zram: Compressed RAM-based block devices

Introduction

The zram module creates RAM-based block devices named /dev/zram<id> (<id> = 0, 1, ...). Pages written to these disks are compressed and stored in memory itself. These disks allow very fast I/O and compression provides good amounts of memory savings. Some of the use cases include /tmp storage, use as swap disks, various caches under /var and maybe many more. :)

Statistics for individual zram devices are exported through sysfs nodes at /sys/block/zram<id>/

Usage

There are several ways to configure and manage zram device(-s):

a. using zram and zram_control sysfs attributes
b. using zramctl utility, provided by util-linux (util-linux@vger.kernel.org).

In this document we will describe only 'manual' zram configuration steps, IOW, zram and zram_control sysfs attributes.

In order to get a better idea about zramctl please consult util-linux documentation, zramctl man-page or *zramctl --help*. Please be informed that zram maintainers do not develop/maintain util-linux or zramctl, should you have any questions please contact util-linux@vger.kernel.org

Following shows a typical sequence of steps for using zram.

WARNING

For the sake of simplicity we skip error checking parts in most of the examples below. However, it is your sole responsibility to handle errors.

zram sysfs attributes always return negative values in case of errors. The list of possible return codes:

-EBUSY	an attempt to modify an attribute that cannot be changed once the device has been initialised. Please reset device first.
-ENOMEM	zram was not able to allocate enough memory to fulfil your needs.
-EINVAL	invalid input has been provided.

If you use 'echo', the returned value is set by the 'echo' utility, and, in general case, something like:

```
echo 3 > /sys/block/zram0/max_comp_streams
if [ $? -ne 0 ]; then
        handle_error
fi
```

should suffice.

1) Load Module

```
modprobe zram num_devices=4
```

This creates 4 devices: /dev/zram{0,1,2,3}

num_devices parameter is optional and tells zram how many devices should be pre-created. Default: 1.

2) Set max number of compression streams

Regardless of the value passed to this attribute, ZRAM will always allocate multiple compression streams - one per online CPU - thus allowing several concurrent compression operations. The number of allocated compression streams goes down when some of the CPUs become offline. There is no single-compression-stream mode anymore, unless you are running a UP system or have only 1 CPU online.

To find out how many streams are currently available:

```
cat /sys/block/zram0/max_comp_streams
```

3) Select compression algorithm

Using comp_algorithm device attribute one can see available and currently selected (shown in square brackets) compression algorithms, or change the selected compression algorithm (once the device is initialised there is no way to change compression algorithm).

Examples:

```
#show supported compression algorithms
cat /sys/block/zram0/comp_algorithm
lzo [lz4]

#select lzo compression algorithm
echo lzo > /sys/block/zram0/comp_algorithm
```

For the time being, the *comp_algorithm* content does not necessarily show every compression algorithm supported by the kernel. We keep this list primarily to simplify device configuration and one can configure a new device with a compression algorithm that is not listed in *comp_algorithm*. The thing is that, internally, ZRAM uses Crypto API and, if some of the algorithms were built as modules, it's impossible to list all of them using, for instance, /proc/crypto or any other method. This, however, has an advantage of permitting the usage of custom crypto compression modules (implementing S/W or H/W compression).

4) Set Disksize

Set disk size by writing the value to sysfs node 'disksize'. The value can be either in bytes or you can use mem suffixes. Examples:

```
# Initialize /dev/zram0 with 50MB disksize
echo $((50*1024*1024)) > /sys/block/zram0/disksize

# Using mem suffixes
echo 256K > /sys/block/zram0/disksize
echo 512M > /sys/block/zram0/disksize
echo 1G > /sys/block/zram0/disksize
```

Note: There is little point creating a zram of greater than twice the size of memory since we expect a 2:1 compression ratio. Note that zram uses about 0.1% of the size of the disk when not in use so a huge zram is wasteful.

