users wishing to upgrade to the new release must logon to the *Unbreakable Linux Network (ULN)* and subscribe to the ***Oracle Linux 5 Latest*** channel.

The subscriber can then use the *up2date* utility to upgrade to the Unbreakable Enterprise Kernel and to the latest ocfs2-tools.

```
# up2date kernel
# up2date oracle-linux
# up2date ocfs2-tools ocfs2console
```

CONFIGURE

OCFS2 volumes can be mounted as clustered or local (single-node) volumes. Users looking to mount volumes locally can skip the cluster configuration and go straight to formatting. Others need to configure the O2CB cluster.

The O2CB cluster stack configuration consists of the cluster layout and the cluster timeouts.

O2CB Cluster Layout Configuration

The cluster layout is specified in /etc/ocfs2/cluster.conf. It is easy to populate and propagate using the GUI utility, ocfs2console(8). It can also be done manually as long as care is taken to format the file correctly.

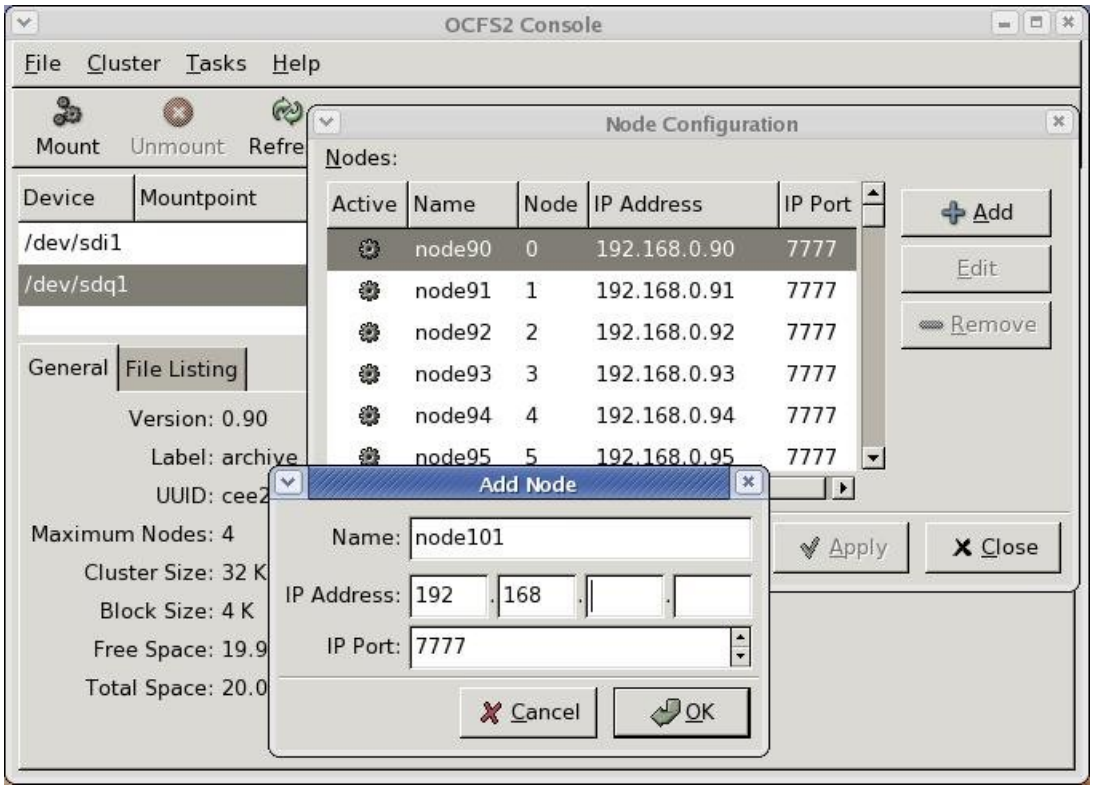
While the console utility is intuitive to use, there are few points to keep in mind.

1. The node name needs to match the *hostname*. It does not need to include the domain name. For example, for *appserver.oracle.com* can be *appserver*.

2. The IP address need *not* be the one associated with that *hostname*. That is, any valid IP address on that node can be used. O2CB will *not* attempt to match the node name (*hostname*) with the specified IP address.

For best performance, the use of a private interconnect (with lower latency) is highly recommended.

The one limitation of the console utility is that it cannot change the IP address and port of existing nodes. Such modifications require stopping the cluster stack and manually editing the configuration file on all nodes before restarting it. One should always ensure that the cluster.conf is the same on all nodes in the cluster.



The configuration file is in a stanza format with two types of stanzas: *cluster* and *node*. A typical cluster.conf will have one cluster stanza and multiple node stanzas.

The Cluster stanza has two parameters:

node_count	Total number of nodes in the cluster
name	Name of the cluster

The Node stanza has five parameters:

ip_port	IP port#
ip_address	IP address (preferably private interface)
number	Unique node number from 0-254
name	Hostname
cluster	Name of the cluster

Users populating cluster.conf manually should follow the format strictly. The stanza header must start at the first column and end with a colon, stanza parameters must start after a tab, and a blank like must separate each stanza. Care should be taken to avoid stray white spaces.

Example

The following is a sample /etc/ocfs2/cluster.conf that describes a three node cluster.

```
cluster:
	node_count = 3
	name = webcluster

node:
	ip_port = 7777
	ip_address = 192.168.0.107
	number = 7
	name = node7
	cluster = webcluster

node:
	ip_port = 7777
	ip_address = 192.168.0.106
	number = 6
	name = node6
	cluster = webcluster

node:
	ip_port = 7777
	ip_address = 192.168.0.110
	number = 10
	name = node10
	cluster = webcluster
```

O2CB Cluster Timeout Configuration

O2CB has four configurable cluster timeouts that are specified in /etc/sysconfig/o2cb. These can be configured using the o2cb init script.

```
# service o2cb configure
Configuring the O2CB driver.

This will configure the on-boot properties of the O2CB driver.
The following questions will determine whether the driver is loaded on
boot.  The current values will be shown in brackets ('[]').  Hitting
<ENTER> without typing an answer will keep that current value.  Ctrl-C
will abort.

Load O2CB driver on boot (y/n) [n]: y
Cluster stack backing O2CB [o2cb]:
Cluster to start on boot (Enter "none" to clear) [ocfs2]: webcluster
Specify heartbeat dead threshold (>=7) [31]:
Specify network idle timeout in ms (>=5000) [30000]:
Specify network keepalive delay in ms (>=1000) [2000]:
Specify network reconnect delay in ms (>=2000) [2000]:
Writing O2CB configuration: OK
Setting cluster stack "o2cb": OK
Starting O2CB cluster webcluster: OK
```

The O2CB cluster stack uses these timings to determine whether a node is dead or alive. While the use of default values is recommended, users can experiment with other values if the defaults cause spurious fencing.

The O2CB cluster timeouts are:

Heartbeat Dead Threshold

The disk heartbeat timeout is the number of two-second iterations before a node is considered dead. The exact formula used to convert the timeout in seconds to the number of iterations is:

```
O2CB_HEARTBEAT_THRESHOLD = (((timeout in seconds) / 2) + 1)
```

For example, to specify a 60 sec timeout, set it to 31. For 120 secs, set it to 61. The default for this is 31 (60 secs). A setting of 61 is recommended for multipath users.

Network Idle Timeout

The network idle timeout specifies the time in milliseconds before a network connection is considered dead. While it defaults to 30000ms, a setting of 60000ms has shown better results with many users.

Network Keepalive Delay

The network keepalive specifies the maximum delay in milliseconds before a keepalive packet is sent to another node. If the node is alive, it is expected to respond. It defaults to 2000 ms.

