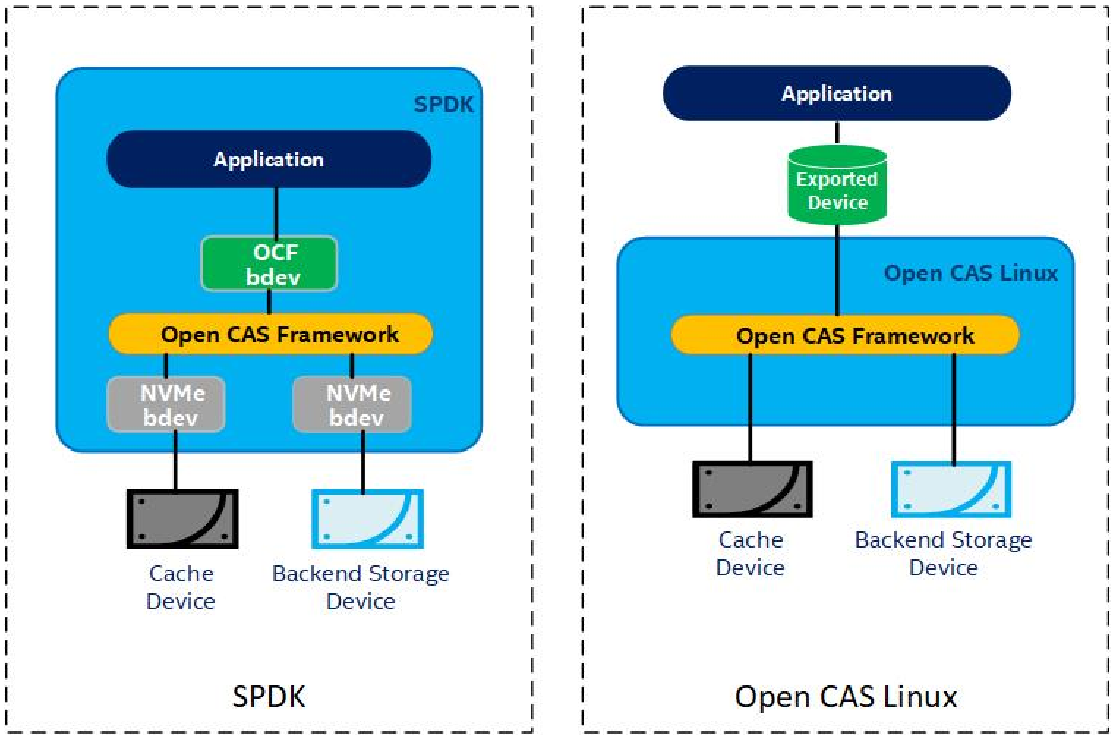
Open Cache Acceleration Software

Introduction

Open Cache Acceleration Software (Open CAS) is an open source project encompassing block caching software libraries, adapters, tools and more. The main goal of this cache acceleration software is to accelerate a backend block device(s) by utilizing a higher performance device(s).

At the core of Open CAS is the Open CAS Framework (OCF). To build upon OCF and in order to provide complete caching solutions, Open CAS also provides adapter implementations for Linux operating systems and for SPDK applications. See the sections Open CAS Linux and SPDK Block Device below for more information.

The figure below shows the software stack of complete caching solutions using SPDK and Open CAS Linux.



The main sub-projects of Open CAS are:

- Open CAS Framework (OCF)
- Open CAS Linux
- SPDK OCF Block Device

Open CAS Framework (OCF)

The Open CAS Framework (OCF) is a high performance block storage caching meta-library written in C.
OCF is the foundation which Open CAS Linux and SPDK depend on to provide complete caching solutions.

Visit the What is OCF and OCF Quick Setup pages for more information.

Open CAS Linux

Open CAS Linux provides kernel adapters to OCF in order to implement a high performance, low latency complete caching solution for Linux operating systems.
The picture below depicts in green the integration of OCF and the implemented kernel adapters provided with Open CAS Linux.

What is OCF?

What is OCF?

Open CAS Framwework (OCF) is high performance block storage caching meta-library written in C. It’s entirely platform and system independent, accessing system API through user provided environment wrappers layer. OCF tightly integrates with the rest of software stack, providing flawless, high performance, low latency caching utility.

What’s in it for me?

For short - performance. OCF enables easy block cache deployment for any performance sensitive application that works with huge data sets stored on persistent memory. It was primarily designed to cache data from HDD drives on SSDs, but it also can be used for caching data from

QLC SSD on TLC SSD, Optane drives, RAM memory, or any combination of above including all kinds of multilevel configurations.

Who already uses OCF?

Open CAS Framework is heart of two established caching products:

- Intel ® Cache Acceleration Software for Linux
- Intel ® Cache Acceleration Software for QEMU

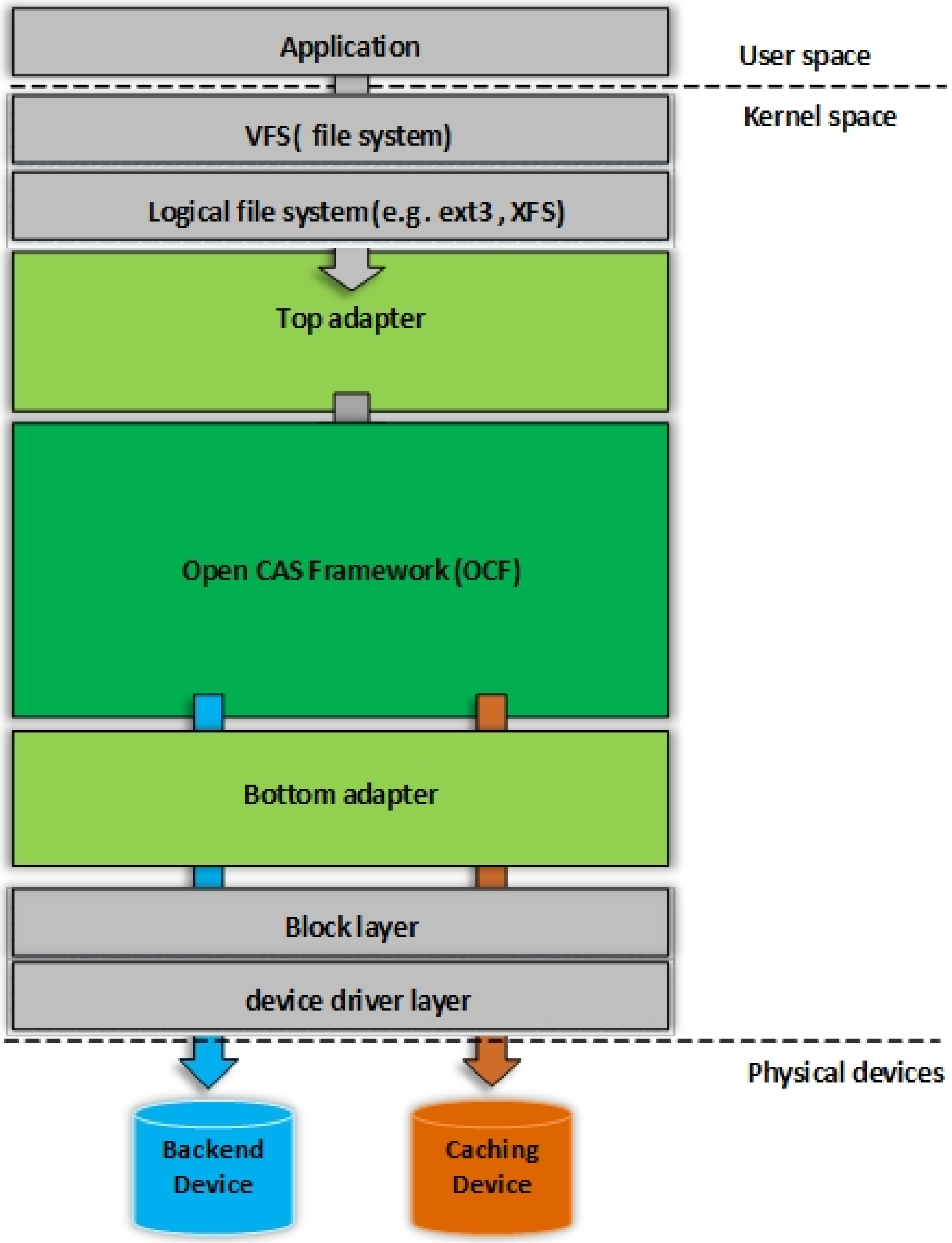
We are also default caching solution for Storage Performance Development Kit (starting from version 19.01).

Who contributes to OCF?

Our open source community is still forming and we are very positive about that. OCF is mostly developed and maintained by Intel. We are the team who developed OCF (as a part of Cache Acceleration Software) for many years. Now we are very extited about undergoing transition to truely open source project. We are open for contributions of any kind. If you feel like helping us make OCF better, take a look at Contributing page.

How do I get it?

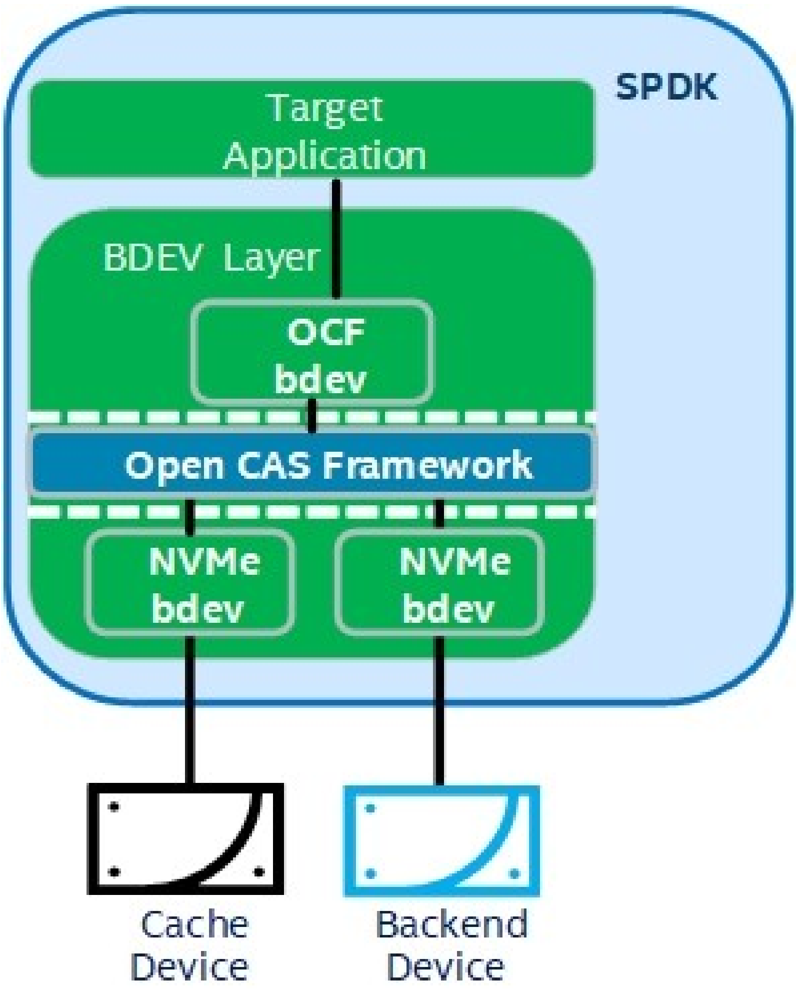
We are fully open source. Visit our GitHub page.



Visit the Open CAS Linux Quick Setup and Open CAS Linux Admin Guide pages for more information.

Storage Performance Development Kit OCF Block Device

The Storage Performance Development Kit (SPDK) is a set of tools and libraries for building high performance applications. Open CAS enhances SPDK by providing an OCF SPDK block device adapter to build high performance caching-aware applications. The SPDK OCF block device is independent from Open CAS Linux as it implements a different type of adapter while still utilizing OCF. The picture below depicts the integration of OCF and SPDK.



Visit the Storage Performance Development Kit and SPDK Quick Setup pages for more information.

What is OCF?

Open CAS Framework (OCF) is high performance block storage caching meta-library written in C. It’s entirely platform and system independent, accessing system API through user provided environment wrappers layer. OCF tightly integrates with the rest of software stack, providing flawless, high performance, low latency caching utility.

What’s in it for me?

For short - performance. OCF enables easy block cache deployment for any performance sensitive application that works with huge data sets stored on persistent memory. It was primarily designed to cache data from HDD drives on SSDs, but it also can be used for caching data from QLC SSD on TLC SSD, Optane drives, RAM memory, or any combination of above including all kinds of multilevel configurations.

Who already uses OCF?

Open CAS Framework is heart of two established caching products:

- Intel ® Cache Acceleration Software for Linux
- Intel ® Cache Acceleration Software for QEMU

We are also default caching solution for Storage Performance Development Kit (starting from version 19.01).

Who contributes to OCF?

Our open source community is still forming and we are very positive about that. OCF is mostly developed and maintained by Intel. We are the team who developed OCF (as a part of Cache Acceleration Software) for many years. Now we are very extited about undergoing transition to truely open source project. We are open for contributions of any kind. If you feel like helping us make OCF better, take a look at Contributing page.

How do I get it?

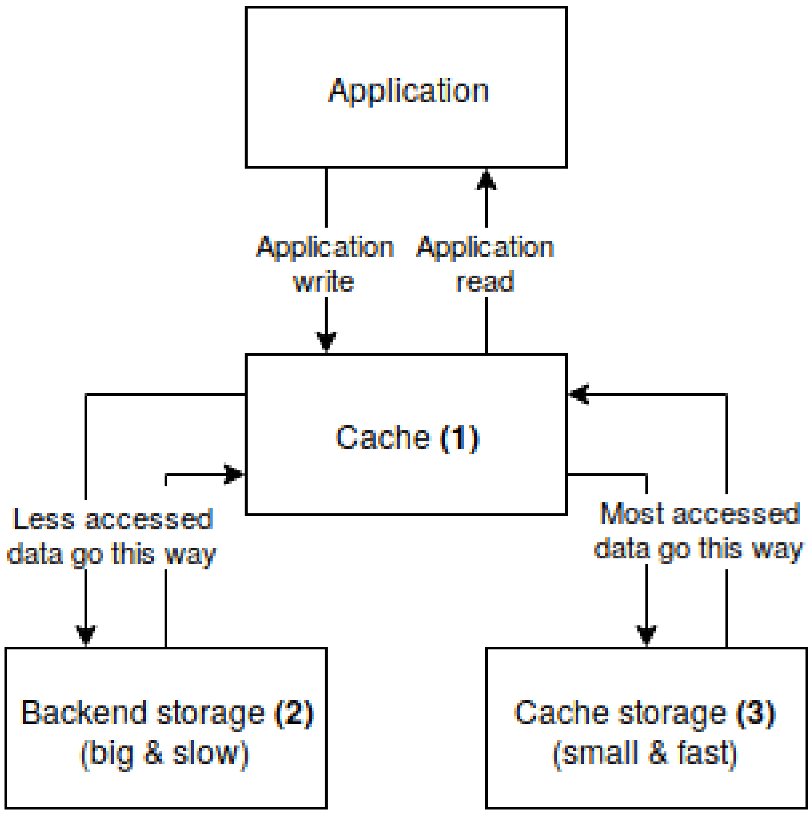
We are fully open source. Visit our GitHub page.

What is cache?

The following section explains the idea of caching and briefly describes role of the **cache** abstraction in OCF. You can learn more about how the **cache** works reading Cache operations page. If you want to learn more about the **cache** configuration parameters, like **cache mode** and **cache line size**, read the Cache configuration page.

Caching concept

In general a **cache (1)** is a component that mediates data exchange between an application and a **backend storage (2)**, selectively storing most accessed data on a relatively smaller and faster **cache storage (3)**. Every time the cached data is accessed by the application, an I/O operation is performed on the cache storage, decreasing data access time and improving performance of the whole system.


Application
Application write
Application read
Cache (1)
Less accessed data go this way
Most accessed data go this way
Backend storage (2) (big & slow)
Cache storage (3) (small & fast)

Application data is handled by the cache with **cache line** granularity. Every **cache line** is associated with corresponding **core line** on the backend storage. However not every **core line** has corresponding **cache line** at a particular moment in time. If it has, we say that this **core line** is *mapped* into the **cache**. Otherwise it’s *unmapped*.

If an application performs an I/O operation, the **cache** checks if **core lines** being accessed are already mapped to **cache lines** in the **cache**. If they’re not, then we deal with a **cache miss**. In such situation the **cache engine** has to access data on the **backend storage**, which is slower than accessing data on the **cache storage**. After that, depending on current **cache mode** cache engine may decide to map the **core line** into **cache**, and thus perfom a **cache insert** operation.

On the other hand, if the accessed **core line** is mapped into cache, then we have a **cache hit**. It’s the optimistic scenario, in which we are able to serve data to the application directly from the **cache storage**.

Cache object

In OCF a **cache** object provides an interface for interacting with the cache state machine. Its API allows for attaching **cache storage**, adding and removing **cores**, modifying various configuration parameters and getting statistics. Every cache operates on a single **cache storage** (represented by the cache **volume**) where it stores data and metadata.

Single **cache** can handle data from many **cores**. It’s called a *multi-core configuration*. The **core** object can become **backend storage** of another **core** or **cache storage** of some **cache**, which makes it possible to construct complex multilevel configurations.

Cache operations

There are several basic operations upon which the entire caching logic is constructed. Getting familiar with them is essential to understanding how OCF caching engines work, and how individual I/O requests are handled depending on the current cache state. The following section provides a brief introduction to these operations.

Mapping

Mapping is an operation of linking up a **core line** with a **cache line** in the **cache** metadata. During *mapping* the **cache storage** area of size of a single **cache line** is being assigned to the **core line**, and during **cache** operation this area can be used for storing **core line** data, until the **cache line** is being *evicted*.

Mapping updates *core id* and *core line number* in **cache line** metadata as well as *hash map* information.

