Assignment 3 Write Up – Page Replacement

Team: Prateek Chawla (pchawla), Adel Danandeh (addanand), Tim Mertogul (tmertogu, Team Captain)

Hypothesis

We speculate (hypothesize) that the Slim Chance algorithm (our modifications) is inferior to the default (standard FreeBSD) paging mechanism. To confirm or refute this hypothesis we tested both the standard FreeBSD and our modified page replacement algorithm 30 times each (60 tests total). Importantly, we standardized these tests in low memory environment to increase memory pressure.

For clarity purposes we refer to the Standard FreeBSD results as *"Original"* and our modified page replacement algorithm as *"Alternate"*.

FreeBSD Configurations:

In order to maintain consistency we used the same virtual machine configurations for all testing (i.e. the system specs did not change between runs of the standard FreeBSD code and our modifications).
Pertinent configurations are as follows (using vmware fusion on the same laptop):

- 128MB RAM
- 2 Cores

Our reasoning in using these specific configurations was an attempt to shrink our universe, so to speak. Essentially, it was to force memory pressure so meaningful statistics could be generated.

Stats Generation Methodology:

As stated in the design document, all the stats analysis is done in a function called vm_pageout_scan() which can be found under sys/vm directory. In each iteration of page daemon, all the pages in active and inactive are being scanned. Initially, time interval for it was set to 600 seconds (10 minutes), but it has been changed to 10 seconds. There are counters defined in order to keep track of the stats as we are asked on the assignment:

- *number of pages scanned*
- *number of pages moved from active to inactive*
- *number of pages moved from inactive to cache*
- *number of pages moved from inactive to free*
- *number of pages queued for flush*

Each individual counter gets incremented in the proper place (see the appropriate subsection in the design doc under 'Modified Files and Methods'). Specifically, in order to find the total number of pages scanned, in each iteration of pages in active and inactive queues the *number of pages scanned* is incremented. Again, all statistical information is explained in further detail in it’s appropriate section in the design doc. Importantly, all the stats are printed to a log file called messages which it could be found under */var/log/*

Additionally, we created a stress test script to help generate these statistics. Important components of this stress test included the `vmstat`, `/usr/bin/time` and `stress` commands. Notably, this testing (and general system configurations) were consistent between the standard standard FreeBSD code and our modifications.

Specifically we ran `/usr/bin/time -l stress -m 3 --vm-bytes 512M -t 20s`

- `/usr/bin/time -l`
    - Reports system resource usage during execution of a given command
    - We added this command to get specific page faults and reclaims for our stress test, rather than having to rely on deltas
- `stress -m 3 --vm-bytes 512M -t 20s`
    - A stress test (workload generator)
    - The `-m 3` flag spawned 3 workers spinning on malloc()/free()
    - The `--vm-bytes 512M` flag malloc 512M per vm worker
    - The `-t 20s` flag is a 20 second timeout

