OCFS2

A CLUSTER FILE SYSTEM FOR LINUX

USER'S GUIDE FOR RELEASE 1.4

Sunil Mushran

July 2008

OCFS2: A Cluster File System for Linux – User's Guide for Release 1.4

TABLE OF CONTENTS

PREFACE

The aim of this guide is to be a starting point for users looking to use the OCFS2 file system. First time 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 must refer to the chapter on New Features and review the sections on compatibilities and defaults. Oracle database users, in addition to the above, must 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 file system specific system calls and the OCFS2 tools. While this document explains the usage in an easy to understand style, it omits the actual parameter names. Users should refer to the man pages for those details.

This guide has been written specifically for the users of OCFS2 1.4, which is available only for the Enterprise distributions. Users of other distributions can also make use of this guide. The information included is up-to-date as of kernel version 2.6.25.

I INTRODUCTION

There are more than 30 disk-based file systems in the mainline Linux kernel alone. While the file systems can be categorized and sub-categorized in various ways, this document splits them into two broad ones: local and shared.

Local, as the name suggests, refers to file systems that can be accessed by a single server only. Most file systems fall under this category. Examples include EXT3/4 (general purpose workloads), XFS (enterprise workloads), JFFS2 (flash drives), ISOFS (cdroms), UFS (UNIX compatibility), NTFS/VFAT (Windows compatibility) and HFS (OS X compatibility).

Shared file systems, on the other hand, allow multiple servers to access them concurrently. This sharing allows users to share information easily, and allows administrators to consolidate storage for easier management and lower cost.

Shared File Systems

The most popular shared file system is the Network File System, NFS. It is relatively easy to setup, uses standard networking hardware and is available on almost all operating systems.

However, NFS provides weak meta-data and data cache coherencies between the client nodes. Meta-data, which in this case refers to the file name, size, modification time, etc., is allowed to go out of sync between client nodes for short periods. While NFS implementations provide options to force full synchronization, they are typically not used because of the performance penalty they impose. This is not to suggest that the file system itself does not maintain its integrity at all times. It does. The only problem is that the client nodes running the application do not have a consistent view of the file system at all times.

This lack of strict cache coherency goes unnoticed in the normal case. Simple applications do not expect different client nodes to write to the same files and directories concurrently. It causes problems when running applications that expect a consistent view of the file system on all client nodes at all times. This becomes very apparent when one attempts to scale out an application that spawns multiple processes reading and writing to the same set of files. Making these applications work on NFS requires either making the application work around the weak cache coherency, or, by disabling the various caches. The former requires application changes; the latter imposes a significant performance penalty.

Another class of shared file systems is the clustered file system. Such file systems are not always easy to setup and typically require more than just standard networking hardware. Thus, they are not as popular as NFS. However, they usually provide strict meta-data and data cache coherencies allowing users to scale out applications easily.

Such file systems provide this feature by making use of a locking infrastructure that allows it to coordinate access to resources across all the server nodes.

Clustered File Systems

Clustered file systems have been around since the 1980s. In the past few years they have generated a lot of interest. The availability of affordable enterprise grade systems with Linux/x86 allows users to replace high end SMP boxes with a cluster of quad-core machines. Clustering provides affordable scalability and availability. A node death does not bring down the entire system as it would a big SMP. Instead it allows the system to continue to operate in a degraded mode until the dead node restarts and rejoins the cluster.

Two popular designs in this class of file systems are distributed parallel (LUSTRE) and shared disk (OCFS2, GFS2).

Distributed parallel file systems have the meta-data and data are distributed across multiple servers. They scale to 1000+ clients and are very popular in high performance computing. These file systems store the file data in the local disks attached to each server node. Some of these are used to store the file system meta-data, and others the data. The data is typically stripped across multiple server nodes to provide high data throughput. While such file systems provide high scalability, the hardware requirements make it unsuitable for small to medium sized clusters.

Shared disk file systems, on the other hand, scale to 100+ nodes. As the name suggests, such file systems require a shared disk (SAN). All server nodes in the cluster must be able to perform I/O directly and concurrently to the disk. A cluster interconnect that provides a low latency network transport is used for communication in between the server nodes. In this design, the I/Os are issued directly to the SAN by each server node and thus do not suffer from the single server bottleneck inherent in NFS. The cluster interconnect is used by the server nodes to coordinate read/write access to the SAN, providing both on-disk data integrity and strict cache coherencies.

Shared disk clustered file systems come in two flavors: asymmetric and symmetric. Asymmetric refers to file systems that allow each server node to read/write data directly to the SAN, but direct all the meta-data change operations to a single node. Symmetric, on the other hand, allows each node to read and write both meta-data and data directly to the SAN.

OCFS2 is a symmetric shared disk cluster file system.

II OVERVIEW

OCFS2 is a cluster file system designed for use in a shared disk environment. It provides both *high performance* and *high availability.* It can also be used in a non-clustered environment, especially ones that are looking for the flexibility to scale out in the future.

Because it provides local file system semantics, it can also be used with applications that are not cluster-aware. They won't make use of parallel I/O, but they will be able to make use of the fail-over facilities. For example, OCFS2 is currently used to provide scalable web servers, fail-over mail servers, fail-over virtual machine image hosting, scalable file-servers, etc.