Insertion

Insertion is an operation of writing **core line** data to a **cache line**. During *insertion* the data of I/O request is being written to an area of **cache storage** storing data of the **cache line** corresponding to the **core line** being accessed. The *valid* and *dirty* bits of the **cache line** in the metadata are being updated accordingly during this operation.

Insertion doesn't change **cache line** mapping metadata, but only *valid* and *dirty* bits. Because of that it can be done only after successfull *mapping* operation. However, unlike *mapping*, which operates on whole **cache lines**, *insertion* is performed with sector granularity. Every change of *valid* bit from 0 to 1 is considered an *insertion*.

Insertion may occur for both read and write requests. Additionally, in the *Write-Back* mode the *insertion* may introduce a *dirty data*. In such situation, besides setting a *valid* bit, it also sets a *dirty* bit for accessed sectors.

Update

Update is an operation of overwriting **cache line** data in the **cache storage**. It's performed when the **core line** accessed during a write request is mapped into the **cache** and the accessed sectors are marked as *valid*. When some of the written sectors within the **cache line** are *invalid*, then those sectors are *inserted*, which means that *update* and *insertion* can be done together by a single I/O request.

Update can be performed only during a write request (a read can never change the data) and similarly to *insert* it can introduce *dirty data*, so it may update *dirty* bit for accessed sectors in the **cache line**. However it will never change *valid* bit, as *update* touches only sectors that are already *valid*.

Invalidation

Invalidation is an operation of marking one or more **cache line** sectors as *invalid*. It can be done for example during handling of *discard* request, during *purge* operation, or as result of some internal **cache** operations. More information about cases when *invalidation* is performed can be found here.

Eviction

Eviction is an operation of removing **cache line** mappings. It's performed when there is not sufficient space in the **cache storage** for *mapping* data of incoming I/O requests (when cache occupancy is close to 100%). The **cache lines** to be *evicted* are selected by an *eviction policy* algorithm. Currently OCF supports one *eviction policy*, the *Least Recently Used* (*LRU*) which selects **cache lines** that have not been accessed for the longest period of time.

Flushing

Flushing is an operation of synchronizing data in the **cache storage** and the **backend storage**. It involves reading sectors marked as *dirty* from the **cache storage**, writing them to the **backend storage**, and setting *dirty* bits in **cache line** metadata to zero.

Flushing is a **cache** management operation, and it has to be triggered by the user, and it's typically done in order to safely detach the **core** from the **cache** without risking data loss.

Cleaning

Cleaning is an operation very simillar to *flushing*, but it's performed by the **cache** automatically as a background task. Process of *cleaning* is controlled by a *cleaning policy*. OCF currently supports three *cleaning policies*:

- *Approximately Least Recently Used* (*ALRU*),
- *Aggressive Cleaning Policy* (*ACP*),
- *No Operation* (*NOP*).

Cache configuration

This page describes various **cache** configuration parameters. They may be used to alter **cache** behavior in some situations and to configure the way in which I/O requests are handled by the cache engines.

Cache mode

The **cache mode** parameter determines how incoming I/O requests are handled by the cache engine. Depending on the **cache mode** some requests may be inserted to the **cache** or not. The **cache mode** determines also if data stored on **cache** should always be coherent with data stored in the **backend storage** (if there is a possibility of **dirty data**).

Currently there are six **cache modes** supported by OCF:

- *Write-Through (WT)*,
- *Write-Back (WB)*,
- *Write-Around (WA)*,
- *Write-Invalidate (WI)*,
- *Write-Only (WO)*.
- *Pass-Through (PT)*.

Write-Through

In *Write-Through* mode, the **cache engine** writes data to the **cache storage** and simultaneously writes the same data “through” to the **backend storage**. *Write-Through* ensures the data written to the **core** is always in sync with data on the **backend storage**. This mode will accelerate only read operations, as writes need to be performed on both **backend** and **cache storages**.

Write-Back

In *Write-Back* mode, the **cache engine** writes the data first to the **cache storage** and acknowledges to the application that the write is completed before the data is written to the **backend storage**. Periodically, those writes are “written back” to the **backend**

storage opportunistically (depending on **cleaning policy**).
While *Write-Back* mode will improve both write and read intensive operations, there is a risk of data loss if the **cache storage** fails before the data is written to the **backend storage**.

Write-Around

In *Write-Around* mode, the **cache engine** writes data to the **cache storage** if and only if that **cache line** is already mapped into the **cache** (it can be mapped during read request) and simultaneously writes the same data “through” to the **backend storage**. *Write-Around* is similar to *Write-Through* in that it ensures the **core** is 100% in sync with the **backend storage** and in that this **cache mode** will accelerate only read intensive operations. *Write-Around* further optimizes the **cache** to avoid *cache pollution* in cases where written data is not often subsequently re-read.

Write-Invalidate

In *Write-Invalidate* mode, the **cache engine** handles write requests by writing data directly to **backend storage**, and if written **cache line** is already mapped into the **cache**, then it becomes **invalid**. In this mode only read requests are mapped into the **cache** (they are handled the same way as in *Write-Through* mode). *Write-Invalidate* mode will improve only read intensive operation. It also reduces number of **eviction** operation for workloads where written data is not often subsequently read.

Write-Only

In *Write-Only* mode, the **cache engine** writes the data exactly like in *Write-Back* mode so the data is written to **cache storage** without writing it to **backend storage** immediatelly. Read operations do not promote data to **cache**. *Write-Only* mode will accelerate only write intensive operations, as reads need to be performed only on the **backend storage**. There is a risk of data loss if the **cache storage** fails before the data is written to the **backend storage**.

Pass-Through

In *Pass-Through* mode, the **cache engine** will bypass the **cache** for all operations. This allows the user to enable and disable caching dynamically (by switching between *Pass-Through* and other **cache modes**) without need to manually alter I/O path in the application. If the **cache** contained a **dirty data** before switching to *Pass-Through* mode, then **read hits** will be handled by reading data from **cache storage** until all the cache lines will be **cleaned**.

Cache line size

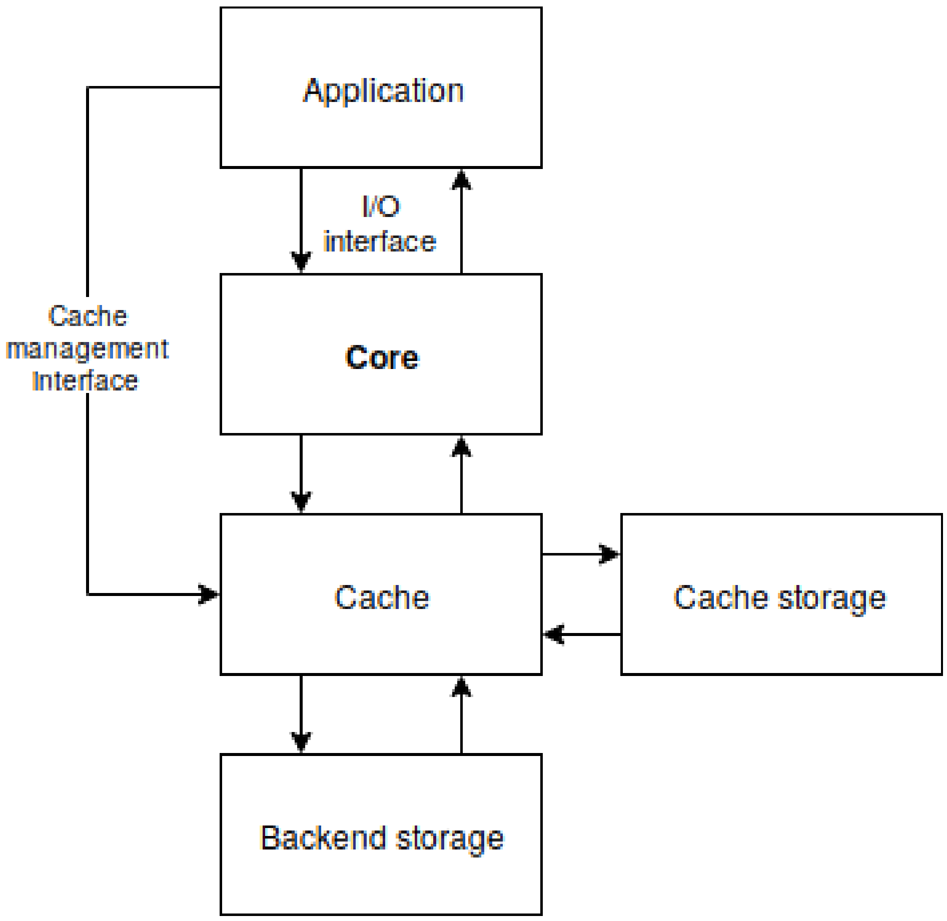
The **cache line size** parameter determines the size of a data block on which the **cache** operates (**cache line**). It's the minimum portion of data on the **backend storage** (**core line**) that can be mapped into the **cache**.

OCF allows to set **cache line size** to one of the following values:

- 4 KiB,
- 8 KiB,
- 16 KiB,
- 32 KiB,
- 64 KiB.

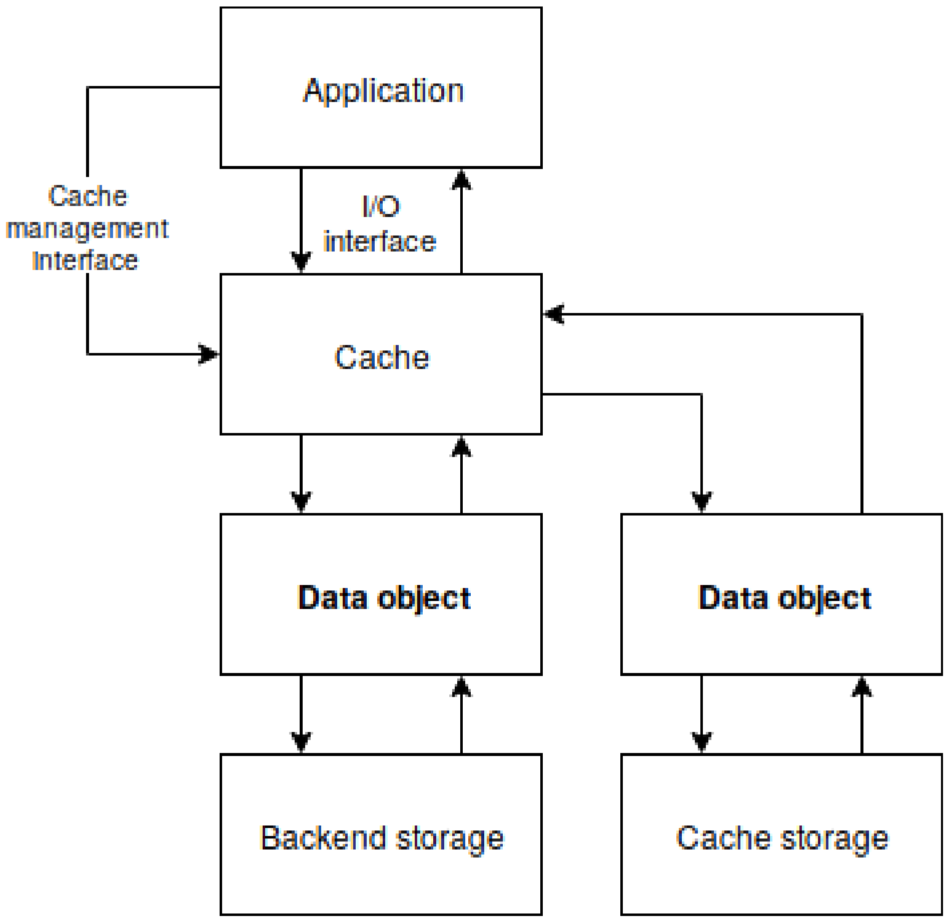
What is core?

The **core** object is an abstraction that allows application access to **backend storage** cached by a **cache**. It provides API for submitting I/O requests, which are handled according to current **cache mode** and other **cache** and **core** configuration parameters. During normal **cache** operation the **backend storage** is exclusively owned by the **cache** object, and application should never access it directly unless the **core** is removed from the **cache**. That's it, using the **core** API is the only proper way of accessing the data on the **backend storage**.



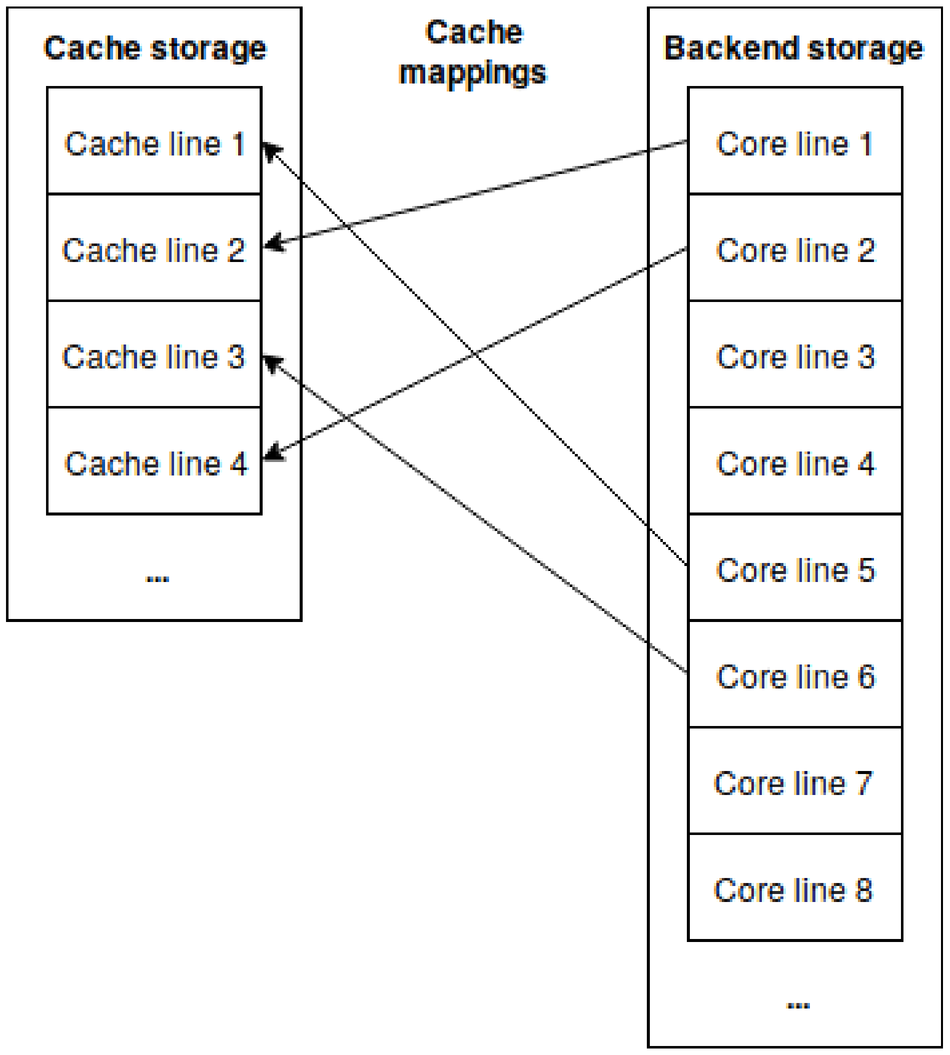
What is volume?

A **volume** is generic representation of a storage, that allows OCF to access different types of storages using common abstract interface. OCF uses a **volume** interface for accessing both **backend storage** and **cache storage**. Storage represented by **volume** may be any kind of storage that allows for random block access - it may be HDD or SSD drive, ramdisk, network storage or any other kind of non-volatile or volatile memory.



What is cache line?

A **cache line** is the smallest portion of data that can be mapped into a **cache**. Every mapped **cache line** is associated with a **core line**, which is a corresponding region on a **backend storage**. Both the **cache storage** and the **backend storage** are split into blocks of the size of a **cache line**, and all the **cache** mappings are aligned to these blocks. The relationship between a **cache line** and a **core line** is illustrated on the picture below.



Cache line metadata

OCF maintains small portion of metadata per each **cache line**, which contains the following information:

- *core id*,
- *core line number*,
- *valid* and *dirty* bits for every sector.

The *core id* determines on which **core** is the **core line** corresponding to given **cache line** located, whereas the *core line number* determines precise **core line** location on the **backend storage**. *Valid* and *dirty* bits are explained below.

Valid and dirty bits

Valid and *dirty* bits define current **cache line** state. When a **cache line** is *valid* it means, that it’s mapped to a **core line** determined by a *core id* and a *core line number*. Otherwise all the other **cache line** metadata information is invalid and should be ignored. When a **cache line** is in *invalid* state, it may be used to map **core line** accessed by I/O request, and then it becomes *valid*. There are few situations in which a **cache line** can return to *invalid* state:

- When the **cache line** is being **evicted**.
- When the **core** pointed to by the *core id* is being **removed**.
- When the **core** pointed to by the *core id* is being **purged**.
- When the entire **cache** is being **purged**.
- During *discard* operation being performed on the corresponding **core line**.
- During I/O request when a **cache mode** which may perform an *invalidation* is selected. Currently these **cache modes** are *Write-Invalidate* and *Pass-Through*.

The *dirty* bit determines if the **cache line** data stored in the **cache** is in sync with the corresponding data on the **backend storage**. If a **cache line** is *dirty*, then only data on the **cache storage** is up to date, and it will need to be **flushed** at some point in the future (after that it will be marked as *clean* by zeroing *dirty* bit). A **cache line** can become dirty only during write operation in *Write-Back* mode.

Valid and *dirty* bits are maintained per sector, which means that not every sector in a *valid* **cache line** has to be *valid*, as well as not every sector in a *dirty* **cache line** has to be *dirty*. An entire **cache line** is considered *valid* if at least one of its sectors is *valid*, and similarly it’s considered *dirty* if at least one of its sectors is *dirty*.