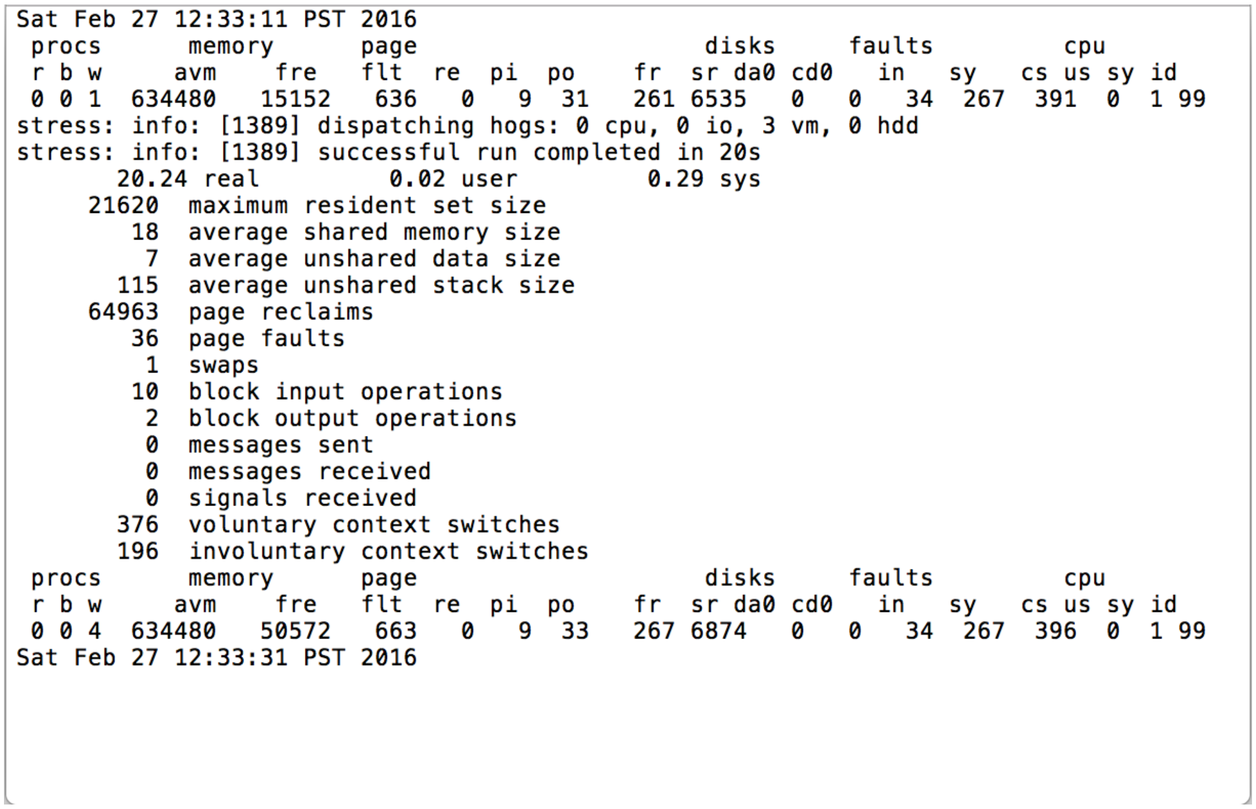
While, vmstat reported additional virtual memory statistics before and after each stress test. We additionally, wrote to log and printed the dates for reference purposes.
The full code of our stress test is as follows (Fig. 1):

```
sudo cp /dev/null /var/log/messages
date
vmstat
logger BEGINNING_STRESS
/usr/bin/time -l stress -m 3 --vm-bytes 512M -t 20s
logger ENDING_STRESS
vmstat
date
```

Fig. 1. The first line clears the log so we get fresh data, the second line prints the current time, the third line reports virtual memory stats before the stress test, the fourth line is for referencing purposes, the fifth is our stress test as explained above, the sixth is another reference line, the seventh reports virtual memory stats after the stress test and finally the eight line prints the current time after the stress test (to record the time interval).

We ran the above stress test in the following manner: ./stress.sh>stress.txt 2>&1

Below (Fig. 2) is an example of the output of a pseudorandomly selected stress test, specifically an instance of stress.txt.



```
Sat Feb 27 12:33:11 PST 2016
 procs      memory      page                    disks     faults         cpu
 r b w     avm    fre   flt  re  pi  po    fr  sr da0 cd0   in   sy    cs us sy id
 0 0 1  634480   15152   636   0   9  31   261 6535   0   0   34  267   391  0  1 99
stress: info: [1389] dispatching hogs: 0 cpu, 0 io, 3 vm, 0 hdd
stress: info: [1389] successful run completed in 20s
       20.24 real         0.02 user         0.29 sys
     21620  maximum resident set size
        18  average shared memory size
         7  average unshared data size
       115  average unshared stack size
     64963  page reclaims
        36  page faults
         1  swaps
        10  block input operations
         2  block output operations
         0  messages sent
         0  messages received
         0  signals received
       376  voluntary context switches
       196  involuntary context switches
 procs      memory      page                    disks     faults         cpu
 r b w     avm    fre   flt  re  pi  po    fr  sr da0 cd0   in   sy    cs us sy id
 0 0 4  634480   50572   663   0   9  33   267 6874   0   0   34  267   396  0  1 99
Sat Feb 27 12:33:31 PST 2016
```

