

zram

Compressed RAM-based block devices

github.com/exc11g

Что такое zram

- Модуль Linux, который создает блочное устройство в оперативной памяти (необязательно в оперативной), куда кладет данные в сжатом виде, предварительно сжимая их. Таким образом мы хотим избавиться от файлов подкачки, которые работают с диском, вместо оперативной памяти, тем самым искусственно увеличить размер оперативной памяти на устройстве.
- Есть в ядре Linux с версии 3.14, а так же используется в Android, ChromeOS and etc..

Почему мы хотим избавиться от файлов подкачки?

- Постоянное обращение к диску, будь то HDD/SSD, что достаточно дорогая операция с точки зрения производительности, т.к. обращение к оперативной памяти намного быстрее
- Постоянное обращение к таким устройствам как SSD или Карта Памяти на Android очень изнашивает его, таким образом использование файла подкачки быстро изнашивает ресурс диска

Как работает zram

- Первое, что делает zram это устанавливается и пользователь его конфигурирует, с этим мы разберемся немного позже
- Далее он выделяет несколько сжатых блочных устройств в оперативной памяти (примерно соизмеримо с кол-вом ядер процессора, т.к. работа с каждым блоком ведется отдельно от всех в одном потоке)
- Модуль должен уметь принимать какую-то последовательность байт, применять какой-нибудь алгоритм сжатия и уже сжатую последовательность класть в эти блоки в оперативной памяти.

Как работает zram

- Размер этих блоков фиксированный, для удобства он выбран размеру странички (PAGE_SIZE)
- Как только zram получает страничку он должен поддерживать свои внутренние структуры данных, чтобы знать что и куда он кладет. Для этого он использует таблицу, реализованную через красно-черное дерево, чтобы в моменте, когда страница понадобится он его нашел и распаковал, т.к. сжатыми данными пользоваться невозможно.

Zsmalloc аллокатор

- Появилась необходимость выделять память для сжатых страниц и нужно было делать это особенно эффективно. Сначала использовался очень хороший аллокатор из ядра - `kmalloc()`, однако под наши задачи он не очень хорошо подходит: т.к. нам надо выделять память в условиях ее недостаточности у нас есть 2 очевидных приоритета: минимальность выделения больших блоков памяти, минимальная фрагментация и максимизация сжатых страниц хранящихся в одном блоке. Поэтому было написано множество различных аллокаторов, однако у них были особенные минусы, из-за которых от них отказались. Таким образом был написан Zsmalloc аллокатор.

Zsmalloc аллокатор

- Он унаследовал лучшие детали от kmalloc однако появились некоторые особенности, благодаря которым он оказался эффективнее в данной задаче: Нет требования выделять блоки размер большего порядка, а так же сжатые страницы одного "класса" (примерно схожего размера) лежали рядом.
- Таким образом данный аллокатор достигает высоким показателем плотности сжатых страниц различного размера
- Однако есть и минусы: из-за высокой плотности в пределах одного класса усложнено удаление страниц и как следствие появляется фрагментация в других классах.

Zbud аллокатор

- Данный аллокатор был предложен еще до zsmalloc, однако из-за некоторых минусов последнего данный аллокатор доработали и на сегодняшний день они оба используется в системе.
- Zbud уделяет меньше внимания высокой плотности, а больше - способности эффективно удалять сжатые страницы, с чем у zsmalloc были проблемы.

Xvmmalloc allocator

- Наконец allocator который используется в zram – xvmmalloc
- Разработанный специально для этого проекта allocator имеет хорошую асимптотику $O(1)$ для alloc/free, очень низкую фрагментацию, но при этом нестандартный интерфейс:

```
int xv_malloc(struct xv_pool * пул, размер u32, u32  
* число страниц, u32 * смещение, флаги gfp_t);  
void xv_free(struct xv_pool *pool, u32 pagenum, u32 offset);
```
- Такой allocator стали использовать вместо SLOB и SLUB из-за их недостатков конкретно в данном проекте

Xvmmalloc allocator

- Например SLOB хоть и обладает хорошей экономией места, однако из-за асимптотики $O(n)$ на alloc/free он стал непригоден в zram.
- А у SLUB - фрагментация, было замечено что kmmalloc использует на 43% больше памяти, чем xvmmalloc
- Так же у SLOB и SLUB allocаторов есть проблемы с выделением маленьких кусков памяти, эту проблему решали еще в zsmalloc и zbud allocаторах.
- Поэтому xvmmalloc заменил эти allocаторы в проекте, может быть заменит когда-нибудь и во всем ядре..