OCF Quick Setup

This page describes the basic steps to get started on Open Cache Acceleration Software (CAS) Framework

About

The Open CAS Framework (OCF) is a high performance block storage caching meta-library. OCF is not intended to compile on its own as it represents the core caching API. OCF is intended to be used as a caching engine by applications that implement their own adaptation layers.

There are existing implementations which include OCF and provide a more comprehensive functionality. Some of these implementations include:

- **Open CAS Linux**.
  Open CAS Linux implements OCF to provide a Linux kernel adapter
- **SPDK OCF Bdev**.
  SPDK OCF Bdev implements OCF to provide a virtual block device to use inside SPDK

OCF contains a sample application to exemplify the usage of OCF. This getting started guide will show how to download OCF and compile this sample application.

OCF is hosted in this **GitHub repository**.

Getting Started

1. First clone the OCF project

   ```
   git clone https://github.com/Open-CAS/ocf
   ```

2. Change the current directory to the project folder

   ```
   cd ocf
   ```

3. Change directories to example application and compile it

   ```
   cd example/simple
   make
   ```

Open CAS Linux Quick Setup

This page describes the basic steps to get started on Open CAS Linux

About

Open CAS Linux builds upon the Open CAS Framework by implementing kernel adapters for the Linux operating system. The Open CAS Linux project is hosted in this **GitHub repository**.

Getting Started

1. First clone the Open CAS Linux project

   ```
   git clone https://github.com/Open-CAS/open-cas-linux
   ```

2. Change current directory to project folder and update submodules

   ```
   cd open-cas-linux
   git submodule update --init
   ```

3. Configure Open CAS Linux

   ```
   ./configure
   ```

4. Compile Open CAS Linux and install it

   ```
   make
   make install
   ```

5. Verify the kernel modules were inserted by checking their versions

   ```
   casadm -V
   ```

6. CAS should now be ready to start. For example, to use block device /dev/nvme0n1 as a caching device:

   ```
   casadm -S -d /dev/nvme0n1
   ```

7. The output should return the cache instance number, use it to add a backend device. For example, to use block device /dev/sda1 as a backend device to cache instance 1:

   ```
   casadm -A -d /dev/sda1 -i 1
   ```

8. Verify CAS instance is operational with:

   ```
   casadm -L
   ```

The output should state status is *Running*. Additionally, a command to list block devices such as `lsblk` should show the exported CAS device for example */dev/cas1-1*

- To stop the cache device execute a command similar to `casadm -T -i 1`.

SPDK Quick Setup

This page describes the basic steps to get started on SPDK with OCF block devices.

About

SPDK (Storage Performance Development Kit) is a set of tools and libraries for writing high performance, user-mode storage applications. The SPDK and OCF teams worked in conjunction to develop an OCF virtual block device (vbdev) submodule for SPDK. The SPDK project is hosted in this **GitHub repository**.

Getting Started

1. First clone the SPDK project

   ```
   git clone https://github.com/spdk/spdk.git
   ```

2. Change current directory to project folder and update submodules

   ```
   cd spdk
   git submodule update --init
   ```

3. Configure SPDK to use OCF

   ```
   ./configure --with-ocf
   ```

4. Compile SPDK

   ```
   make
   ```

5. An SPDK application can now be started. For example, to use the SPDK target app and configure OCF via remote procedure calls (RPCs) :

   - Setup SPDK

     ```
     scripts/setup.sh
     ```

   - Start the SPDK Target App in the background

     ```
     sudo app/spdk_tgt/spdk_tgt &
     ```

- Configure an NVMe bdev named *Nvme0* using PCI device at address 0000:00:0e.0

  `sudo scripts/rpc.py construct_nvme_bdev -b Nvme0 -t PCIe -a 0000:00:0e.0`

- Configure a Memory bdev named *Malloc0* using 200MB of RAM and 4096 KB block size

  `sudo scripts/rpc.py construct_malloc_bdev -b Malloc0 200 4096`

- Create a write-through (wt) OCF bdev named *Cache1* using Malloc0 as a caching device for Nvme0n1

  `sudo scripts/rpc.py construct_ocf_bdev Cache1 wt Malloc0 Nvme0n1`

- The output should state the OCF bdev is *Successfully added*. Additionally, the RPC command to list OCF bdevs `sudo scripts/rpc.py get_ocf_bdev` should show the CAS device

- To delete the OCF bdev execute an RPC command similar to `sudo scripts/rpc.py delete_ocf_bdev Cache1`

Open CAS Linux - Admin Guide

Introduction

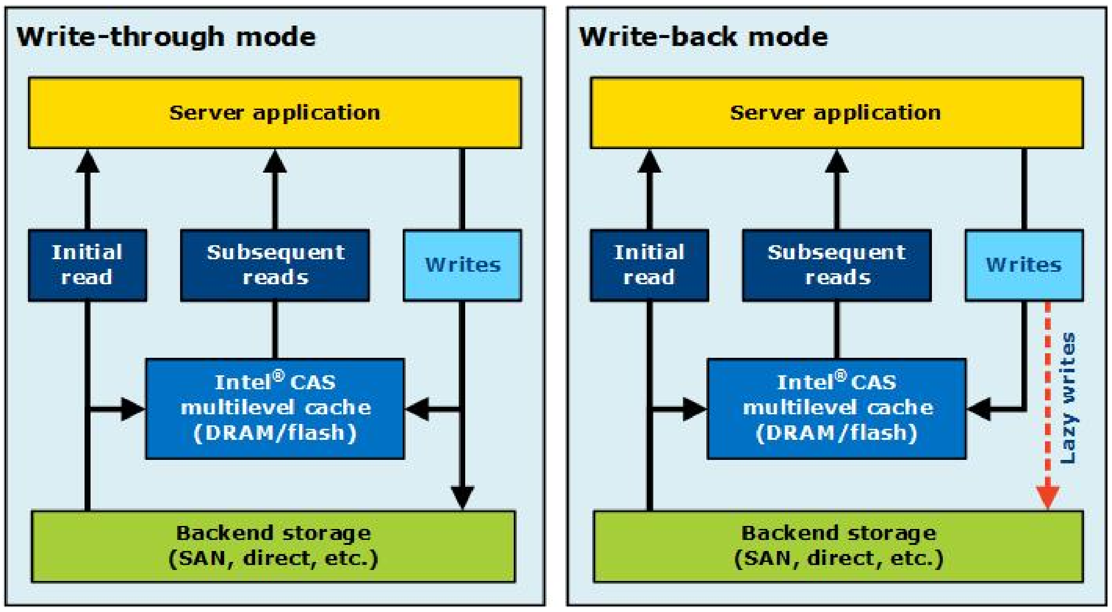
This guide offers instruction on the installation and use of Open Cache Acceleration Software for Linux (Open CAS Linux). It will be helpful for the end user to have a basic knowledge of storage management and Linux* system administration.

Open CAS Linux Overview

Open CAS Linux accelerates Linux applications by caching active (*hot*) data to a local flash device inside servers. Open CAS Linux implements caching at the server level, using local high-performance flash media as the cache drive media within the application server, thus reducing storage latency.

Open CAS Linux installs into the Linux operating system as a kernel module. The nature of the integration provides a cache solution that is transparent to users, applications and your existing storage infrastructure. No storage migration or application changes are required.

As shown in the figure below, initial read data is retrieved from backend storage and copied to the Open CAS Linux cache. A second read promotes data to system memory. Subsequent reads are returned at high-performance RAM or flash speed. In Write-through mode, all data is written synchronously to both the backend storage and the cache. In Write-back mode, all data is written to the cache then eventually flushed to the backend storage. When the cache is full, newly identified active data evicts stale data from the cache, utilizing the Open CAS Linux proprietary eviction algorithm.


Write-through mode
Server application
Initial read
Subsequent reads
Writes
Intel® CAS multilevel cache (DRAM/flash)
Backend storage (SAN, direct, etc.)
Write-back mode
Server application
Initial read
Subsequent reads
Writes
Lazy writes
Intel® CAS multilevel cache (DRAM/flash)
Backend storage (SAN, direct, etc.)

Block Diagram - Write-through and Write-back modes

Open CAS Linux, by default, employs a block-based caching architecture that caches all of the activity on the selected core device.

Documentation Conventions

The following conventions are used in this manual:

```
code examples, command line entries (may also begin with #)
file content examples
```

- *Italic* - user interface items, filenames, etc.
- **< > -** denotes mandatory commands
- **[ ] -** denotes optional commands
- `command` - highlights or differentiates a command to execute from other text

References

Refer to the resources and tools listed in the following table for assistance with Open CAS Linux testing and operations, or to learn more about caching and application IO performance management.

Reference Documents

Name	**Location**
DT generic data testing program	https://github.com/RobinTMiller/dt
FIO	https://github.com/axboe/fio
Vdbench	http://vdbench.sourceforge.net

Revision History

Revision Number	**Description**	**Date**
01	Initial Release	April 2018
02	Updated Information for Write-only Mode	September 2019

Open CAS Linux - Admin Guide

System Requirements

The following table lists system requirements for Open CAS Linux.

Component	Requirement
Memory	RAM requirement is approximately 1GiB + (2% * 4KiB/<selected cache line size> + 0.05%) * cache device capacity. For example, when using 4KiB (default) cache line size, RAM requirement is approximately 1GiB + 2.05% cache device capacity.
CPU Overhead	Open CAS Linux consumes minimal CPU bandwidth (less than 10% in most cases). Ensure that Open CAS Linux has adequate CPU capacity to function optimally.
Flash/SSD	Any Linux flash device (SAS, SATA, PCIe*, Fibre Channel, RAID) is supported, and can be direct attached, expander attached, or attached via SAN (with a single worker).
Storage	Primary storage devices: Any device that appears as a local block device. For example, local disk, RAID, SAN iSCSI (via Fibre Channel, Infiniband, Ethernet), etc. Core device (HDD) logical block size must be 512 bytes or larger. Operating system must be on separate partition or drive than the core storage

Component	Requirement
	device to be cached.
File Systems	The following file systems are supported for primary storage or core storage: • ext3 (limited to 16TiB volume size). This is an ext3 file system limitation • ext4 • xfs
Software	The following prerequisite software must be installed prior to Open CAS Linux installation: • sed • make • gcc • kernel-devel • kernel-headers • python3 • lsblk • argparse (python module) • elfutils-libelf-devel
Power Management	Power Management Suspend (S3) and Hibernate (S4) states must be disabled.

NOTE: The cache device logical block size must be smaller than or equal to the logical block size of the core storage device.

NOTE: For best performance, core device file system block size should match the core device partition logical block size.

NOTE: For best performance, IO request sizes in multiples 4KiB are recommended.

Open CAS Linux - Admin Guide

Installing Open CAS Linux

This section describes how to perform a typical installation of Open CAS Linux. The installation package consists of loadable kernel modules that provide the caching layer and a management CLI that controls caching behavior.

Disabling Sleep States

Before configuring Open CAS Linux, you must disable Power Management Suspend (S3) and Hibernate (S4) states on your OS. Consult your OS documentation for how to accomplish this on your particular OS.

Configuring the Flash Device

Before configuring Open CAS Linux, you must have a flash device installed. The flash device can be any solid state drive (SSD) or any PCIe* flash drive supported by the Linux operating system (see System Requirements for details).

Follow the instructions in the installation guide included with the flash device, and make note of the device’s serial number or WWN so you can identify it once the system is up and running. All devices to be used for Open CAS Linux should be un-mounted and any auto-mount procedures disabled. When using a partition of a device as cache space, make sure the partition is aligned to the device’s physical sector size for best performance.

For best performance, it is recommended to use the *noop* (or *None* for NVMe devices) IO scheduler on the cache device (SSD). Initially, the cas virtual block device that is created when caching is enabled will inherit the IO scheduler of the primary storage device. If desired, the user can change the IO schedulers of the virtual block device and primary storage device independently after the virtual block device is created.

NOTE: Mounting file systems by label, such as in */etc/fstab*, should not be used in older SysV based operating systems (RHEL 6.x, CentOS 6.x, etc) because it interferes with proper startup of Open CAS Linux. This is due to the file system already being mounted by the operating system when Open CAS Linux tries to start.

Code compilation

Before you begin installation, log on as an administrator or verify that you have the requisite privileges to install software. You must have “root” privileges or login as “root” in order to proceed.

1. Download or clone the Open CAS Linux source files to a directory on the target system. The installation instructions use the example of ~ or ~/ (equivalent of $HOME) on the server file system. Optionally, Open CAS Linux can be cloned with:

   ```
   git clone https://github.com/Open-CAS/open-cas-linux
   ```

2. Change current directory to project folder and update submodules

   ```
   cd open-cas-linux
   git submodule update --init
   ```

3. Configure Open CAS Linux

   ```
   ./configure
   ```

4. Compile Open CAS Linux and install it

   ```
   make
   make install
   ```

5. The main outputs of the compilation will be :

   - cas_disk.ko - CAS Disk Kernel Module
   - cas_cache.ko - CAS Cache Kernel module
   - casadm - CAS Administration Tool Note that the kernel modules will be inserted during the installation phase

   - Verify casadm shows the inserted kernel module versions

```
casadm -V
```

Open CAS Linux - Admin Guide

Configuring Open CAS Linux

Open CAS Linux Configuration File - *utils/opencas.conf*

Once Open CAS Linux has been installed, the next stop for the Open CAS Linux administrator should be the *opencas.conf*. This file is the primary point for system cache and core device configuration and is essential to correct Open CAS Linux operation.

The file is read by system udev rules upon startup. For each block device that appears in the system, rules check to see if the device is in the *opencas.conf*, and take action accordingly. The file is divided into two main sections:

1. Caches configuration section
2. Core devices configuration

The Caches configuration section specifies:

1. Cache ID. This indicates the ID used for starting a cache instance on a given device.
2. Path to the NVMe or other cache device(s). The recommended cache device is a Non-volatile Memory Express (NVMe) solid state drive. Open CAS Linux has been engineered to take advantage of the high speed and low latency of even the latest NVMe drives.
3. Desired cache mode.
4. Extra fields, such as the full path to the custom IO Classification file.

The Core devices configuration section specifies:

1. Cache ID. Specifies which cache device each core corresponds to.
2. Core ID. Indicates the ID used for core device(s) corresponding to a specific cache within the system.
3. Path to the core device(s).

Example of caches and cores configuration in an operational *opencas.conf* file:

```
## Caches configuration section

[caches]

## Cache ID Cache device Cache mode Extra fields (optional)

1 /dev/disk/by-id/nvme-INTEL_SSD WT
ioclass_file=/etc/opencas/ioclass-config.csv

## Core devices configuration

[cores]

## Cache ID Core ID Core device

1 1 /dev/disk/by-id/wwn-0x50014ee0aed22393

1 2 /dev/disk/by-id/wwn-0x50014ee0042769ef

1 3 /dev/disk/by-id/wwn-0x50014ee00429bf94

1 4 /dev/disk/by-id/wwn-0x50014ee0aed45a6d

1 5 /dev/disk/by-id/wwn-0x50014ee6b11be556

1 6 /dev/disk/by-id/wwn-0x50014ee0aed229a4

1 7 /dev/disk/by-id/wwn-0x50014ee004276c68
```

Further details are available in the complete default *utils/opencas.conf* file, as follows:

```
version=19.03.00
# Version tag has to be first line in this file

# Open CAS configuration file - for reference on syntax
# of this file please refer to appropriate documentation

# NOTES:
# 1) It is highly recommended to specify cache/core device
using path
# that is constant across reboots - e.g. disk device links in
# /dev/disk/by-id/, preferably those using device WWN if
available:
# /dev/disk/by-id/wwn-0x123456789abcdef0
# Referencing devices via /dev/sd* may result in cache
misconfiguration after
# system reboot due to change(s) in drive order.

## Caches configuration section
[caches]
## Cache ID Cache device Cache mode Extra fields (optional)
## Uncomment and edit the below line for cache configuration
#1 /dev/disk/by-id/nvme-INTEL_SSDP.. WT

## Core devices configuration
[cores]
## Cache ID Core ID Core device
## Uncomment and edit the below line for core configuration
#1 1 /dev/disk/by-id/wwn-0x123456789abcdef0

## To specify use of the IO Classification file, place content of
the following line in the
## Caches configuration section under Extra fields (optional)
## ioclass_file=/etc/opencas/ioclass-config.csv
```

Field details:

- *<Cache ID>* is a numeric value between 1 and 16,384 (valid cache instance numbers).
- *<Core ID>* is a numeric value between 0 and 4095 (valid core instance numbers)
- Cache and core devices must point to existing HDD and SSD devices, preferably referenced by the by-id name (ls -l /dev/disk/by-id). Core devices should reference the WWN identifier, while Cache devices should use model and Serial

Number. Alternatively, devices may be referenced by device name (eg. /dev/sdb) such as when running casadm commands.

- Mode determines the cache mode, either write-through, write-back, write-around, write-only, or pass-through.
- *Optional:* Extra flags allow extra settings for the cache and core devices and are comma separated.
    - *ioclass_file* allows the user to load a file that specifies an IO class policy for this cache.
    - *cleaning_policy* allows the user to specify the cache cleaning policy to be used for this cache, either acp, alru, or nop.
    - *promotion_policy* allows the user to specify the promotion policy to be used for this cache, either always or nhit.
    - *cache_line_size* allows the user to specify the cache line size to be used for this cache in kibibytes, either 4, 8, 16, 32, or 64.

      NOTE: During an upgrade, *opencas.conf* files with earlier formats will automatically be converted to the new format.

The Open CAS Linux Configuration Utility - casadm

Open CAS Linux provides a user-level utility called casadm to allow for configuration and management of the caching software.

In using casadm, it is important to understand certain restrictions:

- You must be logged on as root or have root privileges to start, stop, or configure Open CAS Linux.
- You cannot accelerate the partition where the operating system resides.
- If a super user promotes you to root, there is no guarantee that the */sbin* directory will be in your $PATH environment variable. If casadm is not accessible, check this variable first. Use the command’s full path.