Fig 2. Pertinent information includes, but is not limited to the number of page fault and reclaims that occur during stress testing.

Additionally, a pseudorandomly selected example of the logged output is shown below (Fig. 3). These variables are logged towards the bottom of vm_pageout_scan(). Importantly, we utilize the log() function from sys/syslog.h with the logging level LOG_INFO.

```
Feb 27 22:47:42 kernel: np_scanned: 1525
Feb 27 22:47:42 kernel: np_queued_for_flush: 51
Feb 27 22:47:42 kernel: np_moved_active_to_inactive: 443
Feb 27 22:47:42 kernel: np_moved_inactive_to_free: 0
Feb 27 22:47:42 kernel: np_moved_inactive_to_cache: 342
```

Fig 3. For details on these variables and how they're incremented please see the design doc

Testing Details and Specifics:

- The stress tests were ran 30 times for both the "original" and "alternate" algorithms.

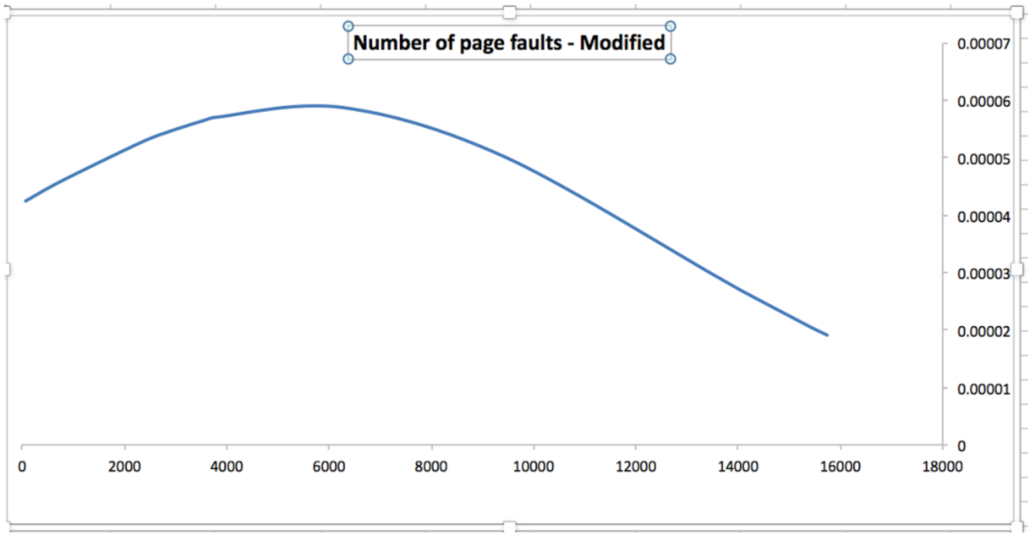
- All of our raw data and original graphs may be found in Paging_algorithm_data.xlsx but we have included the relevant data and graphs below.
- Our scripts and small programs for parsing and retrieving the data may be found under asgn3/scripts. All they do is take the data written to log and displayed by vmstat and write it directly to Excel documents, which we used to graphically interpret it.

