Christopher Feener
Summer Research Report
August 14, 2017

Setup VM for kernel:

The Virtual Machine used was Qemu-KVM. To create an image for the VM, do the following:

```
#Naming the image "qemu-img.img"
IMG=qemu-img.img
DIR=mount-point.dir
qemu-img create $IMG 1g
mkfs.ext2 $IMG
mkdir $DIR
sudo mount -o loop $IMG $DIR
sudo debootstrap --arch amd64 jessie $DIR
sudo umount $DIR
rmdir $DIR
```

Next, to make the kernel:

```
#Get and uncompress kernel, such as linux-4.11 (use something like "xzfv"),
#and go to resulting directory to execute the following commands:
cd ~/Desktop/linux-4.11

make x86_64_defconfig
make kvmconfig
make -j 8
```

To boot:

```
#Run this inside the directory that 'makeKernel' was executed.
cd ~/Desktop/linux-4.11

qemu-system-x86_64 -m 1G -cpu host -enable-kvm \
    -hda ~/Desktop/qemu-img.img  \
    -kernel arch/x86/boot/bzImage -append "root=/dev/sda single"
```

This information was mainly from https://www.collabora.com/news-and-blog/blog/2017/01/16/setting-up-qemu-kvm-for-kernel-development/. Unfortunately, this seemed to result in a VM that required the password from the host computer's administrator, which is unknown. Also, some additional software was required installed. Thus, a previously prepared image was used.

To start the VM (here called "kernel-4.11-vm.img"):

```
#Clear cache
sudo sync && sudo echo 3 | sudo tee /proc/sys/vm/drop_caches
qemu-system-x86_64 -m 1G -cpu host -enable-kvm -drive file=~/Desktop/kernel-4.11-vm.img,
throttling.iops-total=100000
```

To set up the VM with kernel 4.11, do the following steps inside the Qemu VM:
Step 1: First time setup (getting the kernel).

```
#Goes to home directory, for simplicity.
cd
wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.11.tar.xz
tar xf linux-4.11.tar.xz
```

(Note: For kernel version 4.12, simply change instances of 4.11.)

Step 2: Make and Reboot.

```
cd ~/linux-4.11
cp /boot/config-$(uname -r ) .config
make menuconfig
sudo make -j 4 && sudo make modules_install -j && sudo make install -j 4
    #If the make is successful, then reboot:
sudo reboot
```

Step 3: Run code.
Step 4: Edit code.
Step 5: Restart at Step 2 as needed.

Setup experiments on kernel:

The program used to test experiments was ImageMagick, which is already installed inside most Linux machines. This was run on a 4.3 GB picture of a galaxy., retrieved online using the following:

```
wget https://cdn.spacetelescope.org/archives/images/publicationjpg/heic0602a.jpg
```

Most changes were inside linux-4.11/mm/vmscan.c: shrink_page_list() on line 948, which deals with page faults. Printk statements were added to print information to the screen. If there was an overfill, the command "dmesg" could be piped into "less" to allow the user to scroll through the output.

Setup other (YCSB):

Unfortunately, the portion known as "Mapkeeper" did not have current documentation. Thus, due to time constraints, a previously-made VM image was used, instead.

To run YCSB on a workload, the following lines were executed. Due to time constraints, only 3 trials were done on each experiment, with just loading, no running. There were six workloads (A-F). The following is for workload C:

```
#The cache was cleared:
sudo sync && sudo echo 3 | sudo tee /proc/sys/vm/drop_caches

#The mapkeeper was run in the background:
cd ~/mapkeeper/leveldb
./mapkeeper_leveldb &
cd ~/YCSB

#A trial of the experiment was run:
./bin/ycsb load mapkeeper -p mapkeeper.port=9090 -P workloads/workloadc -threads 4 >>
~/researchOutput/k1G_C_1M        2>&1

#The user table was reloaded to clean out any old changes:
```