If you launch the configuration utility via casadm -H, a list of command line options and arguments is returned. For more detailed information on the different command line options, see the section Configuration Tool Details.

For Open CAS Linux configuration, note that the term “cache device” refers to the SSD/NVMe device or RAM disk that is used for caching data from a slower device, while the term “core device” refers to the slower device to be cached.

Using the Configuration Utility

While configuration of Open CAS Linux via the *opencas.conf* file is highly recommended, the following sections detail how to manually configure Open CAS Linux options using casadm. For more details on available commands for the casadm utility, see the section Configuration Tool Details.

The following is assumed for the subsequent instructions:

- The cache device (SSD) is */dev/sdc*. The cache device is either a raw block device or ram disk accessed as a block device. Ensure that the cache device does not have a file system and is not mounted.
- Back up all data on your cache device before completing these steps as all data will be overwritten.
- The core device (primary storage) to be cached is */dev/sdb*.
- The core device may contain a filesystem (with or without data) or may be a raw block device. See system requirements for specific file system types and limitations for Open CAS Linux. Ensure that the device is not mounted.
- If necessary, perform an *ls -l* or *ll* on the */dev/disk/by-uuid* or */dev/disk/by-id* directory to ensure the correct devices are being configured.
- Core device (HDD) logical block size must be 512 bytes or larger.
- Cache device logical block size must be smaller than or equal to the logical block size of the core storage device.
- Ensure that both devices are removed from */etc/fstab* and any other mechanism that auto mounts either the cache device or core device.

If the Open CAS Linux module is not loaded, follow the installation instructions. If the Open CAS Linux installation fails, contact the OCF community for support.

Manual Configuration for Write-through Mode

In write-through mode, which is the Open CAS Linux default caching mode, the caching software writes data to the flash device and simultaneously writes the same data “through” to the core device (disk drives). Write-through ensures the core device is 100% in sync with the cache and its data is always available to other servers sharing that storage. However, this type of cache will accelerate only read intensive operations.

- Ensure that the core device (*/dev/sdb*) is not mounted and that the cache device (*/dev/sdc*) is not mounted and contains no data to be saved. Entering the following command will display all mount points:

```
# mount
```

- Start a new cache with an ID of “1”:

```
# casadm -S -i 1 -d /dev/sdc
```

- You may notice a brief delay after entering the casadm -S command. Typically, this is less than 60 seconds, but can be longer.
- If the cache device is formatted or a file system already exists, you will need to use the “-f” force flag (for example, casadm -S -d /dev/sdc -f).
- All information on the cache device will be deleted if the -f option is used. Please ensure all data has been backed up to another device (see Configuration Tool Details section for further details).
- Pair the core device to this new cache:

```
# casadm -A -i 1 -d /dev/sdb
```

- The *add-core* command creates a new device in the */dev* directory with the following name format:

cas<cache ID>-<core #> for example: */dev/cas1-1.*

- This new device can be treated as a regular block device.

Manual Configuration for Write-back Mode

In write-back mode, the caching software writes the data first to the cache and acknowledges to the application that the write is completed, before the data is written to the core device. Periodically, those writes are “written back” to the disk opportunistically. While write-back caching will improve both write and read intensive operations, there is a risk of data loss if the cache device fails before the data is written to the core device.

Write-back mode is enabled when starting a new cache device with the option “-c wb”:

```
# casadm -S -i 1 -d /dev/sdc -c wb
```

Pairing of a core device is similar to the previous step for Manual Configuration for Write-through Mode section.

Manual Configuration for Write-around Mode

In write-around mode, the caching software writes data to the flash device if and only if that block already exists in the cache and simultaneously writes the same data “through” to the core device (disk drives). Write-around is similar to write-through in that it ensures the core device is 100% in sync with the cache and in that this type of cache will accelerate only read intensive operations. However, write-around further optimizes the cache to avoid cache pollution in cases where data is written and not often subsequently re-read.

Write-around mode is enabled when starting a new cache device with the option “*-c wa*”:

```
# casadm -S -i 1 -d /dev/sdc -c wa
```

Pairing of a core device is similar to the Manual Configuration for Write-through Mode section.

Manual Configuration for Pass-through Mode

In pass-through mode, the caching software will bypass the cache for all operations. This allows the user to associate all their desired core devices to be cached prior to actually enabling caching. Once the core devices are associated, the user would dynamically switch to their desired caching mode.

Pass-through mode is enabled when starting a new cache device with the option “*-c pt*”:

```
# casadm -S -i 1 -d /dev/sdc -c pt
```

Pairing of a core device is similar to the Manual Configuration for Write-through Mode section.

Manual Configuration for Write-only Mode

In write-only mode, the caching software writes the data first to the cache and acknowledges to the application that the write is completed, before the data is written to the core device. Periodically, those writes are written back to the core device opportunistically. The caching software bypasses the cache for new read operations. Read operations can be served from the caching device only if the data was previously written to the cache device. Write-only caching will improve write intensive operations primarily, however, there is a risk of data loss if the cache device fails before the data is written to the core device.

Write-only mode is enabled when starting a new cache device with the option *"-c wo"*:

```
# casadm -S -i 1 -d /dev/sdc -c wo
```

Pairing of a core device is similar to the Manual Configuration for Write-through Mode section.

Switching Caching Modes

You can switch between different caching modes any time without rebooting or restarting Open CAS Linux. For example, you can switch from WT mode to PT mode to stop caching during maintenance or for testing purposes.

To switch from WB to any other mode, you must specify whether dirty data should be flushed now or later using the –flush-cache option. Depending on the –flush-cache option used, switching from WB to other caching modes may require time to complete before caching mode is changed.

The following example places Open CAS Linux into PT mode for cache id 1.

```
# casadm –set-cache-mode –cache-mode pt –cache-id 1
```

or

```
# casadm -Q -c pt -i 1
```

Automatic Partition Mapping

If the core device to be accelerated has existing partitions, selecting the parent device as the core device to be accelerated (e.g. */dev/sdc* as in the previous examples) will accelerate all of the underlying partitions with a single command.

Open CAS Linux will automatically hide the existing core device partitions (ex. */dev/sdc1*, */dev/sdc2*, etc.) from the Operating System, create partitions on the exported device (e.g. */dev/cas1-1p1*, */dev/cas1-1p2*, etc.), and make the exported device a child of the core device.

Comparison of logical device layout before and after Open CAS Linux acceleration

Before	After
/dev/sdc	*/dev/sdc*

Before	After
/dev/sdc1	*/dev/cas1-1*
/dev/sdc2	*/dev/cas1-1p1*
/dev/sdc3	*/dev/cas1-1p2*
/dev/nvme0n1	*/dev/cas1-1p3*
	/dev/nvme0n1

This scheme ensures compatibility with existing object storage device creation and activation scripts from common software defined storage systems (e.g. Ceph).

Mounting an Open CAS Linux Device as a Data Volume

For some applications, a storage system or cluster may require the Open CAS Linux exported device to be mounted to the file system to allow for its use as a data volume.

The file system should be created directly on the core device before it is added to the cache instance. It’s not recommended to run mkfs commands on the exported device (*casX-Y*) as the process can be very slow.

Format the core device as ext4

```
# mkfs.ext4 /dev/sdb1
```

Format the core device as xfs

```
# mkfs.xfs -f /dev/sdb1
```

Once the cache instance has been started and core devices added, the Open CAS Linux exported name can be used for the mount function. The device designation of *casX-Yp1* will be used for mounting since the file system partition on the exported device is required to mount into the file system directory.

As a reminder, a directory must already exist in order to mount the exported device. In this example, the device is */dev/cas1-1p1* and the mount point directory is */mnt/cache1*. Create the mount point directory if it does not exist.

Mount an Open CAS Linux exported device:

```
# mount -t xfs /dev/cas1-1p1 /mnt/cache1
```

1. Open CAS Linux ensures that no changes are required to the application; it can use the same file system mount point (for example, */local/data*) as the core device previously used.
2. If your application uses the raw device directly (for example, some installations of Oracle*), then you must configure the application to use the exported device (e.g. *cas1-1*) instead.
3. To take advantage of the new TRIM functionality, the file system must be mounted with additional options. Please see the Trim Support section for further details.
4. It may be necessary to change ownership and/or permissions for the mount point directory to grant applications access to the cache. Modify as required to allow access.

Persistent mounting of the Open CAS Linux exported device(s)

After the caching devices have been created and Open CAS Linux is running, you can use the */etc/fstab* file to ensure persistent mounts of your mounted Open CAS Linux volumes. Use of the *fstab* file as follows assumes that Open CAS Linux is active and that exported devices have been configured to agree with the contents of the *fstab*.

```
# cat /etc/fstab
…
/dev/cas1-1p1 /mnt/cache1 xfs defaults 0 0
/dev/cas1-2p1 /mnt/cache2 xfs defaults 0 0
```

NOTE: In the case when the caching device is formatted with a 4KiB physical block size and the backend core device is formatted with a 512B physical block size, the caching device should be re-formatted with a 512B block size so that a cache volume can be built correctly.

Open CAS Linux - Admin Guide

Running Open CAS Linux

This chapter describes how to perform typical administrative activities with Open CAS Linux.

Initial configuration of Cache instances

Once Open CAS Linux is installed, but no cache instances are yet configured and running, the user can easily configure and start cache instances. Refer to the guidance in **Configuring CAS** for configuring the file *utils/opencas.conf*, then follow these steps:

1. Edit and configure caches and cores as desired via the *opencas.conf* file.
2. Execute the following:

```
# casctl init
```

Open CAS Linux will read contents of the *opencas.conf* file, initialize the cache/core pairings, and place caches/core devices into an operational or active status.

1. If the cache device contains data, partitions, or a file system from prior use, it will not normally initialize. If you receive a message during initialization about an existing file system, it may need the –force option to initialize. Be aware that the –force

option will destroy any data or formatting on the device. Caches will be started regardless of existing data or partitions. Execute the following:

```
# casctl init –force
```

This command yields the same results as: casadm -S -d /dev/sdX –force for a single cache instance.

Startup of Cache instances

In order to start all previously initialized cache instances configured in the *opencas.conf* file, the user should execute the following:

```
# casctl start
```

Use of the casctl start command assumes that the configured cache instances were previously running.

Rebooting, Power Cycling, and Open CAS Linux Autostart

Once the *opencas.conf* file is configured correctly to include the cache and core devices, Open CAS Linux devices will automatically become available for use after a restart of the system. That is, Open CAS Linux defaults to being enabled within the system, checks the previously initialized Open CAS Linux configuration, along with contents of the *opencas.conf* file for device configuration, and performs the appropriate actions.

If the user does not wish to start Open CAS Linux caching after reboot, it will be necessary to comment out the cache/core devices in the *opencas.conf* file. This will effectively disable Open CAS Linux and prevent autostart of the devices following a system restart.

If the *opencas.conf* file is not yet configured, and Open CAS Linux devices were previously manually started/added, a system reboot will require another manual restart of the cache and addition of the cores to the cache device.

Start Open CAS Linux using the following command syntax:

```
# casadm -S -i <cache_id> -d <cache_device> -c
<cache_mode>
# casadm -A -i <cache_id> -d <core_device>
```

NOTE: If after installing CAS, your system boots into emergency mode due to the **“Failed to start opencas initialization service.”** error, you need to force SELinux relabelling in permissive mode on your filesystem. The easiest and quickest way to do this is to add those kernel parameters to your grub entry at boot time:

```
autorelabel=1 enforcing=0
```

1. When the grub loads during boot, press *e* on the entry you want to boot.
2. Then navigate to the *kernel/linux* line and add those parameters at the end.
3. After that press *Ctrl+x* to boot.

This will set those changes only for the current boot, so you don’t need to change them back again.

Alternatively you can change SELinux mode in */etc/selinux/config*:

```
SELINUX=permissive
```

then force relabelling by creating an empty *.autorelabel* file in the / directory:

```
touch /.autorelabel
```

and reboot your system.

Stopping Cache Instances

Prior to a shutdown or reboot of your system, it is recommended to cleanly stop Open CAS Linux whenever possible. Even though Open CAS Linux will typically restart cleanly following a server hang or pulled plug scenario, protection of data is always of paramount consideration.

In order to stop all cache instances which are configured via the *opencas.conf* file, the user should execute the following:

```
# casctl stop
```

If the operating Open CAS Linux mode is write-back mode and dirty data may exist within the caching environment, Open CAS Linux must be stopped using:

```
# casctl stop –flush
```

The most common reason for corrupt data or other data irregularities within an Open CAS Linux environment is improper accounting for dirty data as part of a system or cluster change. It is not recommended to operate a system or cluster which contains dirty data as both an Open CAS Linux-enabled and Open CAS Linux-disabled platform as may be relevant to a test environment. Close awareness, and flushing, of dirty data prior to system shutdown and changes are highly recommended.

Caution: If the file system changes while caching is not running, data loss may occur. Do not issue any IO to the core device until caching has successfully restarted.

Disabling Open CAS Linux

In order to disable Open CAS Linux on one or more devices, the user should perform either of the following operations:

1. Stop caching via casctl stop (–flush as appropriate). Remove or comment devices from the *opencas.conf* file.
2. Stop caching or remove cores using casadm -T or casadm -R commands. Remove or comment devices from the *opencas.conf* file.

Handling an Unplanned Shutdown

An unplanned shutdown is any time Open CAS Linux devices are not shutdown properly as described in the previous section. This can include power cycling the system, or a power loss to the system, without properly shutting down Open CAS Linux devices.

After an unplanned shutdown, there are two options for restarting caching. The first option is to start the cache in load mode (-l option), which will utilize the dirty data currently in cache for faster recovery, and no flushing of cache will be done. This process is performed automatically in accordance with system Udev rules.

The second option is to reinitialize the cache with new metadata, which will clear the cache, resulting in the loss of all dirty data that is currently in the cache. See the following options for details.

Caution: If the file system changes while caching is not running, data loss may occur. Do not issue any IO to the core device until caching has successfully restarted.

Recovering the Cache

To manually start Open CAS Linux with recovery enabled, enter the following command. All previously attached cores will be reattached to the cache after this command.

```
# casadm -S -d /dev/sdc -l
```

For more details, see ‘-S | –start-cache’.

Reinitializing the Cache

To clear or empty the cache device (invalidate the entire cache), manually start the cache including the -f parameter (and omitting the -l parameter), as shown below:

```
# casadm -S -d /dev/sdc -f
# casadm -A -i 1 -d /dev/sdb
```

This reinitializes the cache instead of loading the old state and results in zero cache occupancy.

Reinitializing the cache with new metadata will destroy all write-cached data that has not yet been flushed to the disk. In addition, reinitializing the cache with the -force (-f) option is still valid and will destroy all existing cache data.

Open CAS Linux maintains state across a reboot. If the system is rebooted without starting the caching subsystem, then the state of the cache data may become out of sync with the data in primary storage (core device). In this case, restarting the cache system without clearing the cache may result in data corruption.

Device IO Error Handling

In the event of a read or write IO error from either the cache device (SSD) or the core device (HDD), Open CAS Linux will intelligently handle the error and, if possible, will return the requested data and continue processing IO in certain cases.

Cases where Open CAS Linux will *return data* and *continue processing IO* include *cache device IO errors* when attempting to:

- Read clean data from the cache
- Write data to the cache on a read operation (attempting to cache the data that was read)
- Write data to the cache in write-through or write-around modes on a write operation

Cases where Open CAS Linux will *return an error* to the calling application and *stop processing IO* for any exported devices (e.g. cas1-1) served by the cache SSD on which the IO error occurred include *cache device IO* errors when attempting to:

- Read dirty data from the cache
- Write data to the cache in write-back mode or write-only mode

Cases where Open CAS Linux will *return an error* to the calling application and *continue processing IO* include *core device IO errors* when attempting to:

- Read data from the core device
- Write data to the core device

Open CAS Linux - Admin Guide

IO Classification

IO Class Configuration

Open CAS Linux provides the ability to control data caching with classification granularity. Open CAS Linux can analyze every IO on the fly to determine whether the requested block is filesystem metadata or data and, if it is data, the size of the destination file. Using this information the administrator can determine the best IO class configuration settings for the typical workload and can set which IO classes to cache and not to cache, and set a priority level for each IO class. As a result, when it becomes necessary to evict a cache line, the software will evict cache lines of the lowest priority first (a major improvement compared to traditional LRU eviction).

The IO classification is configured using an IO classification file. The table below summarizes the configurable fields in this IO classification file.

IO Class Configuration File Fields

Field	Description
IO class id	Unique ID for the IO class. (NOTE: The ID for unclassified must always be 0)
IO class name	The name of the IO class. The name can be a known class name such as *metadata* or *unclassified* or *direct* . It can also consist of a user-defined rule. A rule consists of conditions separated by logical operators and/or arithmetic operators.
Eviction priority	Sets the priority number for the IO class. Priority range is 0-255 with zero having the highest priority. The IO classes with the lowest priority will be evicted first.
Allocation	Boolean value that allows the user to decide whether data of this IO class will be cached or not. 0=do not cache, 1=cache.

IO class names can consist of user-specified rules to satisfy. Multiple conditions can be combined using logical operators and can be fine-tuned using arithmetic operators. The available operators and names are specified below.

IO Class Configuration File Logical Operators

Operator	Description
&	Logical and
|	Logical or

IO Class Configuration File Arithmetic Operators

Operator	Description
eq	Equals
ne	Not equal
lt	Less than
gt	Greater than
le	Less than or equal to
ge	Greater than or equal to

Available IO Class Names and Conditions

IO Class Name	Description
unclassified	The IO cannot be classified by any other class ID.
metadata	The IO contains filesystem metadata.
direct	The IO is flagged to be performed directly on the device without buffering. This may be set by the application using the O_DIRECT flag.
core_id	The IO belongs to a particular core. For example to classify the IO to a core with id 1: core_id:eq:1

IO Class Name	Description
file_size	The IO belongs to a file with a specific size in bytes. For example if the file size is less or equal to 4096 bytes: file_size:le:4096
directory	The IO belongs to a specific directory path. For example: directory:/data/datab
io_class	The IO can be classified by a previously defined io_class ID. For example, to reference IO class with ID 3: io_class:3
extension	The IO belongs to a file with a specific extension. For example: extension:txt
file_name_prefix	The IO belongs to a file with a specific name prefix. For example, all files with name starting with foo: file_name_prefix:foo
lba	The IO belongs to a specific range of lba's on core device. For example: lba:ge:2000&lba:le:5000
pid	The IO was triggered by a process with particular pid. For example: pid:eq:7890
process_name	The IO was triggered by a process with particular name. For example: process_name:fio
file_offset	The IO belongs to an offset within a file. For example: file_offset:gt:2000 &file_offset:lt:5000
request_size	The IO belongs to a request with particular size.