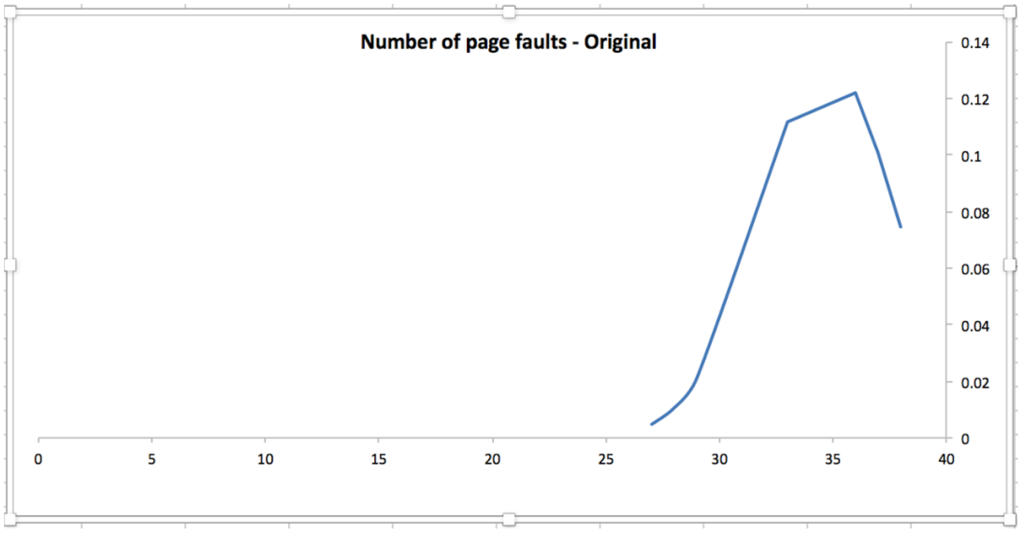
Findings

Through our 60 tests we generated a multitude of statistics. Specifically, we focused on the number of page faults, page reclaims, pages scanned, number of pages moved from inactive to free, number of pages moved from inactive to cache, and number of pages moved from queued for flush, as they dictate the effectiveness of our alterations, compared to the original.

Detailed findings for each of the categories listed above are displayed as followed:

------------------ Number of page faults ------------------

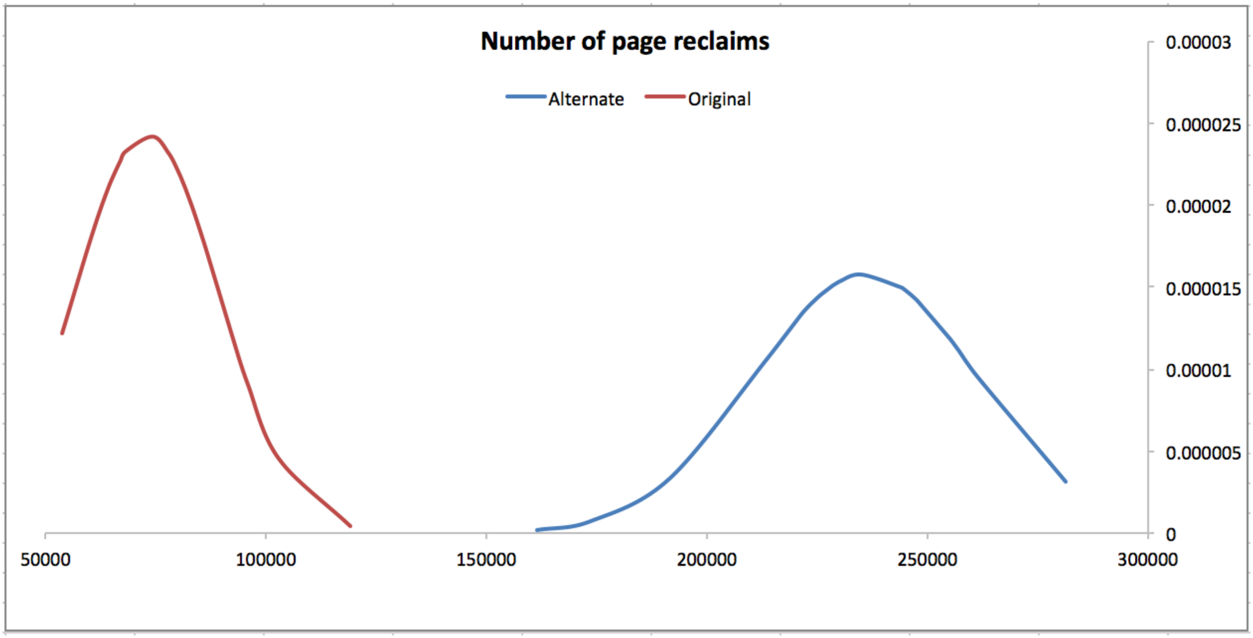




	Average	Standard Deviation
Alternate	5548.666667	6758.413752
Original	34.76666667	3.013570838

Our modified algorithm vastly ruined the number of page faults. The original algorithm had an average of ~35 page faults and our modified algorithm had an average of ~5550 page faults, a significant multiplier more. The modified algorithm had some instances where it would have page faults in the range of 35-90 but then it would jump incredibly high into the 1000s once the pages needed were removed by the poor logic in its design.

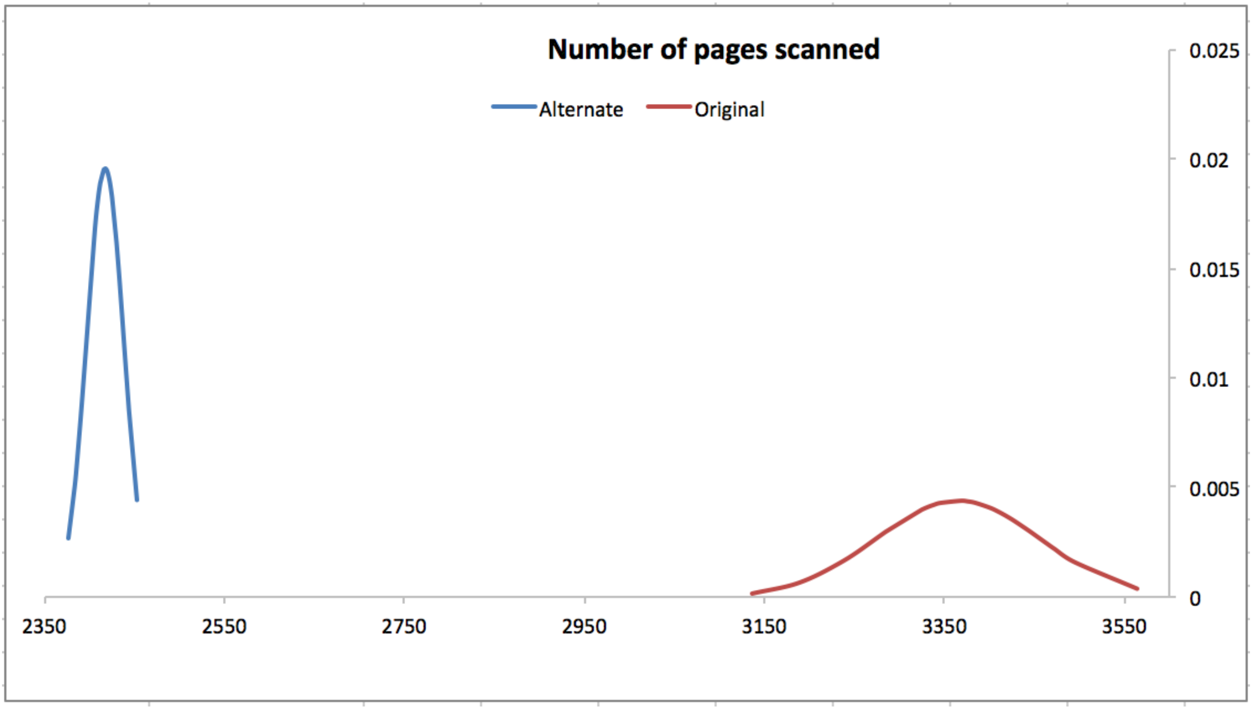
------------------ Number of page reclaims ------------------



	Average	Standard Deviation
Alternate	235876.4333	25275.67108
Original	72973.56667	16422.855

Our modified algorithm shows a significant increase in the number of page reclaims both on average and it shows a greater variance as well. This can be explained by our modification to place recently used pages on the *rear* of the queue, and not the front. In class we learned that the most recently used pages are more likely to be used again and our data supports this.