Network Reconnect Delay

The network reconnect delay specifies the minimum delay in milliseconds between connection attempts. It defaults to 2000 ms.

To view the currently active cluster timeout values, do:

```
# service o2cb status
Driver for "configfs": Loaded
Filesystem "configfs": Mounted
Stack glue driver: Loaded
Stack plugin "o2cb": Loaded
Driver for "ocfs2_dlmfs": Loaded
Filesystem "ocfs2_dlmfs": Mounted
Checking O2CB cluster webcluster: Online
Heartbeat dead threshold = 31
  Network idle timeout: 30000
  Network keepalive delay: 2000
  Network reconnect delay: 2000
Checking O2CB heartbeat: Not active
```

The o2cb init script has additional commands to manage the cluster. See help for the list of commands.

Kernel Configuration

Two sysctl values need to be set for O2CB to function properly. The first, *panic_on_oops*, must be enabled to turn a kernel oops into a panic. If a kernel thread required for O2CB to function crashes, the system must be reset to prevent a cluster hang. If it is not set, another node may not be able to distinguish whether a node is unable to respond or slow to respond.

The other related sysctl parameter is *panic*, which specifies the number of seconds after a panic that the system will be auto-reset. Setting this parameter to zero disables auto-reset; the cluster will require manual intervention. This is not preferred in a cluster environment.

To manually enable panic on oops and set a 30 sec timeout for reboot on panic, do:

```
# echo 1 > /proc/sys/kernel/panic_on_oops
# echo 30 > /proc/sys/kernel/panic
```

To enable the above on every reboot, add the following to /etc/sysctl.conf:

```
kernel.panic_on_oops = 1
kernel.panic = 30
```

OS Configuration

O2CB also requires iptables (firewalling) to be either disabled or modified to allow network traffic on the private network interface. The port used by O2CB is specified in /etc/ocfs2/cluster.conf.

START AND STOP O2CB

To start the O2CB cluster stack, do:

```
# service o2cb online
Loading filesystem "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading stack plugin "o2cb": OK
Loading filesystem "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Setting cluster stack "o2cb": OK
Starting O2CB cluster webcluster: OK
```

To check the status of the O2CB cluster stack, do:

```
# service o2cb status
Driver for "configfs": Loaded
Filesystem "configfs": Mounted
Stack glue driver: Loaded
Stack plugin "o2cb": Loaded
Driver for "ocfs2_dlmfs": Loaded
Filesystem "ocfs2_dlmfs": Mounted
Checking O2CB cluster webcluster: Online
Heartbeat dead threshold = 31
  Network idle timeout: 30000
  Network keepalive delay: 2000
  Network reconnect delay: 2000
Checking O2CB heartbeat: Not active
```

To stop and unload the O2CB cluster stack, do:

```
# service o2cb offline
Stopping O2CB cluster webcluster: OK

# service o2cb unload
Unmounting ocfs2_dlmfs filesystem: OK
Unloading module "ocfs2_dlmfs": OK
Unloading module "ocfs2_stack_o2cb": OK
Unloading module "ocfs2_stackglue": OK
Unmounting configfs filesystem: OK
Unloading module "configfs": OK
```

FORMAT

Like with any file system, a volume needs to be formatted before use. As formatting is the process of initializing a volume, it should be done with care. This is especially true in the OCFS2 environment; the volumes being formatted are shared resources and could be in use on another node.

OCFS2's format utility, mkfs.ocfs2(8), has checks in place to prevent overwriting volumes that are in use across the cluster. However, the checks only prevent overwriting existing OCFS2 volumes and will not prevent overwriting, say, an in-use ext3 volume. Thus, care should always be taken before formatting any volume.

In addition, while it is not required, it is preferred that the volume being formatted is partitioned. Not only are partitioned volumes less likely to be reused by mistake, some features like mount-by-label do not work with unpartitioned volumes. For more on partitioning, check the man pages for fdisk(8) or parted(8).

When run without any options, mkfs.ocfs2(8) heuristically determines the values of the various options. Users can help by providing broad hints, like file system type, that differentiate based on typical usages like *database* (fewer, fully allocated large files), *mail* (lots-of, small files), and *vmstore* (fewer, sparsely allocated large files). Other than the block and cluster sizes, all other options can be later modified using tunefs.ocfs2(8).

The most commonly used options for mkfs.ocfs2(8) are:

-b, --block-size block-size

The block size is the smallest unit of IO performed by the file system. It is also the size of inode and extent blocks and cannot be changed after format. The file system supports block sizes of 512 bytes, 1KB, 2KB and 4KB. 4KB is recommended for almost all users. 512 bytes is never recommended.

-C, --cluster-size cluster-size

The cluster size is the smallest unit of space allocated for file data. All data allocation is in multiples of the cluster size. OCFS2 supports cluster sizes of 4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB and 1MB. 4KB clusters are also recommended for almost all users. However, volumes storing database files should not use a value smaller than the database block size.

-N, --node-slots number-of-node-slots

A node slot refers to a set of system files, like a journal, that are used exclusively by a node. The number of node slots specifies how many nodes can mount the volume concurrently. This number can later be increased or decreased with tunefs.ocfs2(8). For performance reasons, it is recommended to create more node slots than required. This is because creating node slots later means the space allocated to the journal will neither be contiguous nor be on the outer edge of the disk platter. Both of these affect performance negatively.

-J, --journal-options options

OCFS2 uses write-ahead journal, JBD2, with a user-configurable size. If left unspecified, mkfs.ocfs2(8) determines the appropriate value based on the specified file system type and the volume size. The defaults are 64MB for *datafiles*, 128MB for *vmstore*, and 256MB for *mail*.

-L, --label volume-label

Labeling volumes is recommended for easier management. This is especially helpful in a clustered environment in which nodes may detect the devices in different order leading to the same device having different names on different nodes. Labeling allows consistent naming for OCFS2 volumes across a cluster.

-T filesystem-type

Valid types are *mail*, *datafiles* and *vmstore*. *mail* refers to its use as a mail server store that involves lots of metadata changes to lots of small files that benefits by using a larger journal. *datafiles*, on the other hand, suggests fewer fully allocated large files, requiring fewer metadata changes, thus not benefiting from a large journal. *vmstore*, as the name suggests, refers to virtual machine images that are sparsely allocated large files; these require moderate metadata updates.

--fs-features=[no]sparse...

Allows users to enable or disable certain file system features, including *sparse files*, *unwritten extents* and *back-up super blocks*. Refer to the chapter titled File system features for the list of supported features in this release.

--fs-feature-level=feature-level

Valid values are *max-compat*, *default* and *max-features*. *max-compat* enables only those features that are understood by older versions of the file system software. *max-features* is at the other end of the spectrum. It enables all the features that the file system software currently supports. *default* currently enables support for sparse files, unwritten extents, and inline-data.

Examples

To format with all defaults, including heuristically determined block and cluster sizes, the default number of node-slots, and the current default feature level, do:

```
# mkfs.ocfs2 -L "myvolume" /dev/sda1
mkfs.ocfs2 1.6.3
Cluster stack: classic o2cb
Label: myvolume
Features: sparse backup-super unwritten inline-data strict-journal-super
Block size: 4096 (12 bits)
Cluster size: 4096 (12 bits)
Volume size: 53687074816 (13107196 clusters) (13107196 blocks)
Cluster groups: 407 (tail covers 11260 clusters, rest cover 32256 clusters)
Extent allocator size: 8388608 (2 groups)
Journal size: 268435456
Node slots: 8
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 3 block(s)
Formatting Journals: done
Growing extent allocator: done
Formatting slot map: done
Formatting quota files: done
Writing lost+found: done
mkfs.ocfs2 successful
```

Notice the values chosen include 4K blocks and clusters, 4 node slots with a 256 MB journal each, and the default file system features.