IO Class Name	Description
	For example: request_size:ge:8192&request_size:le:16384&done
wlth	The IO uses application/user-space write hints. For example: wlth:eq:0

The table below shows the structure of the IO classification file. This table shows IO class 0 is unclassified and has a high eviction priority of 22 to ensure unclassified data is evicted last. The allocation of 1 specifies this data type is cachable.

IO Class Configuration File Structure

IO Class	IO Class Name	Eviction Priority	Allocation
0	unclassified	22	1

The IO classification file entries are comma-separated and should follow this format:

IO class id,IO class name,Eviction priority,Allocation
0,unclassified,22,1

Open CAS Linux iterates over all the IO classes specified in the classification file and if an I/O request satisfies the given rule, the I/O will be assigned to this IO class and move on to the next IO class in the file. Open CAS Linux can optionally be configured to terminate upon IO class assignment and not move on to the next IO class with the special keyword *“&done”*.

Enabling IO Classification

- To enable IO classification and selective allocation, first look at the provided example IO class configuration file and edit it to suit your needs:

```
# vi utils/ioclass-config.csv
```

- After you have completed your changes to the example IO class configuration and saved the file, you must load the configuration for your cache device with ID number represented by <ID> from file <FILE>:

```
# casadm --io-class --load-config --cache-id <ID> -f <FILE>
```

- Verify that the configuration file was successfully loaded:

```
# casadm --io-class --list --cache-id <ID>
```

File Size-Based Caching

The default IO class configuration file (which has been revised from previous versions) allows the user to specify the file size range of interest using keyword *"file_size"*. It is shown below.

```
IO class id,IO class name,Eviction priority,Allocation

0,unclassified,22,1
1,metadata&done,0,1
11,file_size:le:4096&done,9,1
12,file_size:le:16384&done,10,1
13,file_size:le:65536&done,11,1
14,file_size:le:262144&done,12,1
15,file_size:le:1048576&done,13,1
16,file_size:le:4194304&done,14,1
17,file_size:le:16777216&done,15,1
18,file_size:le:67108864&done,16,1
19,file_size:le:268435456&done,17,1
20,file_size:le:1073741824&done,18,1
21,file_size:gt:1073741824&done,19,1
22,direct&done,20,1
```

Directory-Based Caching

IO classification by directory is also configurable using the keyword *"directory"*. Directory-based caching adds new options to user-specified caching, and works in conjunction with file size-based caching.

The IO classification example below shows a directory caching entry:

```
IO class id,IO class name,Eviction priority,Allocation
```

```
0,unclassified,22,0
1,directory:/data/dataa|directory:/data/datab,1,1
2,metadata,3,1
```

In the above example, to specify the directories to be cached, IO Class 1 shows an IO Class Name of:

```
directory:/data/dataa|directory:/data/datab
```

This example specifies that all data in the /data/dataa or /data/datab directories will be cached; unclassified data will not be cached; metadata will be cached (due to allocation 1 or 0).

If a directory referenced in classification rule “directory” condition is modified (created / removed / moved), it might take a few seconds until the CAS classification procedure picks up the change and starts classifying IO according to the changed directory contents.

Combined IO Classification Rules

The IO classification example below shows a combination of file size and directory based caching. Class ID 4 demonstrates that ID’s may be combined to form a new classification rule:

```
IO class id,IO class name,Eviction priority,Allocation

0,unclassified,22,1
1,metadata,1,1
2,direct,1,1
3,file_size:gt:40960,1,1
4,io_class:3&file_size:lt:81920&done,1,1
6,directory:/mnt/drv1/dir,1,1
7,directory:/mnt/drv1/dir/subdir/|
directory:/mnt/drv1/dir/subdir2/,1,1
```

IO class name length limit is 1024 characters. If exceeded, IO class configuration will fail. The valid IO class id range is 0-32. Please note that multiple classification rules must be combined using the same logical operator. For example, *metadata|file_size:lt:4096|io_class:3* only uses the “**|**” operator. Or in another example, *file_size:gt:4096&file_size:lt:40960&io_class:3* only uses the “**&**” operator.

Rules with more than one logical operator will lead to an unmatched evaluation (for example, *metadata|file_size:lt:4096&io_class:3* will lead to an undefined condition evaluation due to the use of both “**&**” and “**|**” operators).

The keyword **&done** can always be added at the end of any valid combined rule. Note that the classifier is unable to further categorize IO when it is in direct mode, therefore, IO classified as direct cannot jointly classify other conditions such as file_size or directory.

Open CAS Linux - Admin Guide

Advanced Options

Many-to-one Option

You can add multiple core devices to a single cache device. An example *opencas.conf* demonstrates:

```
## Caches configuration section

[caches]

## Cache ID Cache device Cache mode Extra fields (optional)

1 /dev/nvme0n1p5 wb ioclass_file=/etc/opencas/ioclass-config.csv

## Core devices configuration

[cores]

## Cache ID Core ID Core device

1 1 /dev/sda

1 2 /dev/sdb

1 3 /dev/sdc
```

```
1 4 /dev/sdd
```

This example shows 4 core devices paired with a single NVMe drive partition.

For each core device added, a cache-core pair will be created with the following naming format:

- */dev/cas1-1* will be created associated with *<cache_id> =1*, first core device. (For example: */dev/sda*)
- */dev/cas1-2* will be created associated with *<cache_id>=1*, second core device.
  (For example: */dev/sdb*)

Display cache listing via `casadm -L`:

```
# casadm -L type id disk status write policy device

cache 1 /dev/nvme0n1p5 Running wb -
├core 1 /dev/sda Active - /dev/cas1-1
├core 2 /dev/sdb Active - /dev/cas1-2
├core 3 /dev/sdc Active - /dev/cas1-3
└core 4 /dev/sdd Active - /dev/cas1-4
```

Multi-Level Caching

Open CAS Linux supports multi-level caching. For example, this enables the user to cache warm data from slow HDD media to faster SSD media, and then cache hot data from the SSD to an even faster media such as a RAMdisk or Intel(R) Optane(TM) SSDs. In this case, a fixed portion of DRAM will be allocated for buffering, and the SSD will remain as a fully inclusive cache so DRAM cache data will always be available on the SSD.

To set up a RAMdisk-based cache level, do the following:

- Create a RAMdisk (100 MB or larger).
- In RHEL, for example, the default RAMdisk size is 8 MB so it must be configured to be equal or greater than 40 MB.
- For kernels 3.0 or newer, issue the following command to increase the RAMdisk size to 100 MB:

```
# modprobe brd rd_size=102400
```

- Do not format or create a file system or mount the RAMdisk, at this time.
- Do not use */dev/ram0* since it is reserved by the kernel. Use any available RAMdisk */dev/ram1* or greater. We will use */dev/ram1* in this example.

- Reboot the system and start a fresh Open CAS Linux install, if necessary.
- Create the SSD cache instance, where */dev/sdc* is your SSD cache device:

```
# casadm -S -d /dev/sdc
```

- Add a core device (*/dev/sdb*) mapping to the SSD cache instance:

```
# casadm -A -i 1 -d /dev/sdb
```

- Create the RAMdisk cache instance:

```
# casadm -S -d /dev/ram1 -i 2
```

- Add the tiered core device (*/dev/cas1-1*) mapping to the RAMdrive cache instance:

```
# casadm -A -i 2 -d /dev/cas1-1
```

- Create the file system from the newly created tiered cache drive:

```
# mkfs -b 4096 -t ext3 /dev/cas2-1
```
or

```
# mkfs.ext4 -b 4096 /dev/cas2-1
```
or

```
# mkfs.xfs -f -i size=2048 -b size=4096 -s size=4096
/dev/cas2-1
```

- Create a cache directory and mount the cache to it:

```
# mkdir -p /mnt/cache1
# mount /dev/cas2-1 /mnt/cache1
```
The newly created and mounted cache drive */mnt/cache1* is ready for use.

By default, any RAMdisk tiered buffering setup is volatile and will need to be set up again following a system restart. Any data should remain in the SSD following a clean restart. See section **Stopping Cache Instances** for details on stopping Open CAS Linux.

Caution: If the RAMdisk cache tier is started in write-back mode and there is a dirty shutdown, data loss may occur.

Linux* LVM support

Open CAS Linux supports LVM in two ways.

1. LVM physical volumes can be created directly on Open CAS Linux devices. This allows creation of volume groups thereon, and later creation of logical volumes on those groups.

   With older versions of LVM, you must first create a partition on the core device (eg. if the core device is /dev/sdc, you must create /dev/sdc1) prior to accelerating that device with Open CAS Linux device and creating the physical volume or creation of the logical volume will fail. If you see the following warning: *“WARNING: Ignoring duplicate config node: types (seeking types); Found duplicate PV”*, then you must use this workaround.

2. LVM logical volumes can be used as core devices, just like regular partitions.

LVM must be configured in the system before using it with Open CAS Linux, which is outside of the scope of this document.

Ensure Open CAS Linux devices are listed as acceptable block device types in */etc/lvm/lvm.conf* by adding the following line to the advanced settings:

```
# Advanced settings.
# List of pairs of additional acceptable block device types found
# in /proc/devices with maximum (non-zero) number of partitions.
types = [ “cas”, 16 ]
```

After the LVM is configured and an Open CAS Linux device has been created (see **Configuring CAS** for further details) a physical volume can be created thereon:

```
# pvcreate /dev/cas1-1
```

or if creating the physical volume from a partition on the Open CAS Linux device

```
#pvcreate /dev/cas1-1p1
```

Additionally, after the Open CAS Linux PV volume has been created it is imperative the underlying core device is excluded for use by LVM. Excluding the device will avoid LVM warnings about duplicate metadata between the exported Open CAS Linux drive and the core drive. It will also ensure Open CAS Linux starts cleanly after system reboots by preventing LVM from opening the core drive when it has been already opened exclusively by Open CAS Linux. To exclude the core drive from LVM populate the global_filter variable in the */etc/lvm/lvm.conf* file with the core drive’s unique storage identifier (WWN). For example, to exclude the core device with unique identifer *wwn-0x5000c50066d547a9-part1* add the following lines to */etc/lvm/lvm.conf* :

```
# Exclude the core device for use by LVM using a global filter.
# Use the full WWN path in the format [“r|WWN|”]
global_filter = [ “r|/dev/disk/by-id/wwn-0x5000c50066d547a9-part1|” ]
```

After the physical volume is created, the user can create a volume group, for example: *vg_cas*, and then a logical volume on it, for example: *lv_cas*:

```
# vgcreate vg_cas /dev/cas1-1
# lvcreate -n lv_cas -l 100%FREE vg_cas
```

Create and mount LVM filesystem to “*vfs_cas*”:

```
# mkfs.ext4 <-b 4096> /dev/vg_cas/lv_cas
# mkdir -p /mnt/vfs_cas
# mount /dev/vg_cas/lv_cas /mnt/vfs_cas
```

To remove the LVM device, use the following steps:

```
# umount /mnt/vfs_cas
# vgchange -an vg_cas
# casadm -R -i <cache_id> -j <core_id>
# casadm -T -i 1
```

Dynamic logical volume reduction or extension is not automatically supported (e.g. vgreduce or vgextend). The user must remove the Open CAS Linux device (e.g. *cas1-1*) as the physical volume, perform the necessary size changes and then recreate the physical volume based on the Open CAS Linux device. Consult LVM man pages and web wikis for details on how to use and manage LVMs.

Trim Support

The trim command (known as TRIM in the ATA command set, and UNMAP in the SCSI command set) allows an operating system to inform an SSD which blocks of data are no longer considered in use and can be wiped internally. TRIM is enabled by default and supported by Open CAS Linux for Intel SSDs. When a user or application deletes data, Open CAS Linux will free the cache lines associated with that data when TRIM is used. This avoids the time required to flush that data to the backend storage such as hard drives.

In order to take advantage of this feature, the Linux file system has to be configured so TRIM requests are properly sent to Open CAS Linux for processing. The configuration consist of mounting the file system with the discard option, which applies to most file systems, such as ext4 or xfs.

The following command is an example:

```
# mount -o discard /dev/cas1-1 /mnt
```

Kernel Module Parameters

Following is a list of available system and kernel parameters that can be modified to further improve the performance of certain applications. These parameters should not be modified without advanced tuning expertise and a thorough knowledge of the workload. Please contact the Open CAS community for further details or support.

The parameters are: unaligned_io, metadata_layout, and use_io_scheduler.

- **unaligned_io:** This parameter enables user to specify how IO requests unaligned to 4KiB should be handled. This parameter should be changed if caching such requests cause performance drop.
  - Possible values: 0 - handle unaligned requests using PT mode, or 1 - handle unaligned requests using current cache mode (default)
- **metadata_layout:** This parameter enables user to specify layout of metadata on SSD. This is an advanced parameter for **controlling** Open CAS Linux internal metadata layout.
  - Possible values: 0 - striping (default), or 1 - sequential
- **use_io_scheduler:** This parameter enables user to specify how IO requests are handled - if IO scheduler will be used or not. **User** can disable IO scheduler on Open CAS Linux device if there is another block device with its own scheduler on top of it (for example LVM). This parameter has no effect for write-back mode.
  - Possible values: 0 - handle IO requests in application context, thus omit IO scheduler, or 1 - handle IO using IO scheduler (default)

The parameters can only be changed during module loading and must be added to the system startup so they would take effect upon reboots. To change the values of these parameters without modifying Open CAS Linux startup files, you can create a configuration file in */etc/modprobe.d* directory. For example, on a RHEL 7.3 system, create the file */etc/modprobe.d/cas_cache.conf* with the following content, then restart Open CAS Linux or reboot the system:

```
# cat /etc/modprobe.d/cas_cache.conf
```

```
options cas_cache unaligned_io=0 seq_cut_off_mb=10
```

NOTE: The example above may not apply to your specific Operating System. Please see the appropriate documents for your operating system for properly modifying these parameters.

Performance Options In some instances, like sequential read on a newly created cache device, the cache device may become overloaded, causing slower performance, and the queue may increase, consuming system buffers. To avoid this issue, it may be necessary to manually force an optimal queue size for the cache write queue specific to a use case. The default max_writeback queue_size value is 65,536. Another parameter may also be used to determine the unblock threshold for this queue: writeback_queue_unblock_size (default value is set at 60,000). You can use the following command to overwrite this default value and set a custom max_writeback queue_size or alter the writeback_queue_unblock_size threshold:

```
# modprobe cas_cache max_writeback_queue_size=<size>
writeback_queue_unblock_size=<size>
```

or

```
# insmod cas_cache max_writeback_queue_size=<size>
writeback_queue_unblock_size=<size>
```

The above examples are each a single line.

Keep in mind that the modinfo command only displays the default values of these parameters. To check the actual values of the parameters that you have modified, you can do so by examining /sys/modules/cas_cache/parameters/* files:

```
/sys/module/cas_cache/parameters/max_writeback_queue_size
/sys/module/cas_cache/parameters/writeback_queue_unblock_
size.
```

If you are not seeing the correct values in the sysfs files, it is most likely because you have not unloaded the Open CAS Linux module prior to running modprobe. The modprobe command does not return any errors when you try to run it against a module that is already loaded.

Writeback Throttling
One can disable WB throttling for a block device via sysfs, similarly to how IO scheduler is changed:

```
echo 0 > /sys/block//queue/wbt_lat_usec
```

It is recommended to apply this setting to all core/cache devices on all systems with WB throttling functionality and kernel version below 4.19. This feature was introduced in kernel 4.10, however it is possible that distributions would port it to older kernels. Thus the best approach is to disable WB throttling on susceptible kernels whenever the sysfs file exists.

If CAS core/cache device is not dedicated for CAS exclusive use (e.g. partitioned drive used as CAS cache and journal) then disabling WB throttling might affect quality of service for the I/O concurrent to CAS. In this case user needs to either find a compromise or move to a kernel 4.19 or newer.

Open CAS Linux - Admin Guide

Monitoring Open CAS Linux

There are a number of performance counters available for viewing. These counters are accessible using the casadm -P -i <cache_id> command. This section contains a brief description of data included in the *stats* command line option.

```
# casadm –stats –cache-id <ID>
```

or

```
# casadm -P -i <ID>
```

See the '-P | –stats' section for details.

The output of this command contains five tables that describe the activities in the caching system:

- **Usage statistics**
- **Inactive Usage statistics**
- **Request statistics**

- **Block statistics**
- **Error statistics**

Entries are in the form: stat, actual number, percent of overall, units.

You can use these statistics to learn about your data usage. For example, looking at the sequential versus random read statistics reveals how your data is used.