Some of the notable features of the file system are:

- **Variable Block sizes**
  Supports block sizes ranging from 512 bytes to 4KB.
- **Extent-based allocations**
  Tracks the allocated space in ranges of blocks making it especially efficient for storing large files.
- **Flexible Allocation**
  Supports sparse files and unwritten extents for higher performance and efficient storage. Existing files can have holes punched for even more efficiency.
- **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**
  Allows concurrent mounts on 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.
- **Large Inodes**
  Block-sized inodes allow it to store small files in the inode itself.
- **Comprehensive Tools Support**
  Provides a familiar EXT3-style tool-set that uses similar parameters for ease-of-use.

History

OCFS2 file system development began in 2003 as a follow up to OCFS. OCFS was targeted exclusively as a data store for Oracle's Real Application Clustered database product (RAC). The goals for the new project were to maintain the raw-like I/O throughput for the database, be POSIX compliant, and provide near local file system performance for meta-data operations. OCFS2 was intended from the start to be included in the mainline Linux kernel. At the time, there were no clustered file systems in the kernel.

OCFS2 v1.0 was released in August 2005. Shortly thereafter, it was merged into Andrew Morton's -mm Linux kernel tree.

OCFS2 was merged into the mainline Linux kernel tree in January 2006. The 2.6.16 kernel, released that March, included the file system.

OCFS2 v1.2 was released in April 2006. It targeted the Enterprise Linux distributions, *SLES9* from Novell and *RHEL4* from Red Hat. The x86, x86_64, ia64, ppc64 and s390x architectures were all supported.

As of July 2008, OCFS2 is available with all three Enterprise Linux distributions, *SLES*, *RHEL* and Oracle's *EL*.

It is also available with other distributions, notably *Ubuntu*, *openSUSE*, *Fedora Core* and *Debian*. The version of the file system on these distributions is from whichever mainline Linux kernel the distribution ships. Even though the version of the file system available for the Enterprise and other distributions is not the same, the file system maintains on-disk compatibility across all versions.

Development

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 70 developers, and thus, is on track on becoming a community project.

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 kernel trees maintained by the Stable kernel team. The stable trees are tracked by most non-Enterprise Linux kernel distributions.

The source of the file system is made available with the Linux kernel and can be downloaded from http://kernel.org/. The sources for Enterprise kernels are available at http://oss.oracle.com/projects/ocfs2/.

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

Support

Official support for the file system is typically provided by the distribution. For example, Novell supports the file system on SLES whereas Oracle supports it on EL. Oracle also extends support of the file system to RHEL for use with Oracle's database product.

For other distributions, including users of the mainline Linux kernel, Oracle also provides email support via the *ocfs2-users@oss.oracle.com* mailing list.

III NEW FEATURES

The 1.4 release provides the features that have been steadily accumulating in the mainline Linux kernel tree for over two years. The list of new features added since the OCFS2 1.2 release is as follows:

1. **Ordered Journal Mode**
   This new default journal mode (mount option *data=ordered*) forces the file system to flush file data to disk before committing the corresponding meta-data. This flushing ensures that the data written to newly allocated regions will not be lost due to a file system crash. While this feature removes the ever-so-small probability of stale or null data to appearing in a file after a crash, it does so at the expense of some performance. Users can revert to the older journal mode by mounting with *data=writeback* mount option. It should be noted that file system meta-data integrity is preserved by both journaling modes.

2. **File Attribute Support**
   Allows a user to use the chattr(1) command to set and clear EXT2-style file attributes such as the immutable bit. lsattr(1) can be used to view the current set of attributes.

3. **Performance Enhancements**
   Enhances performance by either reducing the numbers of I/Os or by doing them asynchronously.

   - **Directory Readahead**
     Directory operations asynchronously read the blocks that may get accessed in the future.

   - **File Lookup**
     Improves cold-cache stat(2) times by cutting the required amount of disk I/O in half.

   - **File Remove and Rename**
     Replaces broadcast file system messages with DLM locks for unlink(2) and rename(2) operations. This improves node scalability, as the number of messages does not grow with the number of nodes in the cluster.

4. **Splice I/O**
   Adds support for the new splice(2) system call. This allows for efficient copying between file descriptors by moving the data in kernel.

5. **Access Time Updates**
   Access times are now updated consistently and are propagated throughout the cluster. Since such updates can have a negative performance impact, the file system

allows users to tune it via the following mount options:

- **atime_quantum=<number_of_secs>**
  Defaults to 60 seconds. OCFS2 will not update atime unless this number of seconds has passed since the last update. Set to zero to always update it.

- **noatime**
  This standard mount option turns off atime updates completely.

- **relatime**
  This is another standard mount option (added in Linux v2.6.20) supported by OCFS2. 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.

Additionally, all time updates in the file system have **nanosecond resolution**.

6. **Flexible Allocation**
   The file system now supports some advanced features that are intended to allow users more control over file data allocation. These features entail an on-disk change.

   - **Sparse File Support**
     It adds the ability to support holes in files. This allows the ftruncate(2) system call to efficiently extend files. The file system can postpone allocating space until the user actually writes to those clusters.