To format volume for exclusive use as a database store, do:

```
# mkfs.ocfs2 -T datafiles -L "mydatavol" /dev/sde1
mkfs.ocfs2 1.6.3
Cluster stack: classic o2cb
Filesystem Type of datafiles
Label: mydatavol
Features: sparse backup-super unwritten inline-data strict-journal-super
Block size: 4096 (12 bits)
Cluster size: 131072 (17 bits)
Volume size: 53686960128 (409599 clusters) (13107168 blocks)
Cluster groups: 13 (tail covers 22527 clusters, rest cover 32256 clusters)
Extent allocator size: 20971520 (5 groups)
Journal size: 33554432
Node slots: 8
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 3 block(s)
Formatting Journals: done
Growing extent allocator: done
Formatting slot map: done
Formatting quota files: done
Writing lost+found: done
mkfs.ocfs2 successful
```

Notice the choice of a larger cluster size (128 K) and smaller journal size (32 M).

To format the volume with custom values (say, 4K block and cluster sizes, 8 node-slots with 128MB journals each, and with refcount trees and indexed directories), do:

```
# mkfs.ocfs2 -b4K -C 4K -J size=128M --fs-features=indexed-dirs,refcount /dev/sde1
mkfs.ocfs2 1.6.3
Cluster stack: classic o2cb
Label: ocfs2vol
Features: sparse backup-super unwritten inline-data strict-journal-super indexed-
dirs refcount
Block size: 4096 (12 bits)
Cluster size: 4096 (12 bits)
Volume size: 53687074816 (13107196 clusters) (13107196 blocks)
Cluster groups: 407 (tail covers 11260 clusters, rest cover 32256 clusters)
Extent allocator size: 8388608 (2 groups)
Journal size: 134217728
Node slots: 8
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 3 block(s)
Formatting Journals: done
Growing extent allocator: done
Formatting slot map: done
Formatting quota files: done
Writing lost+found: done
mkfs.ocfs2 successful
```

Notice the format utility also enabled the default file system features (sparse, unwritten and inline-data). These could have been disabled by prefixing them with **no** (for example, *nosparse*).

MOUNT

This section assumes the user has formatted the volume. If it is a clustered volume, it is assumed that the O2CB cluster has been started.

The commands to mount and umount OCFS2 volumes are similar to other file systems.

```
$ mount /dev/sda1 /dir
…
$ umount /dir
```

Users mounting a clustered volume should be aware of the following:

1. The cluster stack must to be online for a clustered mount to succeed.

2. The clustered mount operation is not instantaneous; it must wait for the node to join the DLM domain.

3. Likewise, clustered umount is also not instantaneous, as it involves migrating all mastered lock-resources to the other nodes in the cluster.

If the mount fails, detailed errors can be found via dmesg(8). These might include incorrect cluster configuration (say, a missing node or incorrect IP address) or a firewall interfering with O2CB network traffic.

To auto-mount volumes on startup, the file system tools include an *ocfs2* init service. This runs after the *o2cb* init service has started the cluster. The *ocfs2* init service mounts all OCFS2 volumes listed in /etc/fstab.

Mount Options

The file system supports many mount options supported by other Linux file systems. These supported options are:

_netdev
: The file system resides on a device that requires network access (used to prevent the system from attempting to mount these file systems until the network has been enabled on the system). mount.ocfs2(8) transparently appends this option during mount. However, users mounting the volume via /etc/fstab must explicitly specify this mount option. This prevents the system from mounting the volume until after the network has been enabled. Conversely, during shutdown, it instructs the system to unmount the volume before shutting down the network.

atime_quantum=<nrsecs>
: This instructs the file system to limit the granularity of atime updates to *nrsecs* second. The default is 60 secs. A low value will hurt performance as atime is updated on every read or write access. To always update atime, set it to zero.

barrier=1

This enables/disables barriers: barrier=0 disables, barrier=1 enables. Barriers are disabled by default.

commit=<nrsecs>

This instructs the file system to sync all data and metadata every *nrsecs* seconds. The default value is 5 seconds. This means that if you lose your power, you will lose as much as the latest 5 seconds of work (your file system will not be damaged though, thanks to journaling). This default value (or any low value) will hurt performance, but it is good for data-safety. Setting it to 0 will have the same effect as leaving it at the default (5 seconds). Setting it to very large values will improve performance at the expense of greater data-loss.

data=ordered / data=writeback

This specifies the handling of data during metadata journaling.

ordered

This is the default mode. All data is flushed to disk before the metadata is committed to the journal. This prevents a case where stale data can appear in a file after a crash.

writeback

Data ordering is not preserved - data may be flushed to disk after its meta-data has been committed to the journal. This is rumored to be the highest throughput option. While it guarantees internal file system integrity, it can allow null or stale data to appear in files after a crash and journal recovery. This can happen if the system crashes after the journal commit but before the data flush. Ordered data mode journaling avoids this issue by always flushing the data before the commit.

datavolume

Use this mount option to mount volumes storing Oracle data files, control files, redo logs, archive logs, voting disk, cluster registry, etc.

errors=remount-ro / errors=panic

This defines the behavior when an error (on-disk corruption) is encountered. (Either remount the file system read-only or panic and halt the system.) By default, the file system is remounted read-only.

intr / nointr

The default, *nointr*, blocks signals from interrupting certain cluster operations.

localflocks

This disables cluster-aware flock(2).

noatime

This standard mount option turns off atime updates completely.

relatime

Relative atime only updates the atime if the previous atime is older than the mtime or ctime. This is useful for applications that only need to know that a file has been read since it was last modified.

ro

This mounts the file system read-only.

rw

This mounts the file system read-write.

Examples

If mounting clustered volumes, start the O2CB cluster service before attempting any mounts.

```
# service o2cb online
```

To mount device /dev/sda1 at /u01, do:

```
# mount /dev/sda1 /u01
```

To umount the device mounted at /u01, do:

```
# umount /u01
```

To mount OCFS2 volumes automatically at boot, (a) enable *o2cb* and *ocfs2* init services to start on boot, and (b) add the mount entries in /etc/fstab.

```
# chkconfig --add o2cb
o2cb 0:off 1:off 2:on 3:on 4:off 5:on 6:off

$ chkconfig --add ocfs2
o2cb 0:off 1:off 2:on 3:on 4:off 5:on 6:off

$ cat /etc/fstab
…
/dev/sda1  /u01  ocfs2      _netdev,defaults      0     0
…
```

The *_netdev* mount option is required for OCFS2 volumes. This mount option instructs the operating system to mount the volume after the network is started and unmount it before the network is stopped.

To mount-by-label a volume labeled "myvolume", do:

```
# mount -L myvolume /u01
```

VI. ADMINISTRATION

The previous section covered configuring and starting the cluster, as well as, formatting and mounting the volume. This section deals with the administrative operations that can be performed on the volume. This includes tuning, file system check, etc.

TUNING

tunefs.ocfs2(8) can change most of the file system parameters. In fact, every parameter except block and cluster sizes can be modified. As the tool modifies an existing OCFS2 volume, the cluster must be online in order to detect if the volume is in use on another node. If in use, the tune utility restricts itself to tasks that can be performed online.

The most commonly used options for tunefs.ocfs2(8) are:

--fs-features=[no]sparse...
Toggle file system features on and off. Refer to the chapter titled *File System Features* for more.

-J, --journal-options options
Grow or shrink the size of the journals.

-L, --label volume-label
Change the volume label.

-N, --node-slots number-of-node-slots
Increase or decrease the number of node slots. Node slots dictate the number of nodes that can concurrently mount a volume. Having the ability to increase the node slots in useful in any environment. It also allows users to remove node slots in order to recover space used by journals. This is currently an offline operation.