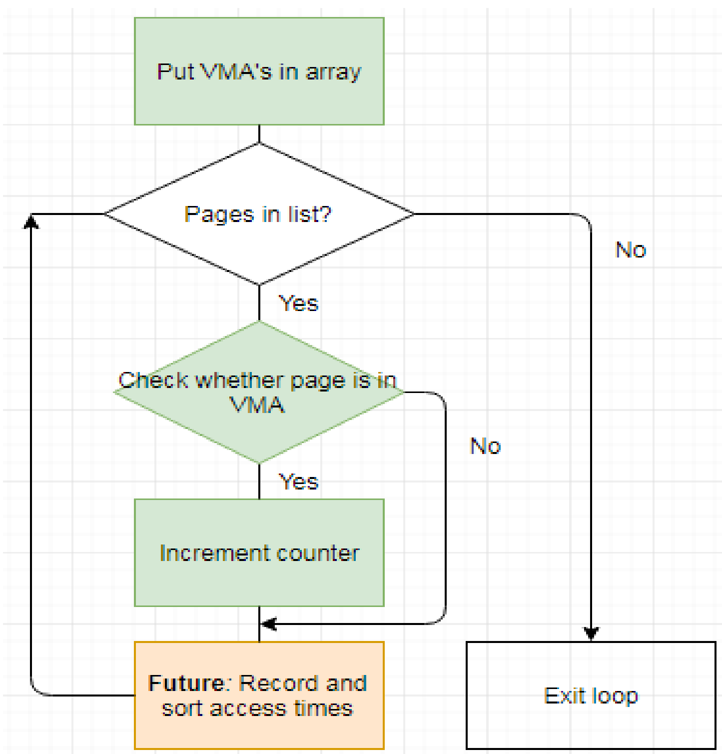
```
rm -rf ~/mapkeeper/leveldb/data/usertable
cd ~
cp -r usertable ~/mapkeeper/leveldb/data
```

When changes to *shrink_page_list()* were made, YCSB crashed. ImageMagick was used, instead.

Other findings:

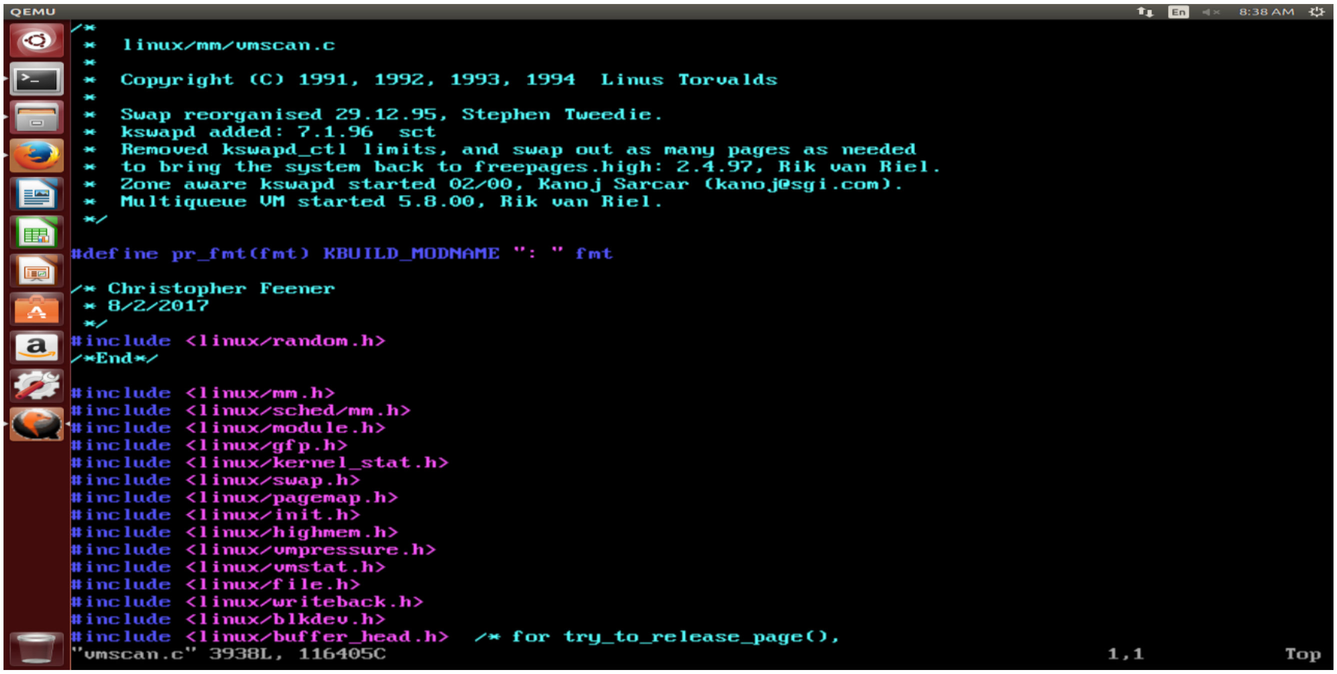
Main changes inside *shrink_page_list()*:
A fixed number of VMA's (virtual memory addresses) were recorded in an array, starting from the current task. During the while-loop which checks each page, the page is checked for presence in any of the VMA's. If the page is found inside a VMA, the count for that VMA is incremented, and the newest count of pages for that VMA is printed.