   - **Unwritten Extents**
     It adds the ability for an application to request a range of clusters to be pre-allocated, but not initialized, within a file. Pre-allocation allows the file system to optimize the data layout with fewer, larger extents. It also provides a performance boost, delaying initialization until the user writes to the clusters. Users can access these features via an ioctl(2), or via fallocate(2) on current kernels.

   - **Punching Holes**
     It adds the ability for an application to remove arbitrary allocated regions within a file. Creating holes, essentially. This could be more efficient if a user can avoid zeroing the data. Users can access these features via an ioctl(2), or via fallocate(2) on later kernels.

7. **Shared Writeable mmap(2)**
   Shared writeable memory mappings are fully supported now on OCFS2.

8. **Inline Data**
   This feature makes use of OCFS2's large inodes by storing the data of small files and directories in the inode block itself. This saves space and can have a positive impact on cold-cache directory and file operations. Data is transparently moved out to an extent when it no longer fits inside the inode block. This feature entails an on-disk change.

9. **Online File system Resize**
   Users can now grow the file system without having to unmount it. This feature requires a compatible clustered logical volume manager. Compatible volumes managers will be announced when support is available.

10. **Clustered flock(2)**
   The flock(2) system call is now cluster-aware. File locks taken on one node from user-space will interact with those taken on other nodes. All flock(2) options are supported, including the kernel's ability to cancel a lock request when an appropriate kill signal is received. (*Note: Support for clustered POSIX file locks, also known as lockf(3) or fcntl(2), has not yet been added. We hope to have that available in the near term.*)

File system Compatibility

OCFS2 1.4 is fully compatible with the 1.2 on-disk format. The newer version can mount volumes from the older version as-is. However, the network protocol is not compatible between the two versions. Concurrent mounts from nodes running the different versions are not supported.

The development team ensures that all versions of OCFS2 are backward compatible on-disk. Users can upgrade OCFS2 to a newer release knowing it can mount existing volumes. The same could not be said of the network protocol in the past. We have recently improved the software infrastructure to allow us to maintain network compatibility. This gives us the flexibility to make changes in the protocol. In the new scheme, the cluster keeps track of the active protocol version. Newer nodes that understand the active version may join at the older level. OCFS2 1.4 is not network compatible with OCFS2 1.2, but this change means that most future releases will be compatible with OCFS2 1.4.

Though OCFS2 maintains on-disk compatibility, new features are not enabled by default. Features requiring on-disk changes need to be enabled explicitly. tunefs.ocfs2(8) allows users to enable the features on existing volumes. For new volumes, mkfs.ocfs2(8) can enable them.

Once new features are enabled, the volume cannot be mounted by older versions that do not understand the feature. If you want to mount the volume on an older version, tunefs.ocfs2(8) can also toggle the features off.

For more on this, refer to the Format and Tuning sections and the mkfs.ocfs2(8) and tunefs.ocfs2(8) man pages.

Tools Compatibility

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

Distribution Compatibility

OCFS2 1.4 is a back-port of the file system included in the mainline Linux kernel version 2.6.25. It has been written to work only with (RH)EL5 Update 2 and SLES10 SP2. It will continue to work with the newer (RH)EL5 and SLES10 kernel updates but will not work on older releases.

New File System Defaults

The support for sparse files and unwritten extents is activated by default when using mkfs.ocfs2(8) v1.4. Users wishing to retain full compatibility with older file system software must specify *--fs-feature-level=max-compat* to mkfs.ocfs2(8).

The other change in the defaults concerns the journaling mode. While OCFS2 1.2 supported the writeback data journaling mode, OCFS2 1.4 adds support for the ordered data journaling mode and makes it the default. Users wishing to keep using the writeback mode should mount the volume with the *data=writeback* option.

In ordered mode, the file system flushes the file data to disk before committing metadata changes. In writeback mode, no such write ordering is preserved.

The writeback mode still guarantees internal file system integrity, and can have better overall throughput than ordered mode. However, stale data can appear in files after a crash and a subsequent journal recovery.

IV 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 and check the file system.

Software Packaging

For almost all distributions, the kernel file system component is bundled with the kernel package. This is ideal, as upgrading the kernel automatically upgrades the file system. Novell's SLES distribution uses this approach, as do most non-enterprise Linux distributions. For Red Hat's RHEL and Oracle's EL distributions, the kernel component is available as a separate package. This package needs to be installed when upgrading the kernel.

In the distributions in which the kernel component is provided separately, care should be taken to install the appropriate package. This is not to say that installing an incorrect package will cause harm. It won't. But it will not work.

An example of OCFS2's kernel component package is:

ocfs2-2.6.18-92.el5PAE-1.4.1-1.el5.i686.rpm

The package name can be broken down as follows:

ocfs2-2.6.18-92.el5PAE	Package name
1.4.1-1	Package version
el5	Distribution
i686	Architecture

The package name includes the kernel version. To learn the appropriate package name for your running kernel do:

```
$ echo ocfs2-`uname –r`
```

The user-space component includes two packages: ocfs2-tools (command line interface) and ocfs2console (graphical user interface). These packages are specific to a distribution and architecture only and have no kernel dependency. For example, ocfs2-tools-1.4.1-1.el5.i386.rpm is for the EL5/x86 platform regardless of the exact kernel version in use.