-S, --volume-size
Grow the size of the volume. This is can be performed both when the volume is online and offline. However, as it requires a clustered volume manager to work effectively, the online feature will only be useful when support for a volume manager is announced. We expect to have it available with the next release. The utility does not shrink the size of the volume.

-U, --uuid-reset[=new-uuid]
Change the volume UUID to either a auto-generated one or user-specified. When user-specified, care should be taken to provide a unique UUID.

--cloned-volume[=new-label]
Change the volume UUID (auto-generated) and the label, if provided, of a cloned OCFS2 volume. This option does not perform volume cloning. It only changes the

UUID and label on a cloned volume so that it can be mounted on the same node as the original volume.

FILE SYSTEM CHECK

fsck.ocfs2(8) is the file system check tool. It detects and fixes on-disk errors. It expects the cluster to be online as it needs to ensure the volume is not in use on another node. It also locks out the volume preventing other nodes from mounting the volume for the duration of the check.

When run with no options, fsck.ocfs2(8) only replays the journals. For a full scan, specify –f to force-check the volume.

```
# fsck.ocfs2 -f /dev/sde1
fsck.ocfs2 1.6.3
Checking OCFS2 filesystem in /dev/sde1:
  Label:              ocfs2vol
  UUID:               088010E5D3704AD396C5F97567CB168B
  Number of blocks:   13107196
  Block size:         4096
  Number of clusters: 13107196
  Cluster size:       4096
  Number of slots:    8

/dev/sde1 was run with -f, check forced.
Pass 0a: Checking cluster allocation chains
Pass 0b: Checking inode allocation chains
Pass 0c: Checking extent block allocation chains
Pass 1: Checking inodes and blocks.
Pass 2: Checking directory entries.
Pass 3: Checking directory connectivity.
Pass 4a: checking for orphaned inodes
Pass 4b: Checking inodes link counts.
All passes succeeded.
```

The man page fsck.ocfs2.checks(8) lists all the checks performed by fsck.ocfs2(8).

Backup Super blocks

A file system super block stores critical information that is hard to recreate. In OCFS2, it stores the block size, cluster size, and the locations of the root and system directories, among other things. As this block is close to the start of the disk, it is very susceptible to being overwritten by an errant write. Say, *dd if=file of=/dev/sda1*.

Backup super blocks are copies of the actual super block. These blocks are dispersed in the volume to minimize the chances of being overwritten. On the small chance that the original gets corrupted, the backups are available to scan and fix the corruption.

mkfs.ocfs2(8) enables this feature by default. Users wishing not to have them can specify *–fs-features=nobackup-super* during format.

tunefs.ocfs2(8) can be used to view whether the feature has been enabled on a device.

```
# tunefs.ocfs2 -Q "%M\n" /dev/sde1
backup-super strict-journal-super
```

In OCFS2, the super block is on the third block. The backups are located at the 1 GB, 4 GB, 16 GB, 64 GB, 256 GB and 1 TB byte offsets. The actual number of backup blocks depends on the size of the device. The super block is not backed up on devices smaller than 1 GB.

fsck.ocfs2(8) refers to these six offsets by numbers, 1 to 6. Users can specify any backup with the *–r* option to recover the volume. The example below uses the second backup. If successful, fsck.ocfs2(8) overwrites the corrupted super block with the backup.

```
# fsck.ocfs2 -f -r 2 /dev/sde1
fsck.ocfs2 1.6.3
[RECOVER_BACKUP_SUPERBLOCK] Recover superblock information from backup block#1048576? <n> y
Checking OCFS2 filesystem in /dev/sde1:
  Label:              ocfs2vol
  UUID:               088010E5D3704AD396C5F97567CB168B
  Number of blocks:   13107196
  Block size:         4096
  Number of clusters: 13107196
  Cluster size:       4096
  Number of slots:    8

/dev/sde1 was run with -f, check forced.
Pass 0a: Checking cluster allocation chains
Pass 0b: Checking inode allocation chains
Pass 0c: Checking extent block allocation chains
Pass 1: Checking inodes and blocks.
Pass 2: Checking directory entries.
Pass 3: Checking directory connectivity.
Pass 4a: checking for orphaned inodes
Pass 4b: Checking inodes link counts.
All passes succeeded.
```

OTHER TOOLS

The previous sections have covered tools to mount, format, tune, and check an OCFS2 volume. This section briefly reviews the remaining tools. Complete documentation for all tools is available in their corresponding man pages.

mounted.ocfs2(8)

This tool detects all OCFS2 volumes. It does so by scanning all the devices listed in /proc/partitions. It has two modes. In the detect (*-d*) mode, it lists all OCFS2 devices.

```
# mounted.ocfs2 -d
Device        FS     Stack  UUID                              Label
/dev/sda1     ocfs2  o2cb   CB637503B1A746DEB3D63656E16FA41B  myvol
/dev/sdc1     ocfs2  o2cb   189D6324084B4B98B4560CA5736E6641  ebizvol
/dev/sde1     ocfs2  o2cb   088010E5D3704AD396C5F97567CB168B  backup
/dev/sdg1     ocfs2  o2cb   D18BCF34BDD940C4BFECF98277A3CD74  archives
/dev/sdk1     ocfs2  o2cb   D2552356DFAC4C2E90008333BEA5C60D  vmstore
```

In the full (*-f*) mode, it lists the nodes currently mounting each volume. However, it should be noted that the information is not always accurate. The information is gleaned by dirty-reading the slot-map on the volume, which may be corrupted if the last node to mount the volume crashed. A corrupted slot-map is recovered by the next mount.

```
$ mounted.ocfs2 -f
Device        FS     Nodes
/dev/sda1     ocfs2  node96, node92
/dev/sdc1     ocfs2  node40, node35, node32, node31, node34, node33
/dev/sde1     ocfs2  Not mounted
/dev/sdg1     ocfs2  node40, node35, node32, node31, node34, node33
/dev/sdg1     ocfs2  node40, node35, node32, node31
```

o2cb_ctl(8)

This is the tool used by the o2cb init script to populate the O2CB cluster. It can also be used to **add new nodes** to a running cluster.

To add the host *node4* as node number *4* with IP address *192.168.0.104* using port *7777* to the cluster *webcluster*, do:

```
# o2cb_ctl -C -i -n node4 -t node -a number=4 -a ip_address=192.168.0.104 \
-a ip_port=7777 -a cluster=webcluster
```

This command needs to be run on **all** nodes in the cluster, and the updated /etc/ocfs2/cluster.conf must be copied to the new node. This configuration file needs to be consistent on all nodes.

debugfs.ocfs2(8)

This is main debugging tool. It allows users to walk directory structures, print inodes, backup files, etc., all without mounting the file system. This tool has been modeled after ext3's debugfs. For more, refer to the man page and the *On-Disk Format* support guide downloadable from the OCFS2 documentation section.

o2image(8)

o2image allows users to backup the OCFS2 file system meta-data from a device to a specified image-file. This image file contains the file system skeleton, including the inodes, directory names, and file names. It does not include any file data.

This image file can be useful in debugging on-disk corruptions that may not reproducible otherwise. It also helps us study on-disk layouts.

VII. ORACLE RDBMS

One of the earliest uses of OCFS2 was with Oracle RAC on Linux. Since then, the file system has also seen use with the standalone database product. It provides advantages over other local file systems on Linux, including efficient handling of large files with full direct and asynchronous I/O support and the ability to convert the file system from local to clustered and back.

This section deals with the additional configuration and issues one needs to be aware of when using OCFS2 with the Oracle RDBMS.

Mount Options

Volumes holding the Oracle RDBMS files should be mounted with three mount options: **n*oatime*** to disable access time updates, ***datavolume*** to force the database to use direct I/O, and ***nointr*** to disable interrupts.