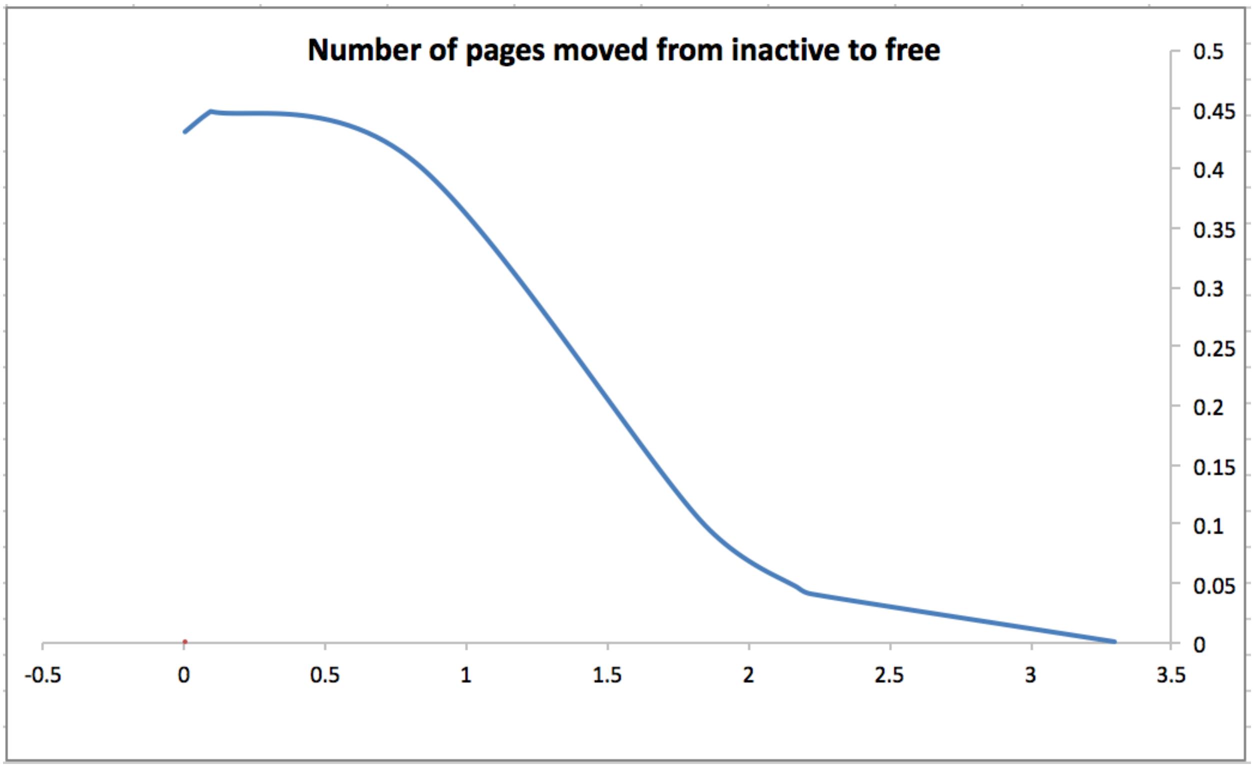
------------------ Number of pages scanned ------------------



	Average	Standard Deviation
Alternate	2416.937052	20.43313997
Original	3364.186396	91.61668841

We did not expect to see a huge difference between number of pages scanned in the original FreeBSD paging algorithm compare to our modified version. The reason for this result shown in the above table is that we divided the activity count for pages, therefore there will less active pages scanned, which that ultimately result in a huge difference between the original and the modified version of FreeBSD.