A DOWNLOAD AND INSTALL

The download and install procedure are specific to each distribution. The procedures for some of the more popular distributions are listed below.

Oracle's Enterprise Linux
Both the file system and the tools need to be installed.

```
$ up2date --install ocfs2-tools ocfs2console
$ up2date --install ocfs2-`uname -r`
```

Novell's SUSE Linux Enterprise Server
Only the tools need to be installed; the file system is bundled with the kernel.

```
$ zypper install ocfs2-tools ocfs2console
```

Canonical's Ubuntu and other Debian-based distributions
Only the tools need to be installed; the file system is bundled with the kernel.

```
$ apt-get install ocfs2-tools ocfs2console
```

Red Hat Enterprise Linux
Both the file system and tools packages need to be downloaded from http://oss.oracle.com/. Once downloaded, the packages can be installed using the rpm utility.

```
$ rpm -Uvh ocfs2-tools-1.4.1-1.el5.i386.rpm
$ rpm -Uvh ocfs2console-1.4.1-1.el5.i386.rpm
$ rpm -Uvh ocfs2-2.6.18-92.el5PAE-1.4.1-1.el5.i686.rpm
```

B 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. Everyone else will need to configure the O2CB cluster.

The O2CB cluster stack requires cluster layout and cluster timeout configurations.

O2CB Cluster Layout Configuration

The cluster layout is specified in /etc/ocfs2/cluster.conf. It is easy to populate and propagate this configuration file using ocfs2console(8), but one can also do it manually if care is taken to format the file correctly.

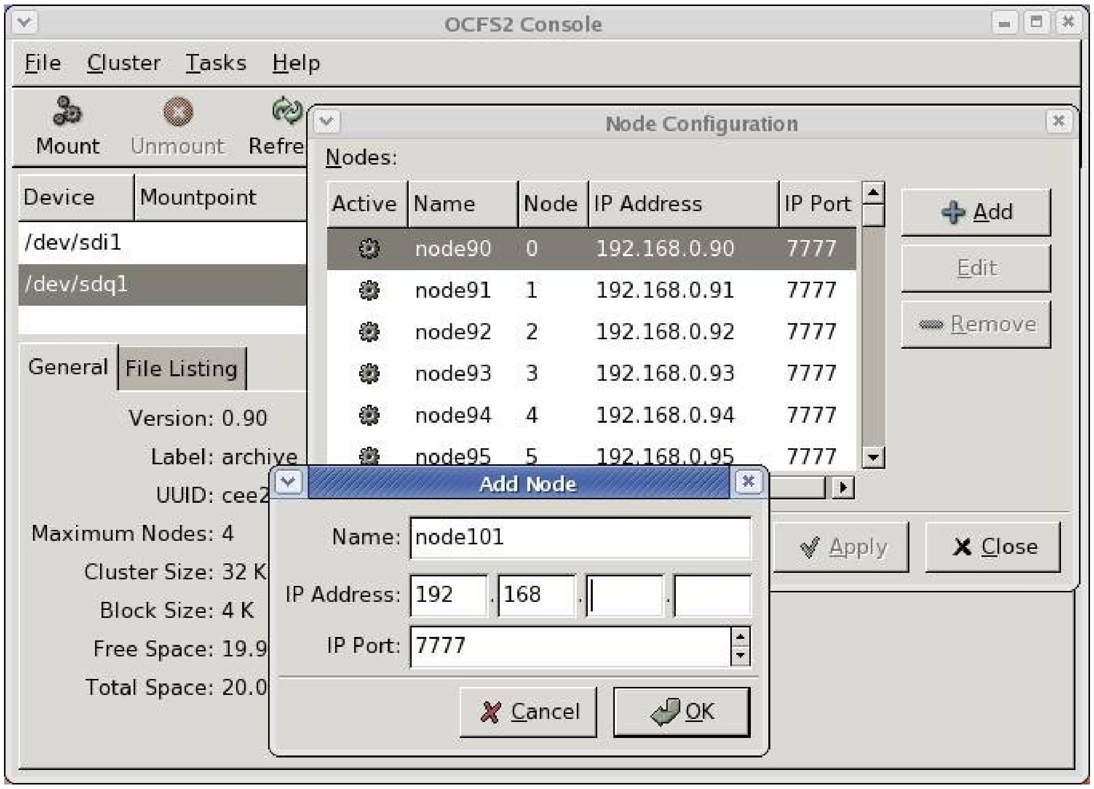
While the console utility is intuitive to use, there are few points to keep in mind.

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 (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 everywhere and manually editing the configuration file on all nodes before restarting it. Always ensure that the cluster.conf is the same on all nodes in the cluster.



Device	Mountpoint
/dev/sdi1	
/dev/sdq1	

Version: 0.90
Label: archive
UUID: cee2
Maximum Nodes: 4
Cluster Size: 32 K
Block Size: 4 K
Free Space: 19.9
Total Space: 20.0

Active	Name	Node	IP Address	IP Port
⚙	node90	0	192.168.0.90	7777
⚙	node91	1	192.168.0.91	7777
⚙	node92	2	192.168.0.92	7777
⚙	node93	3	192.168.0.93	7777
⚙	node94	4	192.168.0.94	7777
⚙	node95	5	192.168.0.95	7777

Name: node101
IP Address: 192 . 168 . .
IP Port: 7777

Users who have configured the cluster with ocfs2console(8) can skip to the timeout configuration.