It is necessary to mount volumes hosting the Oracle datafiles, control files, redo logs, voting disk, ocr, etc. with the *noatime* mount option. This disables unnecessary updates to the access time (atime) on the inodes.

The *nointr* mount option disables signals interrupting I/Os in progress. This mount option is enabled by default starting in OCFS2 Release 1.6.

The *datavolume* mount option is more legacy than anything else. It directs the database to perform direct I/O to the Oracle datafiles, control files, redo logs, etc. to such files on volumes mounted with this option. It is legacy mainly because the same behavior can be enforced by the init.ora parameter, *filesystemio_options*. Users using that parameter do not need this mount option for volumes hosting the database files. However, a volume hosting the Oracle Clusterware's voting disk file and the cluster registry still requires this option.

It should be noted that the *datavolume* mount option is only available in the OCFS2 Releases 1.2, 1.4 and 1.6 for Enterprise distributions. It is not available on other distributions or the mainline kernel. The other two mount options are available on all distributions.

The *datavolume* mount option must **not** be used on volumes hosting the Oracle home or Oracle E-Business Suite or any other use.

The following example is of an entry in /etc/fstab for a volume hosting Oracle RDBMS files.

```
/dev/sda1         /u01  ocfs2       noatime,datavolume,nointr     0     0
```

Timeouts

Oracle RAC uses its own cluster stack, CSS. Both cluster stacks, CSS and O2CB, have configurable timeouts. In later versions of CSS (late 10g and 11g), care has been taken to ensure that the two timeouts are unrelated. This is true when using OCFS2 to host the voting disk files and the cluster registry but not the Oracle Grid Infrastructure home, which should be installed on a local file system.

Node Numbers

It is best if the node numbers in the two stacks are consistent. Both stacks use the lower node number as a tie-breaker in quorum calculations. Changing node numbers in O2CB involves editing /etc/ocfs2/cluster.conf and propagating the new configuration to all nodes. The new numbers take effect once the cluster is restarted.

Cluster Sizes

When specifying *datafiles* as the file system type, mkfs.ocfs2(8) sets the cluster size to 128KB. However, there is no one correct value and users are free to use a different value. The one point to remember is that the cluster size should not be smaller than the database block size. This is the easiest way to ensure that the database blocks will not be fragmented on disk. As 8K is the typical database block size, use a cluster size of at least that value, if not larger.

Modification Times

To allow multiple nodes to concurrently stream I/Os to an Oracle datafile, OCFS2 makes a special dispensation from the POSIX standard by not updating the modification time (mtime) on disk when performing non-extending direct I/O writes. To be precise, while the new mtime is updated in memory, it is not flushed to disk unless the user extends or truncates the file or performs an explicit operation, such as touch(1). This dispensation leads to the file system returning differing time stamps for the same file on different nodes. While this is not ideal, this behavior exists to allow maximum throughput. Updating mtime on each write would negate one of the main benefits (parallel I/O) of a clustered database, because it would serialize the I/Os to each datafile.

User wishing to view the on-disk timestamp of any file can use the debugfs.ocfs2 tool as follows:

```
$ debugfs.ocfs2 -R "stat /relative/path/to/file" /dev/sda1 | grep "mtime:"
```

Certification

The current information on certifications is available on the Oracle Technology Network. http://www.oracle.com/technetwork/index.html. For information on Oracle Clusterware, go to its home page on OTN and look for links for the certification matrix.

VIII. NOTES

1. Balanced Cluster

A cluster is a computer. This is a fact and not a slogan. What this means is that an errant node in the cluster can affect the behavior of other nodes. If one node is slow, the cluster operations will slow down on all nodes. To prevent that, it is best to have a balanced cluster. This is a cluster that has equally powered and loaded nodes.

The standard recommendation for such clusters is to have identical hardware and software across all the nodes.

However, that is not a hard and fast rule. After all, we have taken the effort to ensure that OCFS2 works in a mixed architecture environment.

If one uses OCFS2 in a mixed architecture environment, try to ensure that the nodes are equally powered and loaded. The use of a load balancer can assist with the latter. Power refers to the number of processors, speed, amount of memory, I/O throughput, network bandwidth, etc. In reality, having equally powered heterogeneous nodes is not always practical. In that case, make the lower node numbers more powerful than the higher node numbers. The O2CB cluster stack favors lower node numbers in all of its tie-breaking logic.

This is not to suggest you should add a single core node in a cluster of quad cores. No amount of node number juggling will help you there.

2. File Deletion

In Linux, rm(1) removes the directory entry. It does not necessarily delete the corresponding inode. By removing the directory entry, it gives the illusion that the inode has been deleted. This puzzles users when they do not see a corresponding up-tick in the reported free space. The reason is that inode deletion has a few more hurdles to cross.

First is the hard link count. This indicates the number of directory entries pointing to that inode. As long as a directory entry is linked to that inode, it cannot be deleted. The file system has to wait for that count to drop to zero.

The second hurdle is the Linux/Unix semantics allowing files to be unlinked even while they are in use. In OCFS2, that translates to in use across the cluster. The file system has to wait for all processes across the cluster to stop using the inode.

Once these two conditions are met, the inode is deleted and the freed bits are flushed to disk on the next sync.

This assumes that the inode was not reflinked. If it was, then the deletion would only release space that was private to the inode. Shared space would only be released when the last inode using it is deleted.

Users interested in following the trail can use debugfs.ocfs2(8) to view the node specific system files *orphan_dir* and *truncate_log*. Once the link count is zero, an inode is moved to the *orphan_dir*. After deletion, the freed bits are added to the *truncate_log,* where they remain until the next sync, during which the bits are flushed to the global bitmap.

3. Directory Listing

ls(1) may be a simple command, but it is not cheap. What is expensive is not the part where it reads the directory listing, but the second part where it reads all the inodes, also referred as an inode stat(2). If the inodes are not in cache, this can entail disk I/O.

Now, while a cold cache inode stat(2) is expensive in all file systems, it is especially so in a clustered file system. It needs to take a lock on each node, pure overhead when compared to a local file system.

A hot cache stat(2), on the other hand, has shown to perform on OCFS2 like it does on EXT3.

In other words, the second ls(1) will be quicker than the first. However, it is not guaranteed. Say you have a million files in a file system and not enough kernel memory to cache all the inodes. In that case, each ls(1) will involve some cold cache stat(2)s.

4. Allocation Reservation

Allocation reservation is a new feature in OCFS2 Release 1.6 that allows multiple concurrently extending files to grow as contiguously as possible.

One way to demonstrate its functioning is to run a script that extends multiple files in a circular order. The script below does that by writing one hundred 4KB chunks to four files, one after another.

```
$ for i in $(seq 0 99);
> do
>   for j in $(seq 4);
>   do
>     dd if=/dev/zero of=file$j bs=4K count=1 seek=$i;
>   done;
> done;
```

When run on a system running OCFS2 Release 1.4, we end up with files with 100 extents each. That is full fragmentation. As the files are being extended one after another, the on-disk allocations are fully interleaved.

```
$ filefrag file1 file2 file3 file4
file1: 100 extents found
file2: 100 extents found
file3: 100 extents found
file4: 100 extents found

# debugfs.ocfs2 -R "frag mydir/file1" /dev/sdd1
Inode: 541228  % fragmented: 100.00  clusters: 100  extents: 100  score: 25600
```

When run on a system running OCFS2 Release 1.6, we see files with 7 extents each. That's a lot fewer than in the previous release. Fewer extents mean more on-disk contiguity that always leads to better overall performance.

```
$ filefrag file1 file2 file3 file4
file1: 7 extents found
file2: 7 extents found
file3: 7 extents found
file4: 7 extents found

# debugfs.ocfs2 -R "frag mydir/file1" /dev/sda1
Inode: 646212   % fragmented: 7.00   clusters: 100   extents: 7   score: 1792
```

5. REFLINK

This is a new feature added in OCFS2 Release 1.6 that allows a user to create a write-able snapshot of a regular file. In this operation, the file system creates a new inode with the same extent pointers as the original inode. Multiple inodes are thus able to share data extents. This adds a twist in file system administration because none of the existing file system utilities in Linux expect this behavior. du(1), a utility to used to compute file space usage, simply adds the blocks allocated to each inode. As it does not know about shared extents, it over estimates the space used.