Additionally, the output provides a section detailing a summary of the cache instance as well as configuration information. This section is at the beginning of the output and it includes:

Configuration Information

Configuration	Description
Cache ID	Number representing the ID of the cache instance
Cache Size	The physical cache size in 4 KiB blocks and/or GiB blocks
Cache Device	The name of the cache device
Core Devices	The number of backend storage core devices attached to the cache instance
Inactive Core Devices	Number of core devices which are currently inactive
Promotion Policy	The policy for the promotion of cache lines of this cache instance
Write Policy	The write policy of this cache instance (write-back, write-through, etc.). See **Configuration Details** and **Configuring CAS** for additional information
Eviction Policy	The eviction policy of this cache instance (LRU)

Configuration	Description
Cleaning Policy	The cleaning policy of this cache instance (ALRU, ACP, or NOP). See **Configuration Details** for additional information
Cache Line Size	The block size of a cache line. This parameter can be set during the creation of the cache instance but cannot be modified afterwards
Metadata Memory Footprint	The amount of physical memory cache metadata is occupying
Dirty For	The length of time in seconds that the cache has held dirty data
Metadata Mode	The current operational mode for metadata. Under regular operation this should state *normal*
Status	The current status of the cache instance (Running, Flushing, etc.). Under regular operation this should state *Running*

The following tables list the statistics (counters) that Open CAS Linux records:

Usage Statistics

Statistic	Description
Occupancy	Number of cached blocks

Statistic	Description
Free	Number of empty blocks in the cache
Clean	Number of clean blocks (cache data matches core data)
Dirty	Number of dirty blocks (block written to cache but not yet synced to core)

Inactive Usage Statistics (printed only in special cases, such as when a core is detached from cache)

Statistic	Description
Inactive Occupancy [4KiB Blocks]	Number of inactive cached blocks
Inactive Clean [4KiB Blocks]	Number of inactive clean blocks (cache data matches core data)
Inactive Dirty [4KiB Blocks]	Number of inactive dirty blocks (block written to cache but not yet synced to core)

Request Statistics

Statistic	Description
Read hits	Number of reads that were cache hits
Read partial misses	Number of reads that spanned both data that was in the cache and data that was not in the cache

Statistic	Description
Read full misses	Number of reads that were cache misses
Read total	Total number of reads
Write hits	Number of writes that were cache hits
Write partial misses	Number of writes that spanned both data that was in the cache and data that was not in the cache
Write full misses	Number of writes that were cache misses
Write total	Total number of writes
Pass-through reads	Number of read requests sent directly to the core device (not cached by Open CAS Linux). This statistic is incremented when the request is handled while the cache is in pass-through mode (see -Q | –set-cache-mode for details).
Pass-through writes	Number of write requests sent directly to the core device (not cached by Open CAS Linux). This statistic is incremented when the request is handled while the cache is in pass-through mode (see -Q | –set-cache-mode for details).
Serviced requests	Total number of IO requests serviced (that did not bypass the cache)

Statistic	Description
Total requests	Total number of IO requests (both serviced and bypassed requests)

Percentages are calculated as percentage of total requests. (e.g. Read Hits % = 100*(# read hits / total requests)).

Block Statistics

Statistic	Description
Reads from core(s)	Number of blocks read from core device(s)
Writes to core(s)	Number of blocks written to core device(s)
Total to/from core(s)	Total number of blocks read from or written to core device(s)
Reads from cache	Number of blocks read from cache device
Writes to cache	Number of blocks written to cache device
Total to/from cache	Total number of blocks read from or written to cache device
Reads from exported object(s)	Number of blocks read from exported object(s) (eg. cas1-1)
Writes to exported object(s)	Number of blocks written to exported object(s)
Total to/from exported object(s)	Total number of blocks read from or written to exported object(s)

Error Statistics

Statistic	Description
Cache read errors	Number of read errors to the cache device
Cache write errors	Number of write errors to the cache device
Cache total errors	Total number of read or write errors to the cache device
Core read errors	Number of read errors to the core device(s)
Core write errors	Number of write errors to the core device(s)
Core total errors	Total number of read or write errors to the core device(s)
Total errors	Total number of read or write errors to the cache or core devices

Viewing Cache Statistics

Usage example for cache-level statistics:

```
# casadm -P -i 1
```

Watch statistics updates real time:

```
# watch -d casadm -P -i 1
```

Returned output:

```
Cache Id                1
Cache Size               12055031 [4KiB Blocks] / 45.99 [GiB]

Cache Device             /dev/nvme0n1p1

Core Devices             3

Inactive Core Devices    0

Write Policy             wt

Eviction Policy          lru

Cleaning Policy          alru

Cache line size          4 [KiB]

Metadata Memory Footprint 639.1 [MiB]

Dirty for                0 [s] / Cache clean

Metadata Mode            normal

Status                   Running

╔══════════════════╤══════════╤═══════╤═════════════╗

║ Usage statistics │  Count   │   %   │   Units     ║

╠══════════════════╪══════════╪═══════╪═════════════╣

║ Occupancy        │ 12055001 │ 100.0 │ 4KiB blocks ║

║ Free             │       30 │   0.0 │ 4KiB blocks ║

║ Clean            │ 12055001 │ 100.0 │ 4KiB blocks ║

║ Dirty            │        0 │   0.0 │ 4KiB blocks ║

╚══════════════════╧══════════╧═══════╧═════════════╝
```

```
╔══════════════════════╤═══════════╤═══════╤══════════╗
║ Request statistics   │   Count   │   %   │ Units    ║
╠══════════════════════╪═══════════╪═══════╪══════════╣
║ Read hits            │  50532296 │  32.8 │ Requests ║
║ Read partial misses  │    202850 │   0.1 │ Requests ║
║ Read full misses     │  48537517 │  31.5 │ Requests ║
║ Read total           │  99272663 │  64.4 │ Requests ║
╟──────────────────────┼───────────┼───────┼──────────╢
║ Write hits           │  31762847 │  20.6 │ Requests ║
║ Write partial misses │       436 │   0.0 │ Requests ║
║ Write full misses    │  16197432 │  10.5 │ Requests ║
║ Write total          │  47960715 │  31.1 │ Requests ║
╟──────────────────────┼───────────┼───────┼──────────╢
║ Pass-Through reads   │   2779876 │   1.8 │ Requests ║
║ Pass-Through writes  │   4027209 │   2.6 │ Requests ║
║ Serviced requests    │ 147233378 │  95.6 │ Requests ║
╟──────────────────────┼───────────┼───────┼──────────╢
║ Total requests       │ 154040463 │ 100.0 │ Requests ║
╚══════════════════════╧═══════════╧═══════╧══════════╝
```

```
╔══════════════════════════════════╤════════════╤═══════╤═════════════
╗
║ Block statistics                 │   Count    │   %   │   Units     ║
╠══════════════════════════════════╪════════════╪═══════╪═════════════
╣
║ Reads from core(s)               │ 471298381  │  57.4 │ 4KiB blocks ║
║ Writes to core(s)                │ 349624858  │  42.6 │ 4KiB blocks ║
║ Total to/from core(s)            │ 820923239  │ 100.0 │ 4KiB blocks ║
╟──────────────────────────────────┼────────────┼───────┼─────────────
╢
║ Reads from cache                 │ 140451806  │  16.5 │ 4KiB blocks ║
║ Writes to cache                  │ 713000302  │  83.5 │ 4KiB blocks ║
║ Total to/from cache              │ 853452108  │ 100.0 │ 4KiB blocks ║
╟──────────────────────────────────┼────────────┼───────┼─────────────
╢
║ Reads from exported object(s)    │ 611750187  │  63.6 │ 4KiB blocks ║
║ Writes to exported object(s)     │ 349624858  │  36.4 │ 4KiB blocks ║
║ Total to/from exported object(s) │ 961375045  │ 100.0 │ 4KiB blocks ║
╚══════════════════════════════════╧════════════╧═══════╧═════════════
╝

╔════════════════════╤═══════╤═════╤═════════╗
║ Error statistics   │ Count │  %  │ Units   ║
```

```
╟──────────────────────┼─────────┼───────┼────────────╢
║ Cache read errors  │     0 │ 0.0 │ Requests ║
║ Cache write errors │     0 │ 0.0 │ Requests ║
║ Cache total errors │     0 │ 0.0 │ Requests ║
╟──────────────────────┼─────────┼───────┼────────────╢
║ Core read errors   │     0 │ 0.0 │ Requests ║
║ Core write errors  │     0 │ 0.0 │ Requests ║
║ Core total errors  │     0 │ 0.0 │ Requests ║
╟──────────────────────┼─────────┼───────┼────────────╢
║ Total errors       │     0 │ 0.0 │ Requests ║
╚══════════════════════╧═════════╧═══════╧════════════╝
```

Usage example for core-level statistics:

```
# casadm -P -i 1 -j 1
```

Returned output:

```
Core Id                         1
Core Device                     /dev/sdd1
Exported Object                 /dev/cas1-1
Status                  Active
Seq cutoff threshold            1024 [KiB]
Seq cutoff policy               full
```

```
Core Size                       36620800 [4KiB Blocks] / 139.70 [GiB]

Dirty for                       0 [s] / Cache clean

╔══════════════════╤═════════╤═══════╤═════════════╗
║ Usage statistics │  Count  │   %   │   Units     ║
╠══════════════════╪═════════╪═══════╪═════════════╣
║ Occupancy        │ 4363950 │  36.2 │ 4KiB blocks ║
║ Free             │       1 │   0.0 │ 4KiB blocks ║
║ Clean            │ 4363950 │ 100.0 │ 4KiB blocks ║
║ Dirty            │       0 │   0.0 │ 4KiB blocks ║
╚══════════════════╧═════════╧═══════╧═════════════╝

╔══════════════════════╤══════════╤═══════╤══════════╗
║ Request statistics   │  Count   │   %   │ Units    ║
╠══════════════════════╪══════════╪═══════╪══════════╣
║ Read hits            │ 21075939 │  29.7 │ Requests ║
║ Read partial misses  │    77109 │   0.1 │ Requests ║
║ Read full misses     │ 18448305 │  26.0 │ Requests ║
║ Read total           │ 39601353 │  55.9 │ Requests ║
╟──────────────────────┼──────────┼───────┼──────────╢
║ Write hits           │ 17240919 │  24.3 │ Requests ║
║ Write partial misses │      105 │   0.0 │ Requests ║
║ Write full misses    │  9972336 │  14.1 │ Requests ║
║ Write total          │ 27213360 │  38.4 │ Requests ║
╟──────────────────────┼──────────┼───────┼──────────╢
║ Pass-Through reads   │   983305 │   1.4 │ Requests ║
║ Pass-Through writes  │  3053282 │   4.3 │ Requests ║
║ Serviced requests    │ 66814713 │  94.3 │ Requests ║
╟──────────────────────┼──────────┼───────┼──────────╢
║ Total requests       │ 70851300 │ 100.0 │ Requests ║
╚══════════════════════╧══════════╧═══════╧══════════╝

╔══════════════════════════════╤═══════════╤═══════╤═════════════╗
║ Block statistics             │   Count   │   %   │   Units     ║
╠══════════════════════════════╪═══════════╪═══════╪═════════════╣
║ Reads from core              │ 156393665 │  57.0 │ 4KiB blocks ║
║ Writes to core               │ 118061392 │  43.0 │ 4KiB blocks ║
║ Total to/from core           │ 274455057 │ 100.0 │ 4KiB blocks ║
╟──────────────────────────────┼───────────┼───────┼─────────────╢
║ Reads from cache             │  49888641 │  17.7 │ 4KiB blocks ║
║ Writes to cache              │ 231251909 │  82.3 │ 4KiB blocks ║
║ Total to/from cache          │ 281140550 │ 100.0 │ 4KiB blocks ║
╟──────────────────────────────┼───────────┼───────┼─────────────╢
```

```
║ Reads from exported object    │ 206282306 │  63.6 │ 4KiB blocks ║

║ Writes to exported object     │ 118061392 │  36.4 │ 4KiB blocks ║

║ Total to/from exported object │ 324343698 │ 100.0 │ 4KiB blocks ║

╚═══════════════════════════════╧═══════════╧═══════╧═════════════╝

╔════════════════════╤═══════╤═════╤══════════╗

║ Error statistics   │ Count │  %  │ Units    ║

╠════════════════════╪═══════╪═════╪══════════╣

║ Cache read errors  │     0 │ 0.0 │ Requests ║

║ Cache write errors │     0 │ 0.0 │ Requests ║

║ Cache total errors │     0 │ 0.0 │ Requests ║

╟────────────────────┼───────┼─────┼──────────╢

║ Core read errors   │     0 │ 0.0 │ Requests ║

║ Core write errors  │     0 │ 0.0 │ Requests ║

║ Core total errors  │     0 │ 0.0 │ Requests ║

╟────────────────────┼───────┼─────┼──────────╢

║ Total errors       │     0 │ 0.0 │ Requests ║

╚════════════════════╧═══════╧═════╧══════════╝
```

Usage example for IO class-level statistics:

```
# casadm -P -i 1 -j 1 -d
```

Returned output:

```
IO class ID         1

IO class name       Metadata

Eviction priority   0

Selective allocation Yes

╔══════════════════╤════════╤═══════╤═════════════╗
║ Usage statistics │ Count  │   %   │   Units     ║
╠══════════════════╪════════╪═══════╪═════════════╣
║ Occupancy        │ 864660 │   7.2 │ 4KiB blocks ║
║ Free             │      0 │   0.0 │ 4KiB blocks ║
║ Clean            │ 864660 │ 100.0 │ 4KiB blocks ║
║ Dirty            │      0 │   0.0 │ 4KiB blocks ║
╚══════════════════╧════════╧═══════╧═════════════╝

╔══════════════════════╤═════════╤══════╤══════════╗
║ Request statistics   │  Count  │  %   │ Units    ║
╠══════════════════════╪═════════╪══════╪══════════╣
║ Read hits            │ 8097835 │ 11.4 │ Requests ║
║ Read partial misses  │       0 │  0.0 │ Requests ║
║ Read full misses     │      17 │  0.0 │ Requests ║
║ Read total           │ 8097852 │ 11.4 │ Requests ║
╟──────────────────────┼─────────┼──────┼──────────╢
```

```
║ Write hits           │  414587 │  0.6 │ Requests ║
║ Write partial misses │     104 │  0.0 │ Requests ║
║ Write full misses    │  273056 │  0.4 │ Requests ║
║ Write total          │  687747 │  1.0 │ Requests ║
╟──────────────────────┼─────────┼──────┼──────────╢
║ Pass-Through reads   │       2 │  0.0 │ Requests ║
║ Pass-Through writes  │    5517 │  0.0 │ Requests ║
║ Serviced requests    │ 8785599 │ 12.4 │ Requests ║
╟──────────────────────┼─────────┼──────┼──────────╢
║ Total requests       │ 8791118 │ 12.4 │ Requests ║
╚══════════════════════╧═════════╧══════╧══════════╝

╔══════════════════╤═════════╤═════╤═════════════╗
║ Block statistics │  Count  │  %  │   Units     ║
╠══════════════════╪═════════╪═════╪═════════════╣
║ Blocks reads     │ 8097854 │ 3.9 │ 4KiB blocks ║
╟──────────────────┼─────────┼─────┼─────────────╢
║ Blocks writes    │ 9007260 │ 7.6 │ 4KiB blocks ║
╚══════════════════╧═════════╧═════╧═════════════╝
```

This command will output the above format for each defined IO class.

Usage example for IO class-level statistics output in csv format and saved to file

```
# casadm -P -i 1 -j 1 -d -o csv > stats.txt
```

Returned output:

```
IO class ID,IO class name,Eviction priority,Selective allocation,Occupancy [4KiB blocks],Occupancy [%],Free [4KiB blocks],Free [%],Clean [4KiB blocks],Clean [%],Dirty [4KiB blocks],Dirty [%],Read hits [Requests],Read hits [%],Read partial misses [Requests],Read partial misses [%],Read full misses [Requests],Read full misses [%],Read total [Requests],Read total [%],Write hits [Requests],Write hits [%],Write partial misses [Requests],Write partial misses [%],Write full misses [Requests],Write full misses [%],Write total [Requests],Write total [%],Pass-Through reads [Requests],Pass-Through reads [%],Pass-Through writes [Requests],Pass-Through writes [%],Serviced requests [Requests],Serviced requests [%],Total requests [Requests],Total requests [%],Blocks reads [4KiB blocks],Blocks reads [%],Blocks writes [4KiB blocks],Blocks writes [%]
0,Unclassified,22,Yes,0,0.0,97,100.0,0,0.0,0,0.0,0,0.0,0,0.0,82,0.0,82,0.0,0,0.0,0,0.0,0,0.0,0,0.0,1,0.0,0,0.0,82,0.0,83,0.0,83,0.0,0,0.0
1,Metadata,0,Yes,864660,7.2,0,0.0,864660,100.0,0,0.0,8068949,11.4,0,0.0,17,0.0,8068966,11.4,413037,0.6,104,0.0,273056,0.4,686197,1.0,2,0.0,5517,0.0,8755163,12.4,8760682,12.4,8068968,3.9,8979669,7.6
```

This command will output the above format for each defined IO class.

Any of the previously referenced examples can be output to csv format as well.

Resetting the Performance Counters

The performance counters are automatically reset every time the cache is started. To manually clear the performance counters, use the casadm -Z -i <cache_id> -j <core_id> command line option.

Open CAS Linux - Admin Guide

Upgrading Open CAS Linux version

Open CAS Linux doesn’t support a smooth version upgrade. To perform an upgrade, all dirty data has to be flushed to core devices and all running cache instances have to be stopped. To perform both this actions at once, the following command can be used

```
# casctl stop –flush
```

To allow restoring an existing configuration after an upgrade, the /etc/opencas/opencas.conf file has to be preserved.

After preserving the config file and stopping all the running cache instances, the old version of Open CAS Linux must be removed.

Installing a new version of Open CAS Linux is covered in **Installing CAS**

After the new version of Open CAS Linux is installed, the old config can be restored using the preserved config file with

```
# casctl init –force
```

Open CAS Linux - Admin Guide

- -S | --start-cache
- -T | --stop-cache
- -Q | --set-cache-mode
- -A | --add-core

- -R | --remove-core
- --remove-detached
- -L | --list-caches
- -P | --stats
- -Z | --reset-counters
- -F | --flush-cache
- -E | --flush-core
- -H | --help
- -V | --version
- -C | --io-class
  - -C | --load-config
  - -L | --list
- -X | --set-param
  - seq-cutoff
  - cleaning
  - cleaning-acp
  - cleaning-alru
  - promotion
- -G | --get-param
  - seq-cutoff
  - cleaning
  - cleaning-acp
  - cleaning-alru
  - promotion

Configuration Tool Details

The Open CAS Linux product includes a user-level configuration tool that provides complete control of the caching software. The commands and parameters available with this tool are detailed in this chapter.