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. Take care to avoid any stray white space.

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.

Using the o2cb init script, one can configure the timeouts as follows:

```
$ 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) [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
Cluster webcluster already online
```

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 are causing 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 timeout is 60 secs (O2CB_HEARTBEAT_THRESHOLD = 31).

Network Idle Timeout
The network idle timeout specifies the time in milliseconds before a network connection is considered dead. It defaults to 30000 ms.

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
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 two RHEL components, SELINUX and iptables, to be disabled or modified. We will look into supporting SELINUX once extended attribute support has been added to the file system. The firewall, if enabled, must allow traffic on the private network interface.

C FORMAT

This section assumes that the user has enabled the O2CB cluster stack, as it could be required to re-format an existing OCFS2 volume. The cluster stack, however, need not be enabled if the user intends to use the volume only as a local file system.

Like any other file system, the volume needs to be formatted before use. As formatting is a 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.

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 existing 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 only work with partitioned volumes. For more on partitioning, check the man pages of fdisk(8) or parted(8).

Users can format using ocfs2console(8) or the command line tool, mkfs.ocfs2(8). This document explains the usage and the various options to mkfs.ocfs2(8) but expects the user to refer to the man page for exact parameter names.

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, large sized files) and mail-server (lots-of, small sized files). Other than the block and cluster sizes, all other options can be modified by tunefs.ocfs2(8).

The main options for mkfs.ocfs2(8) are:

Block Size

The block size is the unit of space allocated for meta-data by the file system. OCFS2 supports block sizes of 512, 1K, 2K and 4K bytes. The block size cannot be changed after the format. For almost all uses, a 4K-block size is recommended. A 512 byte block size is never recommended.

Cluster Size

The cluster size is the unit of space allocated for file data. All data allocation is in multiples of the cluster size. OCFS2 supports cluster sizes of 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K and 1M bytes. For almost all uses, a 4K size is recommended.

However, volumes storing database files should not use a value smaller than the database block size and are free to use a larger value.

Node Slots

A node slot refers to a set of system files, like a journal, that are used exclusively by one node. This limits the number of nodes that can concurrently mount a volume to the number of node slots it has. This number can later be increased or decreased using tunefs.ocfs2(8). Different volumes can have a different number of node slots.

Journal Options

OCFS2 uses the write-ahead journal, JBD, with a user-configurable size. If left unspecified, the tool determines the appropriate value based on the specified file system type. The defaults are 64MB for *datafiles* and 256MB for *mail*.

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.

Mount Type

Valid types are *cluster* and *local*, with the former also being the default. Specify *local* if you intend to use the file system on one node only.

File System Type

Valid types are *mail* and *datafiles*. *mail* refers to its use as a mail server store that has the usage characteristic of lots of small files, requiring lots of meta-data changes that in turn are benefited by using a larger journal. *datafiles*, on the other hand, suggests fewer large files, requiring fewer meta-data changes, thus not benefiting from a large journal.

File System Features

Allows users to enable or disable certain file system features, including *sparse files*, *unwritten extents* and *back-up super blocks*. Refer to the man pages for the current list of supported features.

File System 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 features that the file system software currently supports. *default* currently enables support for sparse files and unwritten extents.

Examples

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

```
$ mkfs.ocfs2 -L "myvolume" /dev/sda1
```

```
mkfs.ocfs2 1.4.1
Filesystem label=myvolume
Block size=4096 (bits=12)
Cluster size=4096 (bits=12)
Volume size=53687074816 (13107196 clusters) (13107196 blocks)
407 cluster groups (tail covers 11260 clusters, rest cover 32256
clusters)
Journal size=268435456
Initial number of node slots: 4
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 3 block(s)
Formatting Journals: done
Writing lost+found: done
mkfs.ocfs2 successful
```

The values chosen include 4K block and cluster sizes, 4 node slots with 256 MB journal each. The list of file system features enabled can be viewed using tunefs.ocfs2(8).

```
$ tunefs.ocfs2 -q -Q "All Features: %M %H %O\n" /dev/sda1
All Features: BackupSuper SparseAlloc UnwrittenExtents
```

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

```
$ mkfs.ocfs2 -T datafiles -L "mydatavol" /dev/sda1
mkfs.ocfs2 1.4.1
Overwriting existing ocfs2 partition.
Proceed (y/N): y
Filesystem Type of datafiles
Filesystem label=mydatavol
Block size=4096 (bits=12)
Cluster size=131072 (bits=17)
Volume size=53686960128 (409599 clusters) (13107168 blocks)
13 cluster groups (tail covers 22527 clusters, rest cover 32256
clusters)
Journal size=33554432
Initial number of node slots: 4
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 3 block(s)
Formatting Journals: done
Writing lost+found: done
mkfs.ocfs2 successful
```

mkfs.ocfs2(8) selected a larger cluster size, and the journal size was limited to 32M.