Say, we have a 5GB file in a volume having 42GB free.

```
$ ls -l
total 5120000
-rw-r--r-- 1 jeff jeff 5242880000 Sep 24 17:15 myfile

$ du -m myfile*
5000    myfile

$ df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/sdd1              50G  8.2G   42G  17% /ocfs2
```

If we were to reflink it 4 times, we would expect the directory listing to report five 5GB files, but the df(1) to report no loss of available space. du(1), on the other hand, would report the disk usage to climb to 25GB.

```
$ reflink myfile myfile-ref1
$ reflink myfile myfile-ref2
$ reflink myfile myfile-ref3
$ reflink myfile myfile-ref4

$ ls -l
total 25600000
-rw-r--r-- 1 root root 5242880000 Sep 24 17:15 myfile
-rw-r--r-- 1 root root 5242880000 Sep 24 17:16 myfile-ref1
-rw-r--r-- 1 root root 5242880000 Sep 24 17:16 myfile-ref2
-rw-r--r-- 1 root root 5242880000 Sep 24 17:16 myfile-ref3
-rw-r--r-- 1 root root 5242880000 Sep 24 17:16 myfile-ref4

$ df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/sdd1              50G  8.2G   42G  17% /ocfs2

$ du -m -c myfile*
5000    myfile
5000    myfile-ref1
5000    myfile-ref2
5000    myfile-ref3
5000    myfile-ref4
25000   total
```

Enter shared-du(1), a shared extent–aware du. This utility reports the shared extents per file in parenthesis and the overall footprint. As expected, it lists the overall footprint at 5GB.

```
$ shared-du -m -c --shared-size myfile*
5000    (5000)  myfile
5000    (5000)  myfile-ref1
5000    (5000)  myfile-ref2
5000    (5000)  myfile-ref3
5000    (5000)  myfile-ref4
25000   total
5000    footprint
```

This utility is available at this link (http://oss.oracle.com/~smushran/reflink-tools/). Also available is a shared extent-aware filefrag utlity that lists the location of the extents on the volume.

We are currently in the process of pushing the changes to the upstream maintainers of these utilities.

```
# shared-filefrag -v myfile
Filesystem type is: 7461636f
File size of myfile is 5242880000 (1280000 blocks, blocksize 4096)
 ext logical physical expected length flags
   0       0  2247937            8448
   1    8448  2257921  2256384  30720
   2   39168  2290177  2288640  30720
   3   69888  2322433  2320896  30720
   4  100608  2354689  2353152  30720
   7  192768  2451457  2449920  30720
...
  37 1073408  2032129  2030592  30720 shared
  38 1104128  2064385  2062848  30720 shared
  39 1134848  2096641  2095104  30720 shared
  40 1165568  2128897  2127360  30720 shared
  41 1196288  2161153  2159616  30720 shared
  42 1227008  2193409  2191872  30720 shared
  43 1257728  2225665  2224128  22272 shared,eof
myfile: 44 extents found
```

6. Data Coherency

One of the challenges in a shared file system is data coherency when multiple nodes are writing to the same set of files. NFS, for example, provides close-to-open data coherency that results in the data being flushed to the server when the file is closed on the client. This leaves open a wide window for stale data being read on another node.

A simple test to check the data coherency of a shared file system involves concurrently appending the same file. Like running "uname –a >>/dir/file" using a parallel distributed shell like dsh or pconsole. If coherent, the file will contain the results from all nodes.

```
# dsh -R ssh –wnode32,node33,node34,node35 "uname –a >> /ocfs2/test"

# cat /ocfs2/test
Linux node32 2.6.32-100.0.19.el5 #1 SMP Fri Sep 17 17:51:41 EDT 2010 x86_64 x86_64 x86_64 GNU/Linux
Linux node35 2.6.32-100.0.19.el5 #1 SMP Fri Sep 17 17:51:41 EDT 2010 x86_64 x86_64 x86_64 GNU/Linux
Linux node33 2.6.32-100.0.19.el5 #1 SMP Fri Sep 17 17:51:41 EDT 2010 x86_64 x86_64 x86_64 GNU/Linux
Linux node34 2.6.32-100.0.19.el5 #1 SMP Fri Sep 17 17:51:41 EDT 2010 x86_64 x86_64 x86_64 GNU/Linux
```

OCFS2 is a fully coherent cluster file system.

7. Synthetic File Systems

The OCFS2 development effort included two synthetic file systems, configfs and dlmfs. It also makes use of a third, debugfs.

configfs

configfs has since been accepted as a generic kernel component and is also used by netconsole and fs/dlm. OCFS2 tools use it to communicate the list of nodes in the

cluster, details of the heartbeat device, cluster timeouts, and so on to the in-kernel node manager. The o2cb init script mounts this file system at /sys/kernel/config.

dlmfs

dlmfs exposes the in-kernel o2dlm to the user-space. While it was developed primarily for OCFS2 tools, it has seen usage by others looking to add a cluster-locking dimension in their applications. Users interested in doing the same should look at the libo2dlm library provided by ocfs2-tools. The o2cb init script mounts this file system at /dlm.

debugfs

OCFS2 uses debugfs to expose its in-kernel information to user space. For example, listing the file system cluster locks, dlm locks, dlm state, o2net state, etc. Users can access the information by mounting the file system at /sys/kernel/debug. To auto-mount, add the following to /etc/fstab:

```
debugfs          /sys/kernel/debug     debugfs    defaults   0 0
```

8. Distributed Lock Manager

One of the key technologies in a cluster is the lock manager, which maintains the locking state of all resources across the cluster. An easy implementation of a lock manager involves designating one node to handle everything. In this model, if a node wanted to acquire a lock, it would send the request to the lock manager. However, this model has a weakness: lock manager's death causes the cluster to seize up. A better model is one where all nodes manage a subset of the lock resources. Each node maintains enough information for all the lock resources it is interested in. On event of a node death, the remaining nodes pool in the information to reconstruct the lock state maintained by the dead node. In this scheme, the locking overhead is distributed amongst all the nodes. Hence, the term distributed lock manager.

O2DLM is a distributed lock manager. It is based on the specification titled "Programming Locking Application" written by Kristin Thomas and is available at the following link. http://opendlm.sourceforge.net/cvsmirror/opendlm/docs/dlmbook_final.pdf

9. DLM Debugging

O2DLM has a rich debugging infrastructure that allows it to show the state of the lock manager, all the lock resources, among other things.