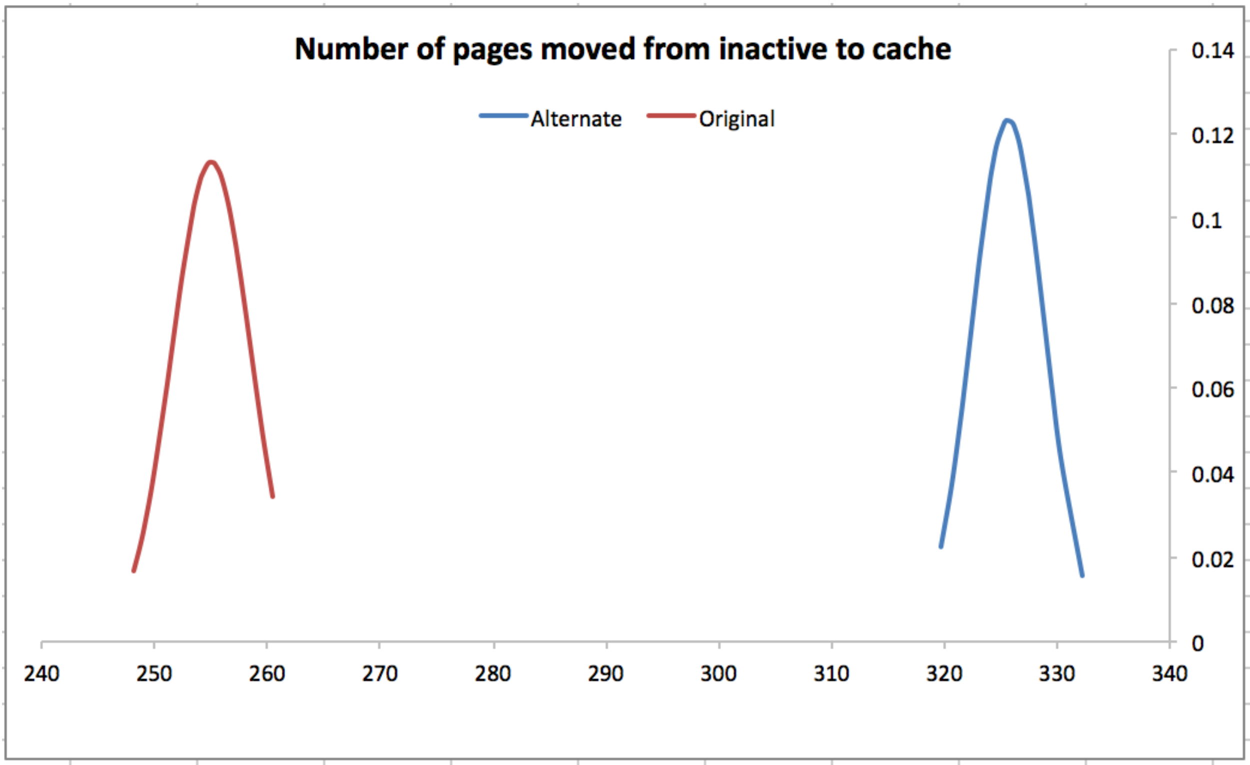
----------------- Number of pages moved from inactive to free -----------------



	Average	Standard Deviation
Alternate	0.355075346	0.847473045
Original	0	0

Please Note: We did not include a graph of the original paging algorithms results due to a divide by 0 error in excel.
The original paging algorithm has 0 pages moved from inactive to free while our paging algorithm occasionally moved a couple from inactive to free. The original algorithm is more efficient as it spent less resources and time moving pages to free, due to a more efficient way of keeping necessary pages around.