To format the volume with custom values (e.g. 4K block and cluster sizes, 8 node-slots with 128MB journals each and maximum compatibility with older file system software) do:

```
# mkfs.ocfs2 -b 4K -C 4K -N 8 -L "ocfs2vol" -J size=128M  \
 --fs-feature-level=max-compat /dev/sda1
mkfs.ocfs2 1.4.1
Overwriting existing ocfs2 partition.
Proceed (y/N): y
Filesystem label=ocfs2vol
Block size=4096 (bits=12)
Cluster size=4096 (bits=12)
Volume size=53687074816 (13107196 clusters) (13107196 blocks)
407 cluster groups (tail covers 11260 clusters, rest cover 32256
clusters)
Journal size=134217728
Initial number of 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
Writing lost+found: done
mkfs.ocfs2 successful
```

D 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 configured and started.

Commands to mount and umount OCFS2 volumes is 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 remaining nodes.

If the mount fails, detailed errors can be found via dmesg(8). These might include incorrect cluster configuration (say, 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 that runs after the *o2cb* init service has started the cluster. The *ocfs2* init service mounts all OCFS2 volumes listed in /etc/fstab.

Mount Options

OCFS2 supports many mount options that are supported by other Linux file systems. The list of supported mount options is as follows:

_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 and 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's 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 some data-loss.

data=ordered / data=writeback
This specifies the handling of data during metadata journaling.

ordered
This is the default mode. All data is forced directly out to the main file system prior to its metadata being committed to the journal.

writeback
Data ordering is not preserved - data may be written into the main file system 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.

datavolume
Use this mount option to mount volumes storing Oracle data files, control files, redo logs, archive logs, voting disk, cluster registry, etc. (This mount option is only available with OCFS2 1.2 and OCFS2 1.4 for Enterprise distributions EL, RHEL and SLES.)

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, *intr* allows signals to interrupt certain cluster operations. *nointr* disables signals during cluster operations.

localflocks
This disables cluster-aware flock(2).

noatime
This standard mount option turns off atime updates completely.

relatime
This is only available with Linux kernel v2.6.20 and later. 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
```

V ADMINISTRATION

The previous section concentrated on getting started. It covered configuring and starting the cluster and formatting and mounting the volume.

This section deals with administrative operations that can be performed on the volume. This includes adding slots, resizing, changing the label, checking the file system, etc.

A TUNING

tunefs.ocfs2(8) allows changing most of the file system's parameters. In fact, other than the block and cluster sizes, all other parameters can be modified. As the tool modifies an existing OCFS2 volume, the cluster must be started to see whether the volume is in use across the cluster. If the volume is not in use, all operations can be performed. If it is in use, only online operations may be performed.

tunefs.ocfs2(8) can perform the following tasks:

- **FS Features Toggle**
  It allows toggling file system features on and off. Enabling new file system features can make the volume un-mountable with older versions of the software. Having the ability to toggle off features is useful during the transition period. However, care should be taken; this operation is not guaranteed to succeed. For example, disabling sparse file support involves filling all the holes; it will only work if the file system has enough free space to fill them.

- **Journal Resize**
  It allows the journal to be grown or truncated. While this operation is not typically recommended, it is provided in case the user wishes to tweak the size of the journal for performance or other reasons.

- **Volume Label update**
  It allows the volume label to be changed. Having the ability to change the label is handy, as labels are useful for identifying a volume across a cluster.

- **Volume Mount Type Toggle**
  It allows toggling the mount type between *cluster* and *local*. When set to *local*, the mount utility can mount the volume without the cluster stack.

- **Node Slots update**
  It allows increasing and decreasing 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 and thus recover space used by journals. This is currently an offline operation. We intend to make add slots an online operation soon.

- **Volume Resize**
  It allows growing 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, if not earlier. The tool does not allow shrinking the volume.

- **UUID Reset**
  It allows changing the UUID of the volume. This is useful when using the volume on EMC and NetApp disk arrays that allow volume cloning. As OCFS2 uses the UUID to uniquely identify a volume, this option needs to be performed on the cloned volume to allow it to distinguish itself from the original.

B FILE SYSTEM CHECK

fsck.ocfs2(8) is the file system check tool. It detects and fixes on-disk errors. Like tunefs.ocfs2(8), it expects the cluster to be online. It needs to ensure the volume is not in use across the cluster.

When run without any options, fsck.ocfs2(8) only replays the journals. For a full scan, specify –f to force-check the volume.

```
$ fsck.ocfs2 -f /dev/sdf1
Checking OCFS2 filesystem in /dev/sdf1:
  label:              apache-data
  uuid:               94 44 83 17 2d 30 41 5a 9b 97 d4 c7 f7 c9 f4 3e
  number of blocks:   13107196
  bytes per block:    4096
  number of clusters: 13107196
  bytes per cluster:  4096
  max slots:          4

/dev/sdf1 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 fsck.ocfs2.checks(8) man page has a listing of all 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 is corrupted, the backups are available to scan and fix the corruption.

mkfs.ocfs2(8) enables this feature by default. Users wishing explicitly 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 -qQ "%M\n" /dev/sda1
BackupSuper
```

In OCFS2, the super block is on the third block. The backups are located on 1GB, 4GB, 16GB, 64GB, 256GB and 1TB 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 1GB.

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/sdf1
[RECOVER_BACKUP_SUPERBLOCK] Recover superblock information from backup
block#1048576? <n> y
Checking OCFS2 filesystem in /dev/sdf1:
  label:              apache-data
  uuid:               94 44 83 17 2d 30 41 5a 9b 97 d4 c7 f7 c9 f4 3e
  number of blocks:   13107196
  bytes per block:    4096
  number of clusters: 13107196
  bytes per cluster:  4096
  max slots:          4

/dev/sdf1 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.
```