Некоторые особенности при сжатии страницы

- Сжатие и распаковка страниц не очень дешевая операция, поэтому мы хотим ограничить кол-во действий алгоритма сжатия. Например, хотим ли мы сжимать страницу если вскоре она понадобится снова и ее придется разжать? Поэтому мы должны тщательно отбирать страницы которые мы хотим сжимать.
- Предугадать размер сжатой страницы - сложная задача. В среднем страницы сжимаются примерно в 2 раза, однако все зависит от того какие данные мы сжимаем и какой алгоритм мы выбрали. Про выбор алгоритма мы еще поговорим далее.

Установка zram

- Перед установкой zram отключите файлы подкачки, т.к. их совместная работа невозможна, т.к. все данные будут перехватываться файлом подкачки и zram даже не успеет поработать.
- Укажем количество сжатых блоков:
`$ modprobe zram num_devices=4`
- Посмотрим доступные алгоритмы сжатия, текущий алгоритм выделен квадратными скобками:
`$ cat /sys/block/zram0/comp_algorithm`
`lzo [lz4]`

Установка zram

- Выберем другой алгоритм (опционально):
`$ echo lzo > /sys/block/zram0/comp_algorithm`
 - Укажем размер для каждого блока:
`$ echo $((50*1024*1024)) > /sys/block/zram0/disksize`
 - Ну и запустим сам модуль:
`$ mkswap /dev/zram0`
`$ swapon /dev/zram0`
-
- `$ mkfs.ext4 /dev/zram1`
 - `$ mount /dev/zram1 /tmp`

Какие алгоритмы использует zram?

- На самом деле сам zram не хранит в себе никакие алгоритмы сжатия, он пользуется Linux Crypto API, который и предоставляет список алгоритмов.
- Существует несколько самых популярных алгоритмов сжатия, используемых в zram.
- LZ4 - быстрый и эффективный алгоритм. Низкое потребление памяти и возможность быстрого сжатия/распаковки данных
- Zstd - алгоритм из Facebook, который показывает очень хорошие коэффициенты сжатия без потерь в скорости








Какие алгоритмы использует zram?

- LZO - отличный баланс между производительностью и потреблением ресурсов.
- DEFLATE - лежит в основе gzip и PNG
- Так же существует еще множество алгоритмов, однако они могут не поставляться Crypto API в зависимости от версии ядра.
- Выбор алгоритма нетривиальная задача, которая зависит от производительности вашего процессора, данными с которыми вы работаете, объемом оперативной памяти, однако вы можете использовать самые популярные алгоритмы и в среднем получить лучшую производительность.

Zram & Android

- В Android в отличие от Linux нет файла подкачки, т.к. он бы очень быстро потратил ресурс памяти, поэтому в Android так же был введен модуль zram.
- Т.к. операционная система Android часто большую часть времени работает в нехватке оперативной памяти из-за того что открытые приложения почти всегда лежат в оперативной памяти, пока пользователь их не освободит, поэтому такой механизм необходим в Android.
- Существует оценка работы приложений, чтобы определить кто именно пойдет в zram

Пример таблицы с процессами

Background apps	
Previous app	
Home app	
Services	
Perceptible apps	
Foreground app	
Persistent	
System	<code>system_server</code>
Native	<code>init kswapd netd logd adbd installd</code>

Примерно так выглядит таблица. Сверху показаны приложения у которых наивысший приоритет загрузки в zram, а снизу таблицы наименьший приоритет

Заключение

- Подводя итоги можно смело сказать что zram очень полезный модуль Linux, который в некоторых случаях может заменить файл подкачки и повысить производительность вашего устройства, будь то ноутбук, компьютер или смартфон.
- Гибкая настройка модуля позволяет подогнать работу под ваше устройство, что может помочь добиться максимальной производительности zram.

Литература

- [https://wiki.archlinux.org/title/Zram_\(Русский\)](https://wiki.archlinux.org/title/Zram_(Русский))
- https://wiki.archlinux.org/title/Improving_performance#zram_or_zswap
- <https://docs.kernel.org/admin-guide/blockdev/zram.html>
- <https://habr.com/ru/articles/693878/>
- <https://ru.wikipedia.org/wiki/ZRam>
- <https://lwn.net/Articles/334649/>
- <https://lwn.net/Articles/545244/>
- <https://developer.android.com/topic/performance/memory-management>
- <https://locall.host/which-zram-algorithm-is-best/>
- https://github.com/torvalds/linux/blob/master/drivers/block/zram/zram_drv.c