The figure below shows the dlm state of a nine-node cluster that has just lost three nodes: 12, 32, and 35. It can be ascertained that node 7, the recovery master, is currently recovering node 12 and has received the lock states of the dead node from all other live nodes.

```
# cat /sys/kernel/debug/o2dlm/45F81E3B6F2B48CCAAD1AE7945AB2001/dlm_state
Domain: 45F81E3B6F2B48CCAAD1AE7945AB2001  Key: 0x10748e61
Thread Pid: 24542  Node: 7  State: JOINED
Number of Joins: 1  Joining Node: 255
Domain Map: 7 31 33 34 40 50
Live Map: 7 31 33 34 40 50
Lock Resources: 48850 (439879)
MLEs: 0 (1428625)
  Blocking: 0 (1066000)
  Mastery: 0 (362625)
  Migration: 0 (0)
Lists: Dirty=Empty  Purge=Empty  PendingASTs=Empty  PendingBASTs=Empty
Purge Count: 0  Refs: 1
Dead Node: 12
Recovery Pid: 24543  Master: 7  State: ACTIVE
Recovery Map: 12 32 35
Recovery Node State:
        7 - DONE
        31 - DONE
        33 - DONE
        34 - DONE
        40 - DONE
        50 - DONE
```

The figure below shows the state of a dlm lock resource that is mastered (owned) by node 25, with 6 locks in the granted queue and node 26 holding the EX (writelock) lock on that resource.

```
# debugfs.ocfs2 –R “dlm_locks M000000000000000022d63c00000000” /dev/sda1
Lockres: M000000000000000022d63c00000000   Owner: 25   State: 0x0
Last Used: 0      ASTs Reserved: 0    Inflight: 0    Migration Pending: No
Refs: 8    Locks: 6    On Lists: None
Reference Map: 26 27 28 94 95
 Lock-Queue  Node  Level  Conv  Cookie           Refs  AST  BAST  Pending-Action
 Granted     94    NL     -1    94:3169409       2     No   No    None
 Granted     28    NL     -1    28:3213591       2     No   No    None
 Granted     27    NL     -1    27:3216832       2     No   No    None
 Granted     95    NL     -1    95:3178429       2     No   No    None
 Granted     25    NL     -1    25:3513994       2     No   No    None
 Granted     26    EX     -1    26:3512906       2     No   No    None
```

The figure below shows a lock from the file system perspective. Specifically, it shows a lock that is in the process of being upconverted from a NL to EX. Locks in this state are are referred to in the file system as busy locks and can be listed using the debugfs.ocfs2 command, “fs_locks –B”.

```
# debugfs.ocfs2 –R “fs_locks –B” /dev/sda1
Lockres: M000000000000000000000b9aba12ec  Mode: No Lock
Flags: Initialized Attached Busy
RO Holders: 0  EX Holders: 0
Pending Action: Convert  Pending Unlock Action: None
Requested Mode: Exclusive  Blocking Mode: No Lock
PR > Gets: 0  Fails: 0    Waits (usec) Total: 0  Max: 0
EX > Gets: 440247  Fails: 0    Waits (usec) Total: 24104335  Max: 2431630
Disk Refreshes: 1
```

With this debugging infrastructure in place, users can debug hang issues as follows:

- Dump the busy fs locks for all the OCFS2 volumes on the node with hanging processes. If no locks are found, then the problem is not related to O2DLM.
- Dump the corresponding dlm lock for all the busy fs locks. Note down the owner (master) of all the locks.
- Dump the dlm locks on the master node for each lock.

At this stage, one should note that the hanging node is waiting to get an AST from the master. The master, on the other hand, cannot send the AST until the current holder has down converted that lock, which it will do upon receiving a Blocking AST. However, a node can only down convert if all the lock holders have stopped using that lock.

After dumping the dlm lock on the master node, identify the current lock holder and dump both the dlm and fs locks on that node.

The trick here is to see whether the Blocking AST message has been relayed to file system. If not, the problem is in the dlm layer. If it has, then the most common reason would be a lock holder, the count for which is maintained in the fs lock.

At this stage, printing the list of process helps.

```
$ ps -e -o pid,stat,comm,wchan=WIDE-WCHAN-COLUMN
```

Make a note of all D state processes. At least one of them is responsible for the hang on the first node.

The challenge then is to figure out why those processes are hanging. Failing that, at least get enough information (like alt-sysrq t output) for the kernel developers to review.

What to do next depends on where the process is hanging. If it is waiting for the I/O to complete, the problem could be anywhere in the I/O subsystem, from the block device layer through the drivers to the disk array. If the hang concerns a user lock (flock(2)), the problem could be in the user's application. A possible solution could be to kill the holder. If the hang is due to tight or fragmented memory, free up some memory by killing non-essential processes.

The thing to note is that the symptom for the problem was on one node but the cause is on another. The issue can only be resolved on the node holding the lock. Sometimes, the best solution will be to reset that node. Once killed, the O2DLM recovery process will clear all locks owned by the dead node and let the cluster continue to operate. As harsh as that sounds, at times it is the only solution. The good news is that, by following the trail, you now have enough information to file a bug and get the real issue resolved.

10. NFS

OCFS2 volumes can be exported as NFS volumes. This support is limited to NFS version 3, which translates to Linux kernel version 2.4 or later. NFS clients must mount volumes

using the *nordirplus* mount option. This disables the READDIRPLUS RPC call to workaround a bug in NFSD, detailed in the following link: http://oss.oracle.com/pipermail/ocfs2-announce/2008-June/000025.html

Users running NFS version 2 can export the volume after having disabled subtree checking (mount option *no_subtree_check*). Be warned, disabling the check has security implications (documented in the exports(5) man page) that users must evaluate on their own.

11. Limits

OCFS2 1.6 has no intrinsic limit on the total number of files and directories in the file system. In general, it is only limited by the size of the device. But there is one limit imposed by the current filesystem. It can address at most 2^{32} (approximately four billion) clusters. A file system with 1MB cluster size can go up to 4PB, while a file system with a 4KB cluster size can address up to 16TB.

12. System Objects

The OCFS2 file system stores its internal meta-data, including bitmaps, journals, etc., as system files. These are grouped in a system directory. These files and directories are not accessible via the file system interface but can be viewed using the debugfs.ocfs2(8) tool. To list the system directory (referred to as double-slash), do:

```
# debugfs.ocfs2 -R "ls -l //" /dev/sde1
        6       drwxr-xr-x   4  0  0          3896 21-Sep-2010 18:00 .
        6       drwxr-xr-x   4  0  0          3896 21-Sep-2010 18:00 ..
        7       -rw-r--r--   1  0  0             0 21-Sep-2010 18:00 bad_blocks
        8       -rw-r--r--   1  0  0       1101824 21-Sep-2010 18:00 global_inode_alloc
        9       -rw-r--r--   1  0  0          4096 21-Sep-2010 18:00 slot_map
        10      -rw-r--r--   1  0  0       1048576 21-Sep-2010 18:00 heartbeat
        11      -rw-r--r--   1  0  0   53687074816 21-Sep-2010 18:00 global_bitmap
        12      -rw-r--r--   1  0  0         24576 21-Sep-2010 18:00 aquota.user
        13      -rw-r--r--   1  0  0         24576 21-Sep-2010 18:00 aquota.group
        14      drwxr-xr-x   2  0  0          3896 21-Sep-2010 18:00 orphan_dir:0000
        15      drwxr-xr-x   2  0  0          3896 21-Sep-2010 18:00 orphan_dir:0001
        16      -rw-r--r--   1  0  0      29360128 21-Sep-2010 18:00 extent_alloc:0000
        17      -rw-r--r--   1  0  0      29360128 21-Sep-2010 18:00 extent_alloc:0001
        18      -rw-r--r--   1  0  0       4194304 21-Sep-2010 18:00 inode_alloc:0000
        19      -rw-r--r--   1  0  0             0 21-Sep-2010 18:00 inode_alloc:0001
        20      -rw-r--r--   1  0  0     268435456 21-Sep-2010 18:00 journal:0000
        21      -rw-r--r--   1  0  0     268435456 21-Sep-2010 18:00 journal:0001
        22      -rw-r--r--   1  0  0             0 21-Sep-2010 18:00 local_alloc:0000
        23      -rw-r--r--   1  0  0             0 21-Sep-2010 18:00 local_alloc:0001
        24      -rw-r--r--   1  0  0             0 21-Sep-2010 18:00 truncate_log:0000
        25      -rw-r--r--   1  0  0             0 21-Sep-2010 18:00 truncate_log:0001
        26      -rw-r--r--   1  0  0          8192 21-Sep-2010 18:00 aquota.user:0000
        27      -rw-r--r--   1  0  0          8192 21-Sep-2010 18:00 aquota.user:0001
        28      -rw-r--r--   1  0  0          8192 21-Sep-2010 18:00 aquota.group:0000
        29      -rw-r--r--   1  0  0          8192 21-Sep-2010 18:00 aquota.group:0001
```

The file names that end with numbers are slot specific and are referred to as node-local system files. The set of node-local files used by a node can be determined from the slot map. To list the slot map, do:

```
# debugfs.ocfs2 -R "slotmap" /dev/sde1
        Slot#   Node#
            0      32
            1      35
            2      40
            3      31
            4      34
            5      33
```

For more information, refer to the OCFS2 support guides available in the Documentation section at http://oss.oracle.com/projects/ocfs2.