5) Set memory limit: Optional

Set memory limit by writing the value to sysfs node 'mem_limit'. The value can be either in bytes or you can use mem suffixes. In addition, you could change the value in runtime. Examples:

```
# limit /dev/zram0 with 50MB memory
echo $((50*1024*1024)) > /sys/block/zram0/mem_limit

# Using mem suffixes
echo 256K > /sys/block/zram0/mem_limit
echo 512M > /sys/block/zram0/mem_limit
echo 1G > /sys/block/zram0/mem_limit

# To disable memory limit
echo 0 > /sys/block/zram0/mem_limit
```

6) Activate

```
mkswap /dev/zram0
swapon /dev/zram0

mkfs.ext4 /dev/zram1
mount /dev/zram1 /tmp
```

7) Add/remove zram devices

zram provides a control interface, which enables dynamic (on-demand) device addition and removal.

In order to add a new /dev/zramX device, perform a read operation on the hot_add attribute. This will return either the new device's device id (meaning that you can use /dev/zram<id>) or an error code.

Example:

```
cat /sys/class/zram-control/hot_add
1
```

To remove the existing /dev/zramX device (where X is a device id) execute:

```
echo X > /sys/class/zram-control/hot_remove
```

8) Stats

Per-device statistics are exported as various nodes under /sys/block/zram<id>/

A brief description of exported device attributes follows. For more details please read Documentation/ABI/testing/sysfs-block-zram.

Name	access	description
disksize	RW	show and set the device's disk size
initstate	RO	shows the initialization state of the device
reset	WO	trigger device reset
mem_used_max	WO	reset the *mem_used_max* counter (see later)
mem_limit	WO	specifies the maximum amount of memory ZRAM can use to store the compressed data
writeback_limit	WO	specifies the maximum amount of write IO zram can write out to backing device as 4KB unit
writeback_limit_enable	RW	show and set writeback_limit feature
max_comp_streams	RW	the number of possible concurrent compress operations
comp_algorithm	RW	show and change the compression algorithm
compact	WO	trigger memory compaction
debug_stat	RO	this file is used for zram debugging purposes
backing_dev	RW	set up backend storage for zram to write out
idle	WO	mark allocated slot as idle

User space is advised to use the following files to read the device statistics.

File /sys/block/zram<id>/stat

Represents block layer statistics. Read Documentation/block/stat.rst for details.

File /sys/block/zram<id>/io_stat

The stat file represents device's I/O statistics not accounted by block layer and, thus, not available in zram<id>/stat file. It consists of a single line of text and contains the following stats separated by whitespace:

failed_reads	The number of failed reads
failed_writes	The number of failed writes
invalid_io	The number of non-page-size-aligned I/O requests
notify_free	Depending on device usage scenario it may account a. the number of pages freed because of swap slot free notifications b. the number of pages freed because of REQ_OP_DISCARD requests sent by bio. The former ones are sent to a swap block device when a swap slot is freed, which implies that this disk is being used as a swap disk. The latter ones are sent by filesystem mounted with discard option, whenever some data blocks are getting discarded.

File /sys/block/zram<id>/mm_stat

The mm_stat file represents the device's mm statistics. It consists of a single line of text and contains the following stats separated by whitespace:

orig_data_size	uncompressed size of data stored in this disk. Unit: bytes
compr_data_size	compressed size of data stored in this disk
mem_used_total	the amount of memory allocated for this disk. This includes allocator fragmentation and metadata overhead, allocated for this disk. So, allocator space efficiency can be calculated using compr_data_size and this statistic. Unit: bytes
mem_limit	the maximum amount of memory ZRAM can use to store the compressed data
mem_used_max	the maximum amount of memory zram has consumed to store the data
same_pages	the number of same element filled pages written to this disk. No memory is allocated for such pages.
pages_compacted	the number of pages freed during compaction
huge_pages	the number of incompressible pages
huge_pages_since	the number of incompressible pages since zram set up

File /sys/block/zram<id>/bd_stat

The bd_stat file represents a device's backing device statistics. It consists of a single line of text and contains the following stats separated by whitespace:

bd_count	size of data written in backing device. Unit: 4K bytes
bd_reads	the number of reads from backing device Unit: 4K bytes
bd_writes	the number of writes to backing device Unit: 4K bytes

9) Deactivate

```
swapoff /dev/zram0
umount /dev/zram1
```

10) Reset

Write any positive value to 'reset' sysfs node:

```
echo 1 > /sys/block/zram0/reset
echo 1 > /sys/block/zram1/reset
```

This frees all the memory allocated for the given device and resets the disksize to zero. You must set the disksize again before reusing the device.