To access help from the CLI, type the *-H* or *--help* parameter for details. You can also view the man page for this product by entering the following command:

```
# man casadm
```

-S | --start-cache

Usage: casadm --start-cache --cache-device <DEVICE> [option…]

Example:

```
# casadm --start-cache --cache-device /dev/sdc
```

or

```
# casadm -S -d /dev/sdc
```

Description: Prepares a block device to be used as device for caching other block devices. Typically the cache devices are SSDs or other NVM block devices or RAM disks. The process starts a framework for device mappings pertaining to a specific cache ID. The cache can be loaded with an old state when using the *-l* or *–load* parameter (previous cache metadata will not be marked as invalid) or with a new state as the default (previous cache metadata will be marked as invalid).

Required Parameters:

[-d, --cache-device <DEVICE>] : Caching device to be used. This is an SSD or any NVM block device or RAM disk shown in the */dev* directory. <device> needs to be the complete path describing the caching device to be used, for example /dev/sdc.

Optional Parameters:

[-i, --cache-id <ID>]: Cache ID to create; <1 to 16384>. The ID may be specified or by default the command will use the lowest available number first.

[-l, --load]: Load existing cache metadata from caching device. If the cache device has been used previously and then disabled (like in a reboot) and it is determined that the data in the core device has not changed since the cache device was used, this option will allow continuing the use of the data in the cache device without the need to re-warm the cache with data.

- *Caution:* You must ensure that the last shutdown followed the instructions in section **Stopping Cache Instances**. If there was any change in the core data prior to enabling the cache, data would be not synced correctly and will be corrupted.

[-f, --force]: Forces creation of a cache even if a file system exists on the cache device. This is typically used for devices that have been previously utilized as a cache device.

- *Caution:* This will delete the file system and any existing data on the cache device.

[-c, --cache-mode <NAME>]: Sets the cache mode for a cache instance the first time it is started or created. The mode can be one of the following:

wt: (default mode) Turns write-through mode on. When using this parameter, the write-through feature is enabled which allows the acceleration of only read intensive operations.

wb: Turns write-back mode on. When using this parameter, the write-back feature is enabled which allows the acceleration of both read and write intensive operations.

- *Caution:* A failure of the cache device may lead to the loss of data that has not yet been flushed to the core device.

wa: Turns write-around mode on. When using this parameter, the write-around feature is enabled which allows the acceleration of reads only. All write locations that do not already exist in the cache (i.e. the locations have not be read yet or have been evicted), are written directly to the core drive bypassing the cache. If the location being written already exists in cache, then both the cache and the core drive will be updated.

pt: Starts cache in pass-through mode. Caching is effectively disabled in this mode. This allows the user to associate all their desired core devices to be cached prior to actually enabling caching. Once the core devices are associated, the user would dynamically switch to their desired caching mode (see '-Q | --set-cache-mode' for details).

wo: Turns write-only mode on. When using this parameter, the write-only feature is enabled which allows the acceleration of write intensive operations primarily.

- *Caution:* A failure of the cache device may lead to the loss of data that has not yet been flushed to the core device.

[-x, --cache-line-size <SIZE>]: Set cache line size {4 (default), 8, 16, 32, 64}. The cache line size can only be set when starting the cache and cannot be changed after cache is started.

-T | --stop-cache

Usage: casadm --stop-cache --cache-id <ID> [option...]

Example:

```
# casadm --stop-cache --cache-id 1
```
or

```
# casadm -T -i 1
```

Description: Stops all cache-core pairs associated with the cache device.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional parameters:

[-n, --no-data-flush]: Do not flush dirty data on exit (UNSAFE). This parameter will not allow the flushing of dirty data from the cache device to the core device upon the stopping of the cache. This will significantly reduce the time needed to stop the cache to allow for activities such as a fast reboot. The core device should not be used until the cache is started again with the --load parameter. Then the Open CAS Linux device can be used as normal.

- *Caution:* Data on the core device will not be complete or in sync with the cache device upon stopping the device. If the core device is used without starting Open CAS Linux cache it will lead to data corruption or data loss.
- The user may interrupt the blocking --stop-cache operation by pressing CTRL-C. When dirty data exists, interrupting the operation prior to the cache being fully stopped will result in the cache continuing to run. If the desire is to stop the cache without flushing the dirty data, use the --no-data-flush command.

-Q | --set-cache-mode

Usage: casadm --set-cache-mode --cache-mode <NAME> --cache-id <ID> –flush-cache <yes/no>

Example:

```
# casadm --set-cache-mode --cache-mode wb --cache-id 1
--flush-cache yes
```

or

```
# casadm -Q -c wb -i 1 -f yes
```

Description: Allows users to dynamically change cache modes while the cache is running.

Required Parameters:

[-c, --cache-mode <NAME>]:

- **wt -** switch from the current cache mode to write-through mode.
- **wb -** switch from the current cache mode to write-back mode.

- **wo -** switch from the current cache mode to write-only mode.
- **wa -** switch from the current cache mode to write-around mode
- **pt -** switch from the current cache mode to pass-through mode.
    - Dynamically switching to pass-through mode is useful in preventing cache pollution when the system is undergoing maintenance operations, for example.

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>

[-f, --flush-cache]: (required only when switching from write-back mode)

Caution: You should carefully consider the following choices.

- **yes** - Flush cache contents immediately to core drive before switching to new cache mode.
    - When choosing *yes* to flush the cache immediately, the operation may take a long time to complete depending on number of dirty blocks. IO to the device will continue at reduced performance until flush completes.
- **no** - Begin transition to new cache mode immediately, but flush cache contents opportunistically.
    - When choosing *no*, IO to the device will continue at normal performance, but you must be aware that the cache will be in a transition state, and not yet in the newly chosen state until the cache is fully flushed. The transition to the new state will take longer than choosing the *yes* option. Current cache state and flush % can be checked using the casadm -L command.

-A | --add-core

Usage: casadm --add-core --cache-id <ID> --core-device <DEVICE> [option…]

Example:

```
# casadm --add-core --cache-id 1 --core-device /dev/sdb
```

or

```
# casadm -A -i 1 -d /dev/sdb
```

Description: Adds/maps a core device (either the full device or a partition) to the framework associated with a specified cache ID. This command can be repeated using the same *cache-id* number to map multiple cores to the same cache device.

Required Parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

[-d, --core-device <DEVICE>]: Location of the HDD storage/core device.
You must use the complete device path in the /dev directory, for example /dev/sdb.

Optional Parameters:

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>.

-R | --remove-core

Usage: casadm --remove-core --cache-id <ID> --core-id <ID> [option…]

Example:

```
# casadm --remove-core --cache-id 1 --core-id 1
```
or

```
# casadm -R -i 1 -j 1
```
Description: Deletes the cache/core device mapping, which is one way to disable caching of a device.

Required Parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>.
You can identify the assigned value for a particular core device using the casadm -L command.

- *Caution:* Before using casadm -R, stop all IO to the mapped core device, ensure it is not in use, and unmount it.
- You can interrupt the blocking --remove-core operation by pressing CTRL-C. When dirty data exists, interrupting the operation prior to the core being fully removed will result in the core continuing to be cached.
- Although legal core ID range starts with 0, Open CAS Linux engine would resort to assigning core ID value
  of 0 only if all other core IDs within cache instance are used. In other words the order of core assignment is as follows: 1, 2, 3, …, 4094, 4095, 0.

Optional parameters:

[-f, --force]: Do not flush dirty data while removing the core device.

--remove-detached

Usage: casadm --remove-detached --device <DEV_NAME>

Example:

```
# casadm --remove-detached --device /dev/sda
```
or

```
# casadm --remove-detached -d /dev/sda
```
Description: Removes a device from the core pool. A device is in the core pool when it’s listed in *opencas.conf* as a core in a configured cache instance, and this cache instance is not yet started (for example, missing the NVMe drive). This command does not currently have a short form.

Required Parameters:

-d | --device <DEV_NAME>

Where DEV_NAME is a device name from the core pool

-L | --list-caches

Usage: casadm --list-caches

Example:

```
# casadm --list-caches
```
or

```
# casadm -L
```
Description: Displays each cache instance with the following details:

- Flash/SSD cache ID used in the instance
- Storage device used in the instance
- Status of the instance
- Write Policy of the instance (write-through by default)

Also displays the associated core devices with the following details:

- Core Pool label
- Numeric ID and disk name
- Status
- Open CAS Linux exported device ID
- Placement within the core pool

Example output:

```
type id disk status write policy device
```

```
cache 1 /dev/nvme0n1p1 Incomplete wt -
+core 1 /dev/disk/by-id/wwn-0x500….-part1 Inactive - /dev/cas1-1
+core 2 /dev/sdc1 Active - /dev/cas1-2
```

-P | --stats

Usage: casadm --stats --cache-id <ID> [option…]

Example:

```
# casadm --stats --cache-id 1
```

or

```
# casadm -P -i 1
```

Description: Prints performance and status counters for a specific cache instance. The section **Viewing Cache Statistics** shows the detailed output.

Required Parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional Parameters:

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>. Display statistics for a specific core device.

[-d, --io-class-id <ID>]: Unique identifier for io class <0 to 23>. Display statistics for a specific IO class.

- <ID> is optional. When the --io-class-id parameter is specified without specifying an <ID>, statistics will be displayed for each individual IO class.

[-f, --filter <filter-spec>]: Comma separated list of filters (e.g., –filter conf, req). Filter statistics output to only the requested statistics.

- *all:* (default mode) Displays all available cache statistics.
- *conf:* Displays cache and core configuration information and dirty timestamp.
- *usage:* Displays statistics on occupancy, free, clean, and dirty.
- *req:* Displays IO request level statistics.
- *blk:* Displays block level statistics.
- *err:* Displays IO error statistics.

[-o, --output-format <format>]: Sets desired output format for statistics.

- *table:* (default mode) Displays a table of the statistics information.
- *csv:* Outputs a comma separated list of statistics information. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

Example output:

# casadm -P -i 1

```
Cache Id                 1

Cache Size               43708 [4KiB Blocks] / 0.17 [GiB]

Cache Device             /dev/nvme0n1p1

Core Devices             2

Inactive Core Devices    1

Write Policy             wt

Eviction Policy          lru

Cleaning Policy          alru

Cache line size          4 [KiB]

Metadata Memory Footprint 19.6 [MiB]

Dirty for                0 [s] / Cache clean

Metadata Mode            normal

Status                   Incomplete

+==================+=======+=======+=============
+

| Usage statistics | Count |   %   |   Units     |
+==================+=======+=======+=============
+

| Occupancy        |   164 |   0.4 | 4KiB blocks |

| Free             | 43544 |  99.6 | 4KiB blocks |

| Clean            |   164 | 100.0 | 4KiB blocks |

| Dirty            |     0 |   0.0 | 4KiB blocks |

+==================+=======+=======+=============
+

+===========================+=======+======+=====
========+

| Inactive usage statistics | Count |  %   |   Units     |

+===========================+=======+======+=====
========+

| Inactive Occupancy        |    82 |  0.2 | 4KiB blocks |

| Inactive Clean            |    82 | 50.0 | 4KiB blocks |

| Inactive Dirty            |     0 |  0.0 | 4KiB blocks |

+===========================+=======+======+=====
========+

+======================+=======+=======+=========
=+

| Request statistics   | Count |   %   | Units    |

+======================+=======+=======+=========
=+
| Read hits            |   328 |  99.7 | Requests |

| Read partial misses  |     0 |   0.0 | Requests |

| Read full misses     |     0 |   0.0 | Requests |

| Read total           |   328 |  99.7 | Requests |

+----------------------+-------+-------+----------+

| Write hits           |     0 |   0.0 | Requests |

| Write partial misses |     0 |   0.0 | Requests |

| Write full misses    |     0 |   0.0 | Requests |

| Write total          |     0 |   0.0 | Requests |

+----------------------+-------+-------+----------+

| Pass-Through reads   |     1 |   0.3 | Requests |

| Pass-Through writes  |     0 |   0.0 | Requests |

| Serviced requests    |   328 |  99.7 | Requests |

+----------------------+-------+-------+----------+

| Total requests       |   329 | 100.0 | Requests |

+======================+=======+=======+=========
=+

+====================================+=======+=====
==+=============+

| Block statistics                   | Count |   %   |   Units     |
+=======================================+=======+=====
==+=============+

| Reads from core(s)              |     1 | 100.0 | 4KiB blocks |

| Writes to core(s)               |     0 |   0.0 | 4KiB blocks |

| Total to/from core(s)           |     1 | 100.0 | 4KiB blocks |

+--------------------------------+-------+-------+-------------+

| Reads from cache                |    82 | 100.0 | 4KiB blocks |

| Writes to cache                 |     0 |   0.0 | 4KiB blocks |

| Total to/from cache             |    82 | 100.0 | 4KiB blocks |

+--------------------------------+-------+-------+-------------+

| Reads from exported object(s)    |    83 | 100.0 | 4KiB blocks |

| Writes to exported object(s)     |     0 |   0.0 | 4KiB blocks |

| Total to/from exported object(s) |    83 | 100.0 | 4KiB blocks |

+=======================================+=======+=====
==+=============+

+=====================+=======+=====+==========+

| Error statistics   | Count |  %  | Units    |

+=====================+=======+=====+==========+

| Cache read errors  |     0 | 0.0 | Requests |

| Cache write errors |     0 | 0.0 | Requests |

| Cache total errors |     0 | 0.0 | Requests |
+-------------------+-------+-----+----------+

| Core read errors   |     0 | 0.0 | Requests |

| Core write errors  |     0 | 0.0 | Requests |

| Core total errors  |     0 | 0.0 | Requests |

+-------------------+-------+-----+----------+

| Total errors       |     0 | 0.0 | Requests |

+====================+=======+=====+==========+
```

-Z | --reset-counters

Usage: casadm --reset-counters --cache-id <CACHE_ID> [–core-id <CORE_ID>]

Example:

```
# casadm --reset-counters --cache-id 1
```

or

```
# casadm -Z -i 1
```

Description: Resets performance and status counters for specific cache/core pairs.

Required Parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional Parameters:

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>. If a core(s) is not specified, statistic counters are reset for all cores in a specified cache instance.

-F | --flush-cache

Usage: casadm --flush-cache --cache-id <ID>

Example:

```
# casadm --flush-cache --cache-id 1
```

or

```
# casadm -F -i 1
```

Description: Flushes all dirty data from the cache device to all the associated core devices.

Required Parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

- You can interrupt the blocking --flush-cache operation by pressing CTRL-C. When dirty data exists, interrupting the operation prior to the cache being fully flushed will result in some dirty data remaining in the cache. The dirty data will be flushed opportunistically as normal. IO to the device will continue with reduced performance during cache flushing.

-E | --flush-core

Usage: casadm --flush-core --cache-id <ID> --core-id <ID>

Example:

```
# casadm --flush-core --cache-id 1 --core-id 2
```

or

```
# casadm -E -i 1 -j 2
```

Description: Flushes all dirty data from the specified cache device to the specified associated core device.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>.

- You can interrupt the blocking --flush-core operation by pressing CTRL-C. When dirty data exists, interrupting the operation prior to the cache being fully flushed will result in some dirty data remaining in the cache. The dirty data will be flushed opportunistically as normal. IO to the device will continue with reduced performance during cache flushing.

-H | --help

Usage: casadm --help or casadm --<command> --help

Examples:

```
# casadm --help
```

or

```
# casadm -H
# casadm --start-cache --help
```
or

```
# casadm -S -H
```
Description: Displays a list of casadm commands along with a brief description. Use this command to also get more information on specific commands.

-V | --version

Usage: casadm --version

Example:

```
# casadm --version
```
or

```
# casadm -V
```
Description: Reports the Open CAS Linux kernel module and command line utility version numbers.

Optional parameters:

[-o, --output-format <format>]: Sets desired output format of the IO class configuration.

- *table:* (default mode) Displays a table of the IO class configuration.
- *csv:* Outputs a comma separated list of the IO class configuration. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

-C | --io-class

-C | --load-config

Usage: casadm --io-class --load-config --cache-id <ID> --file <file_path>

Example:

```
# casadm --io-class --load-config --cache-id 1 --file ioclass-config.csv
```
or

```
# casadm -C -C -i 1 -f ioclass-config.csv
```
Description: Loads IOclass configuration settings for the selected cache.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

[-f,–file]: Specifies the IO class configuration csv file to load.

-L | --list

Usage: casadm --io-class --list --cache-id <ID> [option…]

Example:

```
# casadm --io-class --list --cache-id 1
```
or

```
# casadm -C -L -i 1
```
Description: Displays the current IO class configuration settings for the specified cache ID.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional parameters:

[-o, --output-format <format>]: Sets desired output format of the IO class configuration.

- *table:* (default mode) Displays a table of the IO class configuration.
- *csv:* Outputs a comma separated list of the IO class configuration. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

-X | --set-param

Description: Used in conjunction with caching parameters or namespaces to set cache policies. This command is targeted to add additional parameters in future releases. See applicable configuration details below.

seq-cutoff

Usage: casadm --set-param --name seq-cutoff --cache-id <CACHE_ID> [–core-id <CORE_ID>] [–policy <POLICY>] [–threshold <THRESHOLD>]

Example:

```
# casadm --set-param --name seq-cutoff --cache-id 1 --core-id 1 --policy always --threshold 4096
```

or

```
# casadm -X -n seq-cutoff -i 1 -j 1 -p always -t 4096
```

Description: Set a cutoff size in KiB to stop the caching of sequential IO reads or writes.

Required parameters:

NOTE: -p and -t parameters aren’t both required. At least one is required.

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

[-p, --policy <POLICY>]: Sequential cutoff policy to be implemented.