13. Heartbeat, Quorum, and Fencing

Heartbeat is an essential component in any cluster. It is charged with accurately designating nodes as dead or alive. A mistake here could lead to a cluster hang or a corruption.

O2HB is the disk heartbeat component of O2CB. It periodically updates a timestamp o disk, indicating to others that this node is alive. It also reads all the timestamps to identify other live nodes. Other cluster components, like O2DLM and O2NET, use the O2HB service to get node up and down events.

The quorum is the group of nodes in a cluster that is allowed to operate on the shared storage. When there is a failure in the cluster, nodes may be split into groups that can communicate in their groups and with the shared storage but not between groups. O2QUO determines which group is allowed to continue and initiates fencing of the other group(s).

Fencing is the act of forcefully removing a node from a cluster. A node with OCFS2 mounted will fence itself when it realizes that it does not have quorum in a degraded cluster. It does this so that other nodes won't be stuck trying to access its resources.

O2CB uses a machine reset to fence. This is the quickest route for the node to rejoin the cluster.

14. Processes

[o2net]

One per node. It is a work-queue thread started when the cluster is brought on-line and stopped when it is off-lined. It handles network communication for all mounts. It gets the list of active nodes from O2HB and sets up a TCP/IP communication channel with each live node. It sends regular keep-alive packets to detect any interruption on the channels.

[user_dlm]

One per node. It is a work-queue thread started when dlmfs is loaded and stopped when it is unloaded (dlmfs is a synthetic file system that allows user space processes to access the in-kernel dlm).

[ocfs2_wq]

One per node. It is a work-queue thread started when the OCFS2 module is loaded and stopped when it is unloaded. It is assigned background file system tasks that may take cluster locks like flushing the truncate log, orphan directory recovery and local alloc recovery. For example, orphan directory recovery runs in the background so that it does not affect recovery time.

[o2hb-14C29A7392]

One per heartbeat device. It is a kernel thread started when the heartbeat region is populated in configfs and stopped when it is removed. It writes every two seconds to a block in the heartbeat region, indicating that this node is alive. It also reads the region to maintain a map of live nodes. It notifies subscribers like o2net and o2dlm of any changes in the live node map.

[ocfs2dc]

One per mount. It is a kernel thread started when a volume is mounted and stopped when it is unmounted. It downgrades locks in response to blocking ASTs (BASTs) requested by other nodes.

[kjournald2]

One per mount. It is part of JBD2, which OCFS2 uses for journaling.

[ocfs2cmt]

One per mount. It is a kernel thread started when a volume is mounted and stopped when it is unmounted. It works with kjournald2.

[ocfs2rec]

It is started whenever a node has to be recovered. This thread performs file system recovery by replaying the journal of the dead node. It is scheduled to run after dlm recovery has completed.

[dlm_thread]

One per dlm domain. It is a kernel thread started when a dlm domain is created and stopped when it is destroyed. This thread sends ASTs and blocking ASTs in response to lock level convert requests. It also frees unused lock resources.

[dlm_reco_thread]

One per dlm domain. It is a kernel thread that handles dlm recovery when another node dies. If this node is the dlm recovery master, it re-masters every lock resource owned by the dead node.

[dlm_wq]

One per dlm domain. It is a work-queue thread that o2dlm uses to queue blocking tasks.

15. Future

This release has seen a lot of development on the file system side of the product. In the next release, we expect to showcase work on the cluster stack side. This will include support for user-space cluster stacks and enhancements to the default O2CB cluster stack. But we are not done with the file system. We are working on enhancing that too by adding more features including transparent compression, transparent encryption, delayed allocation, multi-device support, discard support (thin provisioning), etc.

If you are interested in contributing, refer to the development wiki at http://oss.oracle.com/osswiki/OCFS2 for a list of projects. You can also email the development team at ocfs2-devel@oss.oracle.com.

ACKNOWLEDGEMENTS

When I was writing the guide the last time around, I was wondering whether we had just completed the most productive phase in the development of the file system. Well, if changed lines of code is any guide, then we have been almost twice as productive in this cycle than the last.

```
$ git diff --shortstat v2.6.16..v2.6.25 fs/ocfs2
87 files changed, 18968 insertions(+), 8274 deletions(-)

$ git diff --shortstat v2.6.25..v2.6.36-rc4 fs/ocfs2
104 files changed, 39737 insertions(+), 7433 deletions(-)
```

Safe to say, this is one monster of a release and I have several people to acknowledge it for.

Joel Becker. Joel has been a longtime OCFS2 developer and maintainer. He was instrumental in either designing or reviewing the designs of all features. He also reviewed almost all patches submitted upstream. In his spare time, he added features like metaecc and support for JBD2.

Mark Fasheh. Mark has been a long time OCFS2 developer and maintainer. Like Joel, he was also instrumental and designing or reviewing the designs of all features. He also added features like indexed directories and allocation reservations. He can now be found dreaming up new ways to make the allocations even more efficient.

Tao Ma. Tao developed the reflink and the discontiguous block group features, helped Tiger in adding support for extended attributes and POSIX ACLs. Apart from the external features, he also contributed in enhancing the core btree support that will allow us add more features in the future.

Jan Kara. Jan added the user and group quotas feature. He also frequently lends a hand in debugging and fixing JBD2 issues.

Tiger Yang. Tiger, along with Tao, developed the extended attributes and POSIX ACLs features.

Tristan Ye. Tristan had the challenging task of writing tests for all features added in this release. He would tag team with the developer and find new ways to break the code. He can now be found dipping his toes in the developer's pond.

Marcos Matsunaga. Marcos has been the release tester for the entire life of the file system. Running regressions in a unwieldy cluster setup is not for the faint of heart. With his effort, we have maintained quality across all our releases.

Srinivas Eeda, Wengang Wang, Goldwyn Rodrigues, Jiaju Zhang, etc. Srinivas, Wengang, et al., have contributed by fixing bugs reported by our users. They all typically go beyond just fixing the problem and look for ways to improve the file system.

Jeff Liu. Jeff started by enhancing the ethereal dissector for O2DLM. He then worked on shared-du. He is currently working with the coreutils team to make cp(1) and du(1) reflink aware.

Wim Coekaerts and Kurt Hackel for their timely advice and support.

David Teigland, **Fabio Massimo Di Nitto**, **Lars Marowsky-Bree**, **Andrew Beekhof**, **Coly Li, et al.** of the Linux Cluster community.

Christoph Hellwig, Chris Mason and other members of the Linux file system community.

The **Oracle Linux** team that builds, tests and distributes the file system.

Various testing teams in **Oracle** and **Novell** that test the file system regularly with different workloads.

EMC, **Emulex**, **HP**, **IBM**, **Intel** and **Network Appliance** for providing hardware for testing. Cluster testing is an expensive proposition. Without their help, OCFS2 would not have been possible.