Optional Feature

writeback

With CONFIG_ZRAM_WRITEBACK, zram can write idle/incompressible page to backing storage rather than keeping it in memory. To use the feature, admin should set up backing device via:

```
echo /dev/sda5 > /sys/block/zramX/backing_dev
```

before disksize setting. It supports only partitions at this moment. If admin wants to use incompressible page writeback, they could do it via:

```
echo huge > /sys/block/zramX/writeback
```

To use idle page writeback, first, user need to declare zram pages as idle:

```
echo all > /sys/block/zramX/idle
```

From now on, any pages on zram are idle pages. The idle mark will be removed until someone requests access of the block. IOW, unless there is access request, those pages are still idle pages. Additionally, when CONFIG_ZRAM_MEMORY_TRACKING is enabled pages can be marked as idle based on how long (in seconds) it's been since they were last accessed:

```
echo 86400 > /sys/block/zramX/idle
```

In this example all pages which haven't been accessed in more than 86400 seconds (one day) will be marked idle.

Admin can request writeback of those idle pages at right timing via:

```
echo idle > /sys/block/zramX/writeback
```

With the command, zram will writeback idle pages from memory to the storage.

If an admin wants to write a specific page in zram device to the backing device, they could write a page index into the interface.

```
echo "page_index=1251" > /sys/block/zramX/writeback
```

If there are lots of write IO with flash device, potentially, it has flash wearout problem so that admin needs to design write limitation to guarantee storage health for entire product life.

To overcome the concern, zram supports "writeback_limit" feature. The "writeback_limit_enable"'s default value is 0 so that it doesn't limit any writeback. IOW, if admin wants to apply writeback budget, they should enable writeback_limit_enable via:

```
$ echo 1 > /sys/block/zramX/writeback_limit_enable
```

Once writeback_limit_enable is set, zram doesn't allow any writeback until admin sets the budget via /sys/block/zramX/writeback_limit.

(If admin doesn't enable writeback_limit_enable, writeback_limit's value assigned via /sys/block/zramX/writeback_limit is meaningless.)

If admin wants to limit writeback as per-day 400M, they could do it like below:

```
$ MB_SHIFT=20
$ 4K_SHIFT=12
$ echo $((400<<MB_SHIFT>>4K_SHIFT)) > \
        /sys/block/zram0/writeback_limit.
$ echo 1 > /sys/block/zram0/writeback_limit_enable
```

If admins want to allow further write again once the budget is exhausted, they could do it like below:

```
$ echo $((400<<MB_SHIFT>>4K_SHIFT)) > \
        /sys/block/zram0/writeback_limit
```

If an admin wants to see the remaining writeback budget since last set:

```
$ cat /sys/block/zramX/writeback_limit
```

If an admin wants to disable writeback limit, they could do:

```
$ echo 0 > /sys/block/zramX/writeback_limit_enable
```

The writeback_limit count will reset whenever you reset zram (e.g., system reboot, echo 1 > /sys/block/zramX/reset) so keeping how many of writeback happened until you reset the zram to allocate extra writeback budget in next setting is user's job.

If admin wants to measure writeback count in a certain period, they could know it via /sys/block/zram0/bd_stat's 3rd column.

memory tracking

With CONFIG_ZRAM_MEMORY_TRACKING, user can know information of the zram block. It could be useful to catch cold or incompressible pages of the process with*pagemap.

If you enable the feature, you could see block state via /sys/kernel/debug/zram/zram0/block_state". The output is as follows:

```
300    75.033841 .wh.
301    63.806904 s...
302    63.806919 ..hi
```

First column

zram's block index.

Second column

access time since the system was booted

Third column

state of the block:

s:
same page

w:
written page to backing store

h:
huge page

i:
idle page

First line of above example says 300th block is accessed at 75.033841sec and the block's state is huge so it is written back to the backing storage. It's a debugging feature so anyone shouldn't rely on it to work properly.

Nitin Gupta ngupta@vflare.org