The following is a flowchart of the changes (green are present changes, orange is future changes):



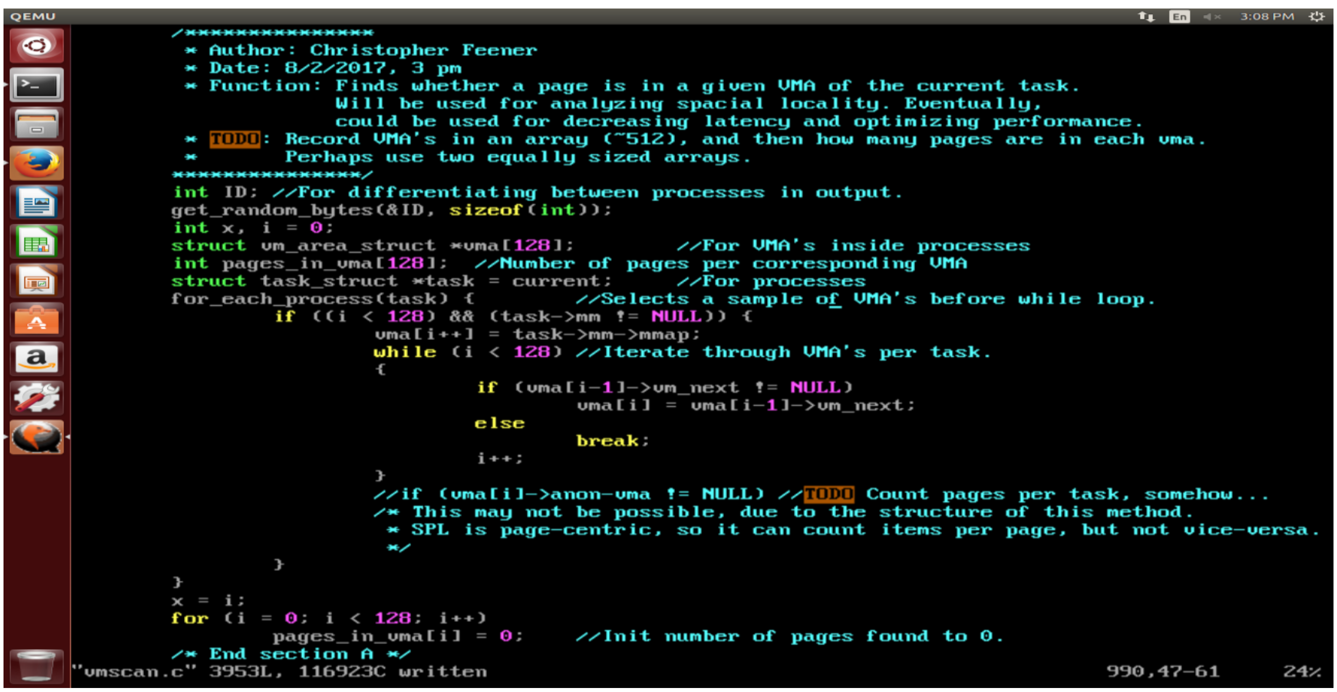
The following pages show code that was changed inside SPL:

Section 1/4:



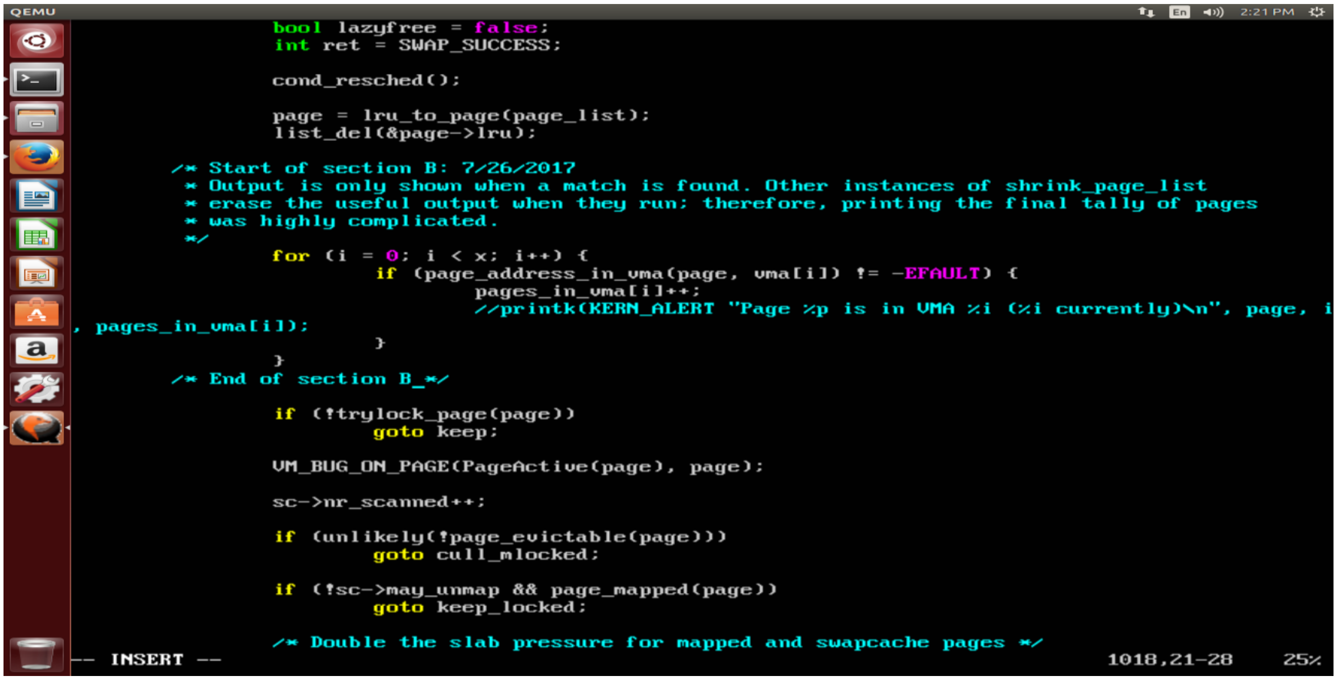
```
/*
 *  linux/mm/vmscan.c
 *
 *  Copyright (C) 1991, 1992, 1993, 1994  Linus Torvalds
 *
 *  Swap reorganised 29.12.95, Stephen Tweedie.
 *  kswapd added: 7.1.96  sct
 *  Removed kswapd_ctl limits, and swap out as many pages as needed
 *  to bring the system back to freepages.high: 2.4.97, Rik van Riel.
 *  Zone aware kswapd started 02/00, Kanoj Sarcar (kanoj@sgi.com).
 *  Multiqueue VM started 5.8.00, Rik van Riel.
 */

#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt

/* Christopher Feener
 * 8/2/2017
 */
#include <linux/random.h>
/*End*/

#include <linux/mm.h>
#include <linux/sched/mm.h>
#include <linux/module.h>
#include <linux/gfp.h>
#include <linux/kernel_stat.h>
#include <linux/swap.h>
#include <linux/pagemap.h>
#include <linux/init.h>
#include <linux/highmem.h>
#include <linux/vmpressure.h>
#include <linux/vmstat.h>
#include <linux/file.h>
#include <linux/writeback.h>
#include <linux/blkdev.h>
#include <linux/buffer_head.h>  /* for try_to_release_page(),
"vmscan.c" 3938L, 116405C                                    1,1          Top
```

Section 2/4:



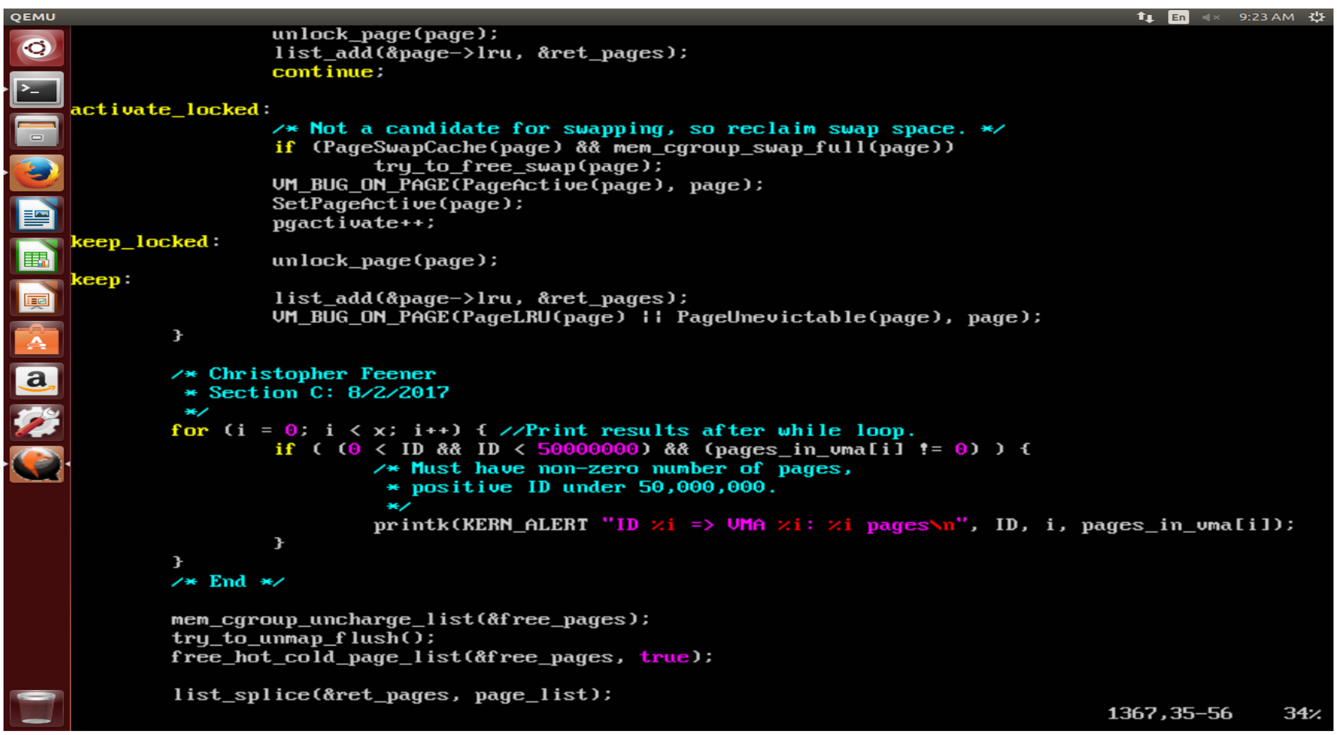
```
        /****************
         * Author: Christopher Feener
         * Date: 8/2/2017, 3 pm
         * Function: Finds whether a page is in a given VMA of the current task.
                     Will be used for analyzing spacial locality. Eventually,
                     could be used for decreasing latency and optimizing performance.
         * TODO: Record VMA's in an array (~512), and then how many pages are in each vma.
         *       Perhaps use two equally sized arrays.
        ****************/
        int ID; //For differentiating between processes in output.
        get_random_bytes(&ID, sizeof(int));
        int x, i = 0;
        struct vm_area_struct *vma[128];         //For VMA's inside processes
        int pages_in_vma[128];  //Number of pages per corresponding VMA
        struct task_struct *task = current;      //For processes
        for_each_process(task) {         //Selects a sample of VMA's before while loop.
                if ((i < 128) && (task->mm != NULL)) {
                        vma[i++] = task->mm->mmap;
                        while (i < 128) //Iterate through VMA's per task.
                        {
                                if (vma[i-1]->vm_next != NULL)
                                        vma[i] = vma[i-1]->vm_next;
                                else
                                        break;
                                i++;
                        }
                        //if (vma[i]->anon-vma != NULL) //TODO Count pages per task, somehow...
                        /* This may not be possible, due to the structure of this method.
                         * SPL is page-centric, so it can count items per page, but not vice-versa.
                         */
                }
        }
        x = i;
        for (i = 0; i < 128; i++)
                pages_in_vma[i] = 0;    //Init number of pages found to 0.
        /* End section A */
"vmscan.c" 3953L, 116923C written                              990,47-61     24%
```

Section 3/4:



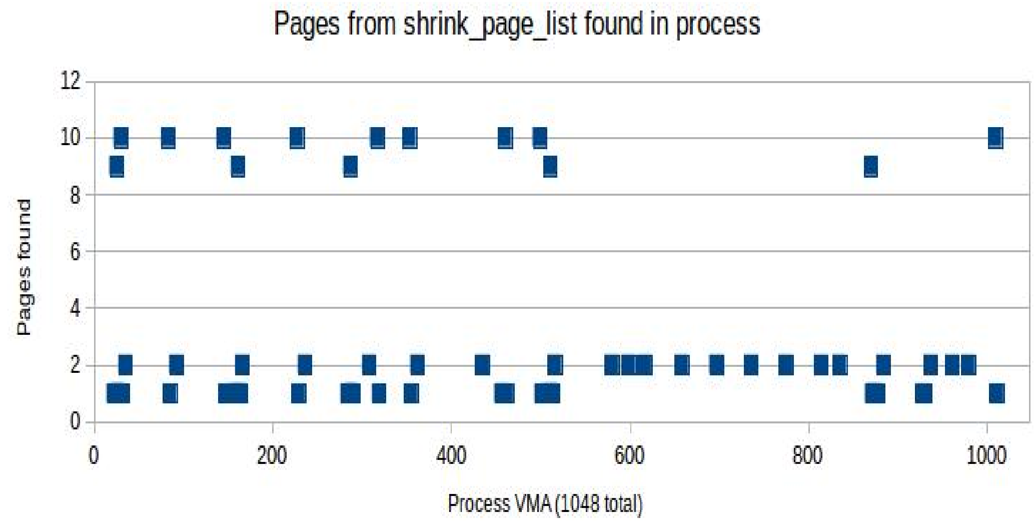
```
                bool lazyfree = false;
                int ret = SWAP_SUCCESS;

                cond_resched();

                page = lru_to_page(page_list);
                list_del(&page->lru);

        /* Start of section B: 7/26/2017
         * Output is only shown when a match is found. Other instances of shrink_page_list
         * erase the useful output when they run; therefore, printing the final tally of pages
         * was highly complicated.
         */
                for (i = 0; i < x; i++) {
                        if (page_address_in_vma(page, vma[i]) != -EFAULT) {
                                pages_in_vma[i]++;
                                //printk(KERN_ALERT "Page %p is in VMA %i (%i currently)\n", page, i
, pages_in_vma[i]);
                        }
                }
        /* End of section B */

                if (!trylock_page(page))
                        goto keep;

                VM_BUG_ON_PAGE(PageActive(page), page);

                sc->nr_scanned++;

                if (unlikely(!page_evictable(page)))
                        goto cull_mlocked;

                if (!sc->may_unmap && page_mapped(page))
                        goto keep_locked;

                /* Double the slab pressure for mapped and swapcache pages */
-- INSERT --                                                          1018,21-28     25%
```

Section 4/4:



```
                unlock_page(page);
                list_add(&page->lru, &ret_pages);
                continue;

activate_locked:
                /* Not a candidate for swapping, so reclaim swap space. */
                if (PageSwapCache(page) && mem_cgroup_swap_full(page))
                        try_to_free_swap(page);
                VM_BUG_ON_PAGE(PageActive(page), page);
                SetPageActive(page);
                pgactivate++;
keep_locked:
                unlock_page(page);
keep:
                list_add(&page->lru, &ret_pages);
                VM_BUG_ON_PAGE(PageLRU(page) || PageUnevictable(page), page);
        }

        /* Christopher Feener
         * Section C: 8/2/2017
         */
        for (i = 0; i < x; i++) { //Print results after while loop.
                if ( (0 < ID && ID < 50000000) && (pages_in_vma[i] != 0) ) {
                        /* Must have non-zero number of pages,
                         * positive ID under 50,000,000.
                         */
                        printk(KERN_ALERT "ID %i => VMA %i: %i pages\n", ID, i, pages_in_vma[i]);
                }
        }
        /* End */

        mem_cgroup_uncharge_list(&free_pages);
        try_to_unmap_flush();
        free_hot_cold_page_list(&free_pages, true);

        list_splice(&ret_pages, page_list);
                                                                      1367,35-56     34%
```

This graph shows how non-zero amounts of pages were patterned. The vast majority of pages overall were zero, while most of the rest hovered around 1 and 2. There were fourteen “peaks” at 9 or 10.



Using page information:

- Different pages accessed at nearly the same time and place could be further tested.
- Then, if a page is accessed often, it is swapped to disk.
- If not, it is sent to SSD.

- **In future work**, access times may be sorted from least to greatest, and then split into two sections: The first section going to SSDs, and the second to hard disks.