C OTHER TOOLS

The previous sections have covered tools to mount, format, tune and check an OCFS2 volume. This section briefly reviews the remaining tools. The complete usage description 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     UUID                                  Label
/dev/sda1       ocfs2  84044d1e-fffd-4330-92c6-3486cab85c0f  myvol
/dev/sdc1       ocfs2  03b685de-2df6-42ce-9385-665ad4b1ba62  cman-test
/dev/sdd1       ocfs2  4d6ef662-f9c6-4db8-ab12-8e87aedec207  racdb
/dev/sdf1       ocfs2  94448317-2d30-415a-9b97-d4c7f7c9f43e  apache-data
```

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/sdd1       ocfs2  Not mounted
/dev/sdf1       ocfs2  Not mounted
```

o2cb_ctl(8)

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

To add node *node4* as node number *4* with IP address *192.168.0.104* using port *777* to 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's documentation section.

o2image(8)

This is a new tool modeled after ext3's e2image. It allows users to extract the meta-data from a volume and save it to another file. This file can then be read using debugfs.ocfs2(8). This will be useful for debugging on-disk issues that are not being fixed by fsck.ocfs2(8). Moreover, the OCFS2 developers hope to use this information to analyze the allocation layouts of user volumes in order to better understand typical usage.

It should be noted that o2image(8) does not backup any user data. It only backs up meta-data, such as inodes and directory listings.

VI ORACLE RDBMS

One of the earliest uses of the OCFS2 file system was with Oracle's Real Application Cluster database product (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 details some additional configuration and issues one needs to be aware of when using OCFS2 with the Oracle RDBMS.

Mount Options

Two mount options have been added specifically for the Oracle database. One disables signals (*nointr*); the other forces the database to use direct I/O (*datavolume*).

It is necessary to mount volumes hosting the datafiles, control files, redo logs, etc with the *nointr* mount option. This prevents short I/Os, which could happen if a signal were to interrupt an I/O in progress. This is similar to the mount option in NFS.

The *datavolume* mount option is more legacy than anything else. It directs the database to perform direct I/O to the datafiles, control files, redo logs, etc. on 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 RAC voting disk file and the cluster registry (OCR) still requires this option.

It should be noted that the *datavolume* mount option is only available in the OCFS2 1.2 and 1.4 releases for Enterprise distributions. It is not available in other distributions shipping the mainline kernel.

These mount options are **not** required on a volume for the Oracle home or any other use. Users should not use the same volume for the Oracle home and datafile storage.

Timeouts

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 (OCR) but not the CRS/CSS binaries, 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 after the cluster is restarted.

Cluster Sizes

When specifying *datafiles* as the file system type, mkfs.ocfs2(8) sets the cluster size to 128K. However, there is no one correct value and users are free to use a different value. The only 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. The only point to note is that as the cluster size specifies the smallest file data allocation. Using a large value could lead to space wastage if the volume is being used to store many small files as well.

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 with Oracle products is available by clicking on the *Certify & Availability* option on *metalink*, http://metalink.oracle.com/.

VII NOTES

a) 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, in OCFS2 we have taken the effort to ensure it works in a mixed architecture environment.

If one uses OCFS2 in a mixed architecture environment, still 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. This is because the O2CB cluster stack favors lower node numbers in all of its tie-breaking logic.

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

b) File Deletion

In Linux, rm(1) removes the directory entry. It does not necessarily delete the corresponding inode too. 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.

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, the 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.

c) 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 in comparison to any 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.

d) 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 all 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
```

e) 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 lock resources. Each node maintains enough information for all the lock resources it is interested in. If a node dies, the lock state information maintained by the dead node can be reconstructed by the remaining nodes. 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 available at the following link. http://opendlm.sourceforge.net/cvsmirror/opendlm/docs/dlmbook_final.pdf

f) DLM Debugging

One new feature in the release that has gone unmentioned is the improvement in the debugging infrastructure, especially in O2DLM. As per kernel convention, all debugging related information is made accessible via the debugfs file system.

Among the information provided is the dlm state. In the sample below, we can see a nine-node cluster that has just lost three nodes: 12, 32 and 35. Node 7 is the recovery master, is currently recovering node 12, and has received the lock states of the dead node from the 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
Mastered Resources Total: 48850  Locally: 48844  Remotely: 6  Unknown: 0
Lists: Dirty=Empty  Purge=Empty PendingASTs=Empty PendingBASTs=Empty  Master=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 previous version of the file system allowed users to dump the file system lock states (fs_locks). This one also allows users to dump the dlm lock states (dlm_locks). The difference is that the file system has no knowledge of the other nodes in the cluster. In the sample below, we see that the lock resource is owned (mastered) by node 25 and that node 26 holds 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
```