Seq-cutoff policies:

- *always* - sequential cutoff is always enabled regardless of cache occupancy; sequential data will not be cached at all after threshold is reached.
- *full* - sequential cutoff is enabled only when the cache is full.
- *never* - sequential cutoff is disabled and will not be triggered; sequential data is handled using current cache policy.

[-t, --threshold <THRESHOLD>]: a value from range 1-4194181 (inclusive). Threshold is expressed in KiB.

Optional parameters:

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>. If not specified, core parameter is set to all cores in a given cache.

cleaning

Usage: casadm --set-param --name cleaning --cache-id <CACHE_ID> --policy <POLICY>

Example:

```
# casadm --set-param --name cleaning --cache-id 1 --policy
acp
```
or

```
# casadm -X -n cleaning -i 1 -p acp
```
Description: This parameter specifies the flushing policy to be used for the applicable cache instance, thus customizing the behavior for flushing dirty data.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

[-p, --policy <POLICY>]: Flushing policy to be used.

Cleaning policies:

- *acp* - (Aggressive Cleaning Policy) Cleans dirty cache lines as fast as possible so as to maintain the highest possible bandwidth to the backend storage, typically HDD’s. ACP is intended to stabilize the timing of write-back mode data flushing and to sustain more consistent cache performance.
- *alru* - (Approximate Least Recently Used) A modified least recently used method that will flush dirty data periodically (default mode).
- *nop* - (No Operation) Disables cache flushing except as needed for cache line replacement.

cleaning-acp

Usage: casadm --set-param --name cleaning-acp --cache-id <CACHE_ID> [–wake-up <NUMBER>] [–flush-max-buffers <NUMBER>]

Example:

```
# casadm --set-param --name cleaning-acp --cache-id 1
--wake-up 20 --flush-max-buffers 50
```
or

```
# casadm -X -n cleaning-acp -i 1 -w 20 -b 50
```
Description: This parameter specifies the desired characteristics of the acp flushing policy to be used for the applicable cache instance.

Required parameters:

NOTE: -w and -b parameters aren’t both required. At least one is required.

[-w, --wake-up <NUM>]: Period of time between awakenings of flushing thread in milliseconds. MIN: 0, MAX: 10000 (inclusive), DEFAULT: 10.

[-b, --flush-max-buffers <NUM>]: Number of dirty cache blocks to be flushed in one cleaning cycle. MIN: 1, MAX: 10000 (inclusive), DEFAULT: 128.

cleaning-alru

Usage: casadm --set-param --name cleaning-alru --cache-id <CACHE_ID> [–wake-up <NUMBER>] [–staleness-time <NUMBER>] [–flush-max-buffers <NUMBER>] [–activity-threshold <NUMBER>]

Example:

```
# casadm --set-param --name cleaning-alru --cache-id 1
--wake-up 30 --staleness-time 120 --flush-max-buffers 50
--activity-threshold 5000
```

or

```
# casadm -X -n cleaning-alru -i 1 -w 30 -s 120 -b 50 -t 5000
```

Description: This parameter specifies the desired characteristics of the alru flushing policy to be used for the applicable cache instance.

Required parameters:

NOTE: -w, -s, -b, and -t parameters aren’t all required. At least one is required.

[-w, --wake-up <NUM>]: Period of time between awakenings of flushing thread in seconds. MIN: 0, MAX: 3600, DEFAULT: 20.

[-s, --staleness-time <NUM>]: Time that has to pass from the last write operation before a dirty cache block can be scheduled to be flushed in seconds. MIN: 1, MAX: 3600, DEFAULT: 120.

[-b, --flush-max-buffers <NUM>]: Number of dirty cache blocks to be flushed in one cleaning cycle. MIN: 1, MAX: 10000, DEFAULT: 100.

[-t, --activity-threshold <NUM>]: Cache idle time before flushing thread can start in milliseconds. MIN: 0, MAX: 1000000, DEFAULT: 10000.

promotion

Usage: casadm --set-param --name promotion --cache-id <CACHE_ID> --policy

Example:

```
# casadm --set-param --name promotion --cache-id 1 --policy always
```

or

```
# casadm -X -n promotion -i 1 -p always
```

Description: This parameter specifies the desired promotion policy of core lines

Required Parameters:

[-p, --policy <POLICY>]: The policy desired for promotion of core lines

Promotion policies:

- *always* - Core lines are attemped to be promoted each time they are accessed.
- *nhit* - Core lines are attemped to be promoted after the n-th access. This n-th access threshold can be set using one of these commands:

  casadm –-set-param --name promotion-nhit --cache-id <CACHE_ID> [-o, --trigger] <PERCENTAGE>
  Description: Percent of cache to be occupied before cache inserts will be filtered by the policy.

  casadm –-set-param –-name promotion-nhit --cache-id <CACHE_ID> [-t, --threshold] <NUMBER>
  Description: Number of core line accesses required for it to be inserted into cache. Valid values are from range <2-1000>.

-G | --get-param

Description: This command will retrieve a variety of cache/core parameters which are set via --set-param. See applicable configuration details below.

seq-cutoff

Usage: casadm --get-param --name seq-cutoff --cache-id <CACHE_ID> –core-id <CORE_ID> [–output-format <FORMAT>]

Example:

```
# casadm --get-param --name seq-cutoff --cache-id 1 --core-id 1 --output-format csv
```

or

```
# casadm -G -n seq-cutoff -i 1 -j 1 -o csv
```

Description: Allows the ability to obtain current values of seq-cutoff cache/core parameters which are set with --set-param. Parameters that are returned: Sequential Cutoff Threshold and Sequential Cutoff Policy.

Required Parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional Parameters:

[-j, --core-id <ID>]: Unique identifier for core <0 to 4095>. Display statistics for a specific core device. If not specified, core parameter is set to all cores in a given cache.

[-o, --output-format <format>]: Sets desired output format for statistics.

- *table:* (default mode) Displays a table of the statistics information.
- *csv:* Outputs a comma separated list of statistics information. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

cleaning

Usage: casadm --get-param --name cleaning --cache-id <CACHE_ID> [–output-format <FORMAT>]

Example:

```
# casadm --get-param --name cleaning --cache-id 1 --output-format csv
```

or

```
# casadm -G -n cleaning -i 1 -o csv
```

Description: Allows the ability to obtain current values of cleaning cache/core parameters which are set with --set-param. Parameters that are returned: Cleaning Policy Type.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional Parameters:

[-o, --output-format <format>]: Sets desired output format for statistics.

- *table:* (default mode) Displays a table of the statistics information.
- *csv:* Outputs a comma separated list of statistics information. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

cleaning-acp

Usage: casadm --get-param --name cleaning-acp --cache-id <CACHE_ID> [–output-format <FORMAT>]

Example:

```
# casadm --get-param --name cleaning-acp --cache-id 1
--output-format csv
```

or

```
# casadm -G -n cleaning-acp -i 1 -o csv
```

Description: Allows the ability to obtain current values of acp cache/core parameters which are set with --set-param. Parameters that are returned: Wake Up Time and Flush Max Buffers.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional Parameters:

[-o, --output-format <format>]: Sets desired output format for statistics.

- *table:* (default mode) Displays a table of the statistics information.
- *csv:* Outputs a comma separated list of statistics information. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

cleaning-alru

Usage: casadm --get-param --name cleaning-alru --cache-id <CACHE_ID> [–output-format <FORMAT>]

Example:

```
# casadm --get-param --name cleaning-alru --cache-id 1
--output-format csv
```

or

```
# casadm -G -n cleaning-alru -i 1 -o csv
```

Description: Allows the ability to obtain current values of alru cache/core parameters which are set with --set-param. Parameters that are returned: Wake Up Time, Flush Max Buffers, Stale Buffer Time, and Activity Threshold.

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Optional Parameters:

[-o, --output-format <format>]: Sets desired output format for statistics.

- *table:* (default mode) Displays a table of the statistics information.
- *csv:* Outputs a comma separated list of statistics information. This output can be piped to a file and easily parsed or opened in a spreadsheet editor.

promotion

Usage: casadm –-get-param –-name promotion –-cache-id <CACHE_ID>

Example:

```
# casadm --get-param --name promotion --cache-id 1 --output-
format csv
```

or

```
# casadm –G –n promotion –i 1
```

Description: Retrieves the promotion policy

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

The additional command below can be used to obtain the nhit promotion values for the *nhit* policy

Usage: casadm –-get-param –-name promotion-nhit –-cache-id <CACHE_ID>

Example:

```
# casadm --get-param --name promotion-nhit --cache-id 1 --output-format csv
```
or

```
# casadm –G –n promotion-nhit –i 1
```
Description: Retrieves the nhit promotion policy values

Required parameters:

[-i, --cache-id <ID>]: Unique identifier for cache <1 to 16384>.

Open CAS Linux - Admin Guide

Terminology

Terms and Definitions

Term	Definition
cache	The transparent storage of data so that future requests for that data can be served faster.
cache instance	A single occurrance of the caching engine mapping a core device and a cache device.
cache hit	When requested data is contained in (and returned

Term	Definition
	from) the cache.
cache miss	When requested data is not in the cache, and therefore must be retrieved from its primary storage location.
core device	The device to be cached.
cache device	The device caching for a core device.
dirty data	This refers to data that is modified within the cache but not modified in primary storage.
IO	Abbreviation for input/output as it relates to the flow of data.
lazy write	The process of mirroring to primary storage.
pass-through	A caching mode in which the cache will be bypassed for all operations.
primary storage	As it relates to caching, the storage system or location (DAS, SAN, NAS, etc.) where the data is stored.
SAN	Storage Area Network. Framework used to attach remote computer storage devices to servers. Storage devices appear as if they were attached locally to the operating system.

Term	Definition
SSD	Solid State Drive. A device used for data storage that uses memory chips instead of a revolving disk.
tiered storage	A data storage technique that moves data between two or more kinds of storage, which are differentiated by four primary attributes: price, performance, capacity, and function.
write-around (wa)	A caching mode in which some write operations are not cached. Writes to blocks that do not exist in cache are written directly to the core device, bypassing the cache. If a write operation is issued to a block that is already in cache (because of a previous read operation), then writes are sent to both the core device the cache device. Write-around cache improves performance of workloads where write operations are done rarely and no further read accesses to that data are performed, so there is no benefit in caching it.
write-back (wb)	A caching mode in which data is written first to the cache and then mirrored to primary storage when IO bandwidth is available. The process of mirroring to primary storage is known as a *lazy write*.
write-through (wt)	A caching mode in which every write to the cache causes a synchronous write to primary storage.
write-only (wo)	A caching mode in which data is written first to the cache and then mirrored to primary storage when IO bandwidth is available. Reads can be served by the cache device only if previously written otherwise reads are bypassed and read from primary storage.

Frequently Asked Questions

This appendix provides contact information and answers to frequently asked questions.

How do I post an issue related to Open CAS Linux

Post a new issue at the following URL:

- Open CAS GitHub Issue (https://github.com/Open-CAS/open-cas-linux/issues/new)

Why does Open CAS Linux use some DRAM space?

Open CAS Linux uses a portion of system memory for metadata, which tells us where data resides. The amount of memory needed is proportional to the size of the cache space. This is true for any caching software solution. However with Open CAS Linux this memory footprint can be decreased using a larger cache line size set by the parameter –*cache-line-size* which may be useful in high density servers with many large HDDs.

Does Open CAS Linux work with non-Intel® SSDs?

Yes, however we validate only on Intel(R) SSDs. In addition Open CAS Linux utilizes the features of Intel(R) SSDs to provide improved performance and functionality that may not be available with other products.

How do I test performance?

In addition to the statistics provided (see Monitoring Open CAS Linux for details), third-party tools are available that can help you test IO performance on your applications and system, including:

- FIO (http://freecode.com/projects/fio)
- dt (https://github.com/RobinTMiller/dt) for disk access simulations

Is it possible to experience slower than HDD performance when using caching?

Yes, it is possible. For example, if the cache is in write-back mode and the entire cache is full of dirty data and a read occurs which requires new blocks to be loaded into the cache, performance will be degraded even if the read is sequential. The cache must first evict dirty blocks, which requires random writes to the HDD, then read the new data from the HDD and finally, write it to the cache. Whereas, without caching it would have simply resulted in a single read from the HDD. To avoid situations such as these, Open CAS Linux opportunistically flushes dirty data from the cache during idle IO times.

Where are the cached files located?

Open CAS Linux does not store files on disk; it uses a pattern of blocks on the SSD as its cache. As such, there is no way to look at the files it has cached.

How do I delete all the Open CAS Linux installation files?

Stop the Open CAS Linux software as described in the section **Stopping Cache Instances**. If Open CAS Linux was installed via the `make install` command, Open CAS Linux can be uninstalled using the command:

```
# make uninstall
```

Otherwise, if Open CAS Linux was manually installed then manually unload any CAS kernel modules. Once uninstalled remove the source files.

Does Open CAS Linux support write-back caching?

Yes. See the section **Manual Configuration for Write-Back Mode** for details.

Must I stop caching before adding a new pair of cache/core devices?

No, you can create new cache instances while other instances are running.

Can I assign more than one core device to a single cache?

Yes. Many core devices (up to 4096) may be associated with a single cache drive or instance. You can add them using the `casadm -A` command.

Can I add more than one cache to a single core device?

No, if you want to map multiple cache devices to a single core device, the cache devices must appear as a single block device through the use of a system such as RAID-0.

Why do tools occasionally report data corruption with Open CAS Linux?

Some applications, especially micro benchmarks like *dt* and *FIO*, may use a device to perform direct or raw accesses. Some of these applications may also allow you to configure values like a device’s alignment and block size restrictions explicitly, for instance via user parameters (rather than simply requesting these values from the device). In order for these programs to work, the block size and alignment for the cache device must match the block size and alignment selected in the tool.

Do I need to partition the cache device?

No. If you do not specify a partition, Open CAS Linux uses the entire device as the cache device.

Can I use a partition on a SSD as a cache device?

Yes, however, using the entire SSD device as the cache is highly recommended for best performance.

Do I need to format the partition or the device configured as the cache device?

No, the cache device has no format requirement. However, note that Open CAS Linux overwrites any data previously stored on the caching device (including any partition formatting).

What is the logical and physical block size for Open CAS Linux cache volumes (for exported objects)?

The logical block size for Open CAS Linux cache volumes is inherited from the core device, while the physical block size will be represented as the larger of the physical block sizes of the cache or core devices.

It is not possible to add a core device to a cache instance when the logical block size of the cache device is greater than that of the core device (eg. when the SSD has 4KiB logical block size and the HDD has 512B logical block size).

What happens if my SSD or HDD becomes unresponsive or disconnected?

In the event of a cache or core device becoming unresponsive, Open CAS Linux will fail all IO to all exported devices for the related cache (eg. /dev/cas1-1, /dev/cas1-2, etc.). To resume IO to the exported devices for the given cache, the user must restart the affected cache (in this example, cache ID 1).

When my device becomes disconnected, will I receive any notification from Cache Acceleration Software (CAS)?

No. The OS does not send notification to Open CAS Linux of the disconnection of the device, so the software will not know of this event until IO to the device is attempted. The device will still be listed in the –list-caches and –stats output, and no warning will be logged, until IO to the device is attempted. Check /var/log/messages and dmesg for standard Linux device IO errors.

Where is the log file located?

All events are logged in the standard Open CAS Linux system logs. Use the *dmesg* command or inspect the */var/log/messages* file.

To log all messages during testing or kernel debugging, use the command:

```
# echo 8 > /proc/sys/kernel/printk.
```

Typical log sample of successful cache initialization:

```
Cache line size: 64 KiB
Metadata capacity: 25 MiB
Parameters (policies) accepted:: 0 1 1 4
Pre-existing metadata, Clean shutdown
Done saving cache state!
Cache 1 successfully added
IO Scheduler of cas1-1 is cfq
Core “/dev/sdc” successfully added
```

Typical log sample of successful cache removal:

```
Removing Cache 1
Trying to remove 1 cached device(s)
Removed device cas1-1.
Done saving cache state!
Cache 1 successfully removed
```

Why does flushing the cache drive in Write Back mode take so long?

Flushing time has many factors, including but not limited to, cache device capacity, storage performance, and overall system utilization. Flushing time can vary greatly depending on these factors. You can check the status of the cache device being flushed by using the -L option in casadm.

```
# casadm -L
```

The following command can be used as well to verify activity and loading of the IO devices.

```
# iostat -xmt 1
```

Why does the CAS statistic counter for "Reads from Cache" increase when only write operations are being issued from the host application?

The *Reads from Cache* counter from the CAS statistic output (See *Monitoring CAS*) can increase when only doing writes from the host in several cases. For example, during the flushing of dirty data to the backing store in write-back or write-only mode. Similarly, during eviction when the cache is full in write-back or write-only mode and there is in incoming write which requires to read dirty data from the cache and flush it to backing store in order to make space for the incoming write. Another example is if there are any misaligned I/O that requires read-modify-write of a block that is in the cache.

Should nr_request be modified for further tuning?

The *nr_requests* controls how many requests may be allocated in the block layer for read or write requests. It represents the IO queue size. Each request queue has a limit on the total number of request descriptors that can be allocated for each read and write IO. By default, the number is 128, meaning 128 reads and 128 writes can be queued at a time before putting a process to sleep.

Large values of *nr_request* may increase throughput for workloads writing many small files. For larger IO operations, you may decrease the *nr_request* value. To get better read performance, you can set the *nr_request* value to 1024 for example, but increasing the value too high might introduce latency and degrade write performance. For latency sensitive applications, the converse is also true.

Configuring *nr_requests* to a larger value may improve performance for some customers who are caching multi-HDD RAID onto one SSD. However, this may not be useful with environments, such as Ceph, where each HDD is being cached to one SSD or a partition on an SSD. Your application and its IO patterns may vary and require a detailed tuning exercise.

To change the *nr_requests* value, use the following procedure:

```
# cat /sys/block/sda/queue/nr_requests
128
# echo 256 > /sys/block/sda/queue/nr_requests
```