Another enhancement in this release is to the debugfs command, *fs_locks*. It now supports a *–B* option to limit the output to only the *busy* locks. This is useful, as a typical file system can easily have over 100,000 lock resources. Being able to filter out the unnecessary information makes it much easier to isolate the problem.

```
$ 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.

g) 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. Users must mount the NFS volumes on the clients 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.

h) Limits

OCFS2 1.4 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. There are three limits imposed by the current filesystem:

- An inode can have at most 32000 hard links. This means that a directory is limited to 32000 sub-directories. This limit may be raised when indexed directories are added to the file system.

- OCFS2 can address at most 2^{32} (approximately four billion) clusters. A file system with 4K clusters can go up to 16TB, while a file system with 1M clusters can reach 4PB.

- OCFS2 uses JBD for journaling. JBD can address a maximum of 2^{32} (approximately four billion) blocks. This limits the current maximum file system size to 16TB. This limit will increase when support for JBD2 is added to OCFS2.

i) 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/sdd1
        514     drwxr-xr-x   4  0  0      4096  8-Jul-2008 10:18 .
        514     drwxr-xr-x   4  0  0      4096  8-Jul-2008 10:18 ..
        515     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 bad_blocks
        516     -rw-r--r--   1  0  0   1048576  8-Jul-2008 10:18 global_inode_alloc
        517     -rw-r--r--   1  0  0   1048576  8-Jul-2008 10:18 slot_map
        518     -rw-r--r--   1  0  0   1048576  8-Jul-2008 10:18 heartbeat
        519     -rw-r--r--   1  0  0  81788928  8-Jul-2008 10:18 global_bitmap
        520     drwxr-xr-x   2  0  0      4096  8-Jul-2008 10:18 orphan_dir:0000
        521     drwxr-xr-x   2  0  0      4096  8-Jul-2008 10:18 orphan_dir:0001
        522     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 extent_alloc:0000
        523     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 extent_alloc:0001
        524     -rw-r--r--   1  0  0   4194304  8-Jul-2008 10:18 inode_alloc:0000
        525     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 inode_alloc:0001
        526     -rw-r--r--   1  0  0   4194304  8-Jul-2008 10:18 journal:0000
        527     -rw-r--r--   1  0  0   4194304  8-Jul-2008 10:18 journal:0001
        528     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 local_alloc:0000
        529     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 local_alloc:0001
        530     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 truncate_log:0000
        531     -rw-r--r--   1  0  0         0  8-Jul-2008 10:18 truncate_log: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/sdd1
        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.

j) 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.

k) 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 threads. 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.

[kjournald]
One per mount. It is part of JBD, 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 kjournald.

[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.

I) Future

As clustering has become more popular, so have the demands for a common cluster stack. One of the developments projects underway is to allow OCFS2 to work with user-space cluster stacks. This is a work in progress, and we hope to have a solution by (RH)EL6 and SLES11.

Work is underway on the core filesystem as well. We are looking to add support for extended attributes, POSIX locking, on-line node-slot addition, and JBD2.

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

This release marks an important milestone in the development of OCFS2, which began in late 2001. The author wishes to take this opportunity to acknowledge the people whose contributions made this milestone possible.

Kurt Hackel started the OCFS project with the author and went on to design and write large parts of OCFS2, including the distributed lock manager.

Mark Fasheh, who joined the two us a little after the first release of OCFS, worked on designing and rewriting the entire file system. He went on to add advanced features like sparse files, unwritten extents, inline data, shared writeable mmap, and dlmfs and is currently an official maintainer of the file system.

Joel Becker, who, having experienced different cluster stacks, designed O2CB to be the simplest one yet. He is currently adding support for user-space cluster stacks while keeping the original interface intact. He also wrote large chunks of the user-space libraries and configfs. He is also an official maintainer of the file system.

Zach Brown, having worked on Lustre, wrote o2net and other kernel pieces of the O2CB cluster stack, also wrote fsck.ocfs2, and can now be found working on CRFS, a cache-coherent network file system.

Tiger Yang started by adding support for splice I/Os. He can now be found adding support for extended attributes.

Tao Ma started by identifying and fixing bugs in fsck.ocfs2, went on to write online resize, and can be currently found adding support for the extended attributes.

Marcos Matsunaga, the one-man testing machine, tests all production releases, mainline kernels, and development patches in an unwieldy cluster setup.

Wim Coekaerts drove the mainline submission process.

Manish Singh had the undesirable task of debugging the supposed compiler issues we encountered during the development. He also wrote the console and lent a hand in the build infrastructure.

Rusty Lynch, Sonic Zhang, John L. Villalovos and Xiaofeng Ling from Intel lent a hand during the dog days of development in 2004.

Christoph Hellwig not only cleaned up a lot of the source code, he helped get the file system accepted in the mainline kernel.

Jeff Mahoney, Jan Kara, Lars Marowsky-Bree, Andrew Beekhof, Coly Li and Zhen Wei from SUSE Labs continue to ensure the file system works on the SLES distribution.

Fabio Massimo Di Nitto helped get the file system included in the Ubuntu distribution. He also maintains the debian specific parts of our build infrastructure.

David Teigland and the members of Red Hat's Linux Cluster group who are helping us integrate OCFS2 with the new CMAN.

Various testing teams in Oracle, who did not give up on the file system even after suffering numerous oopses and hangs in the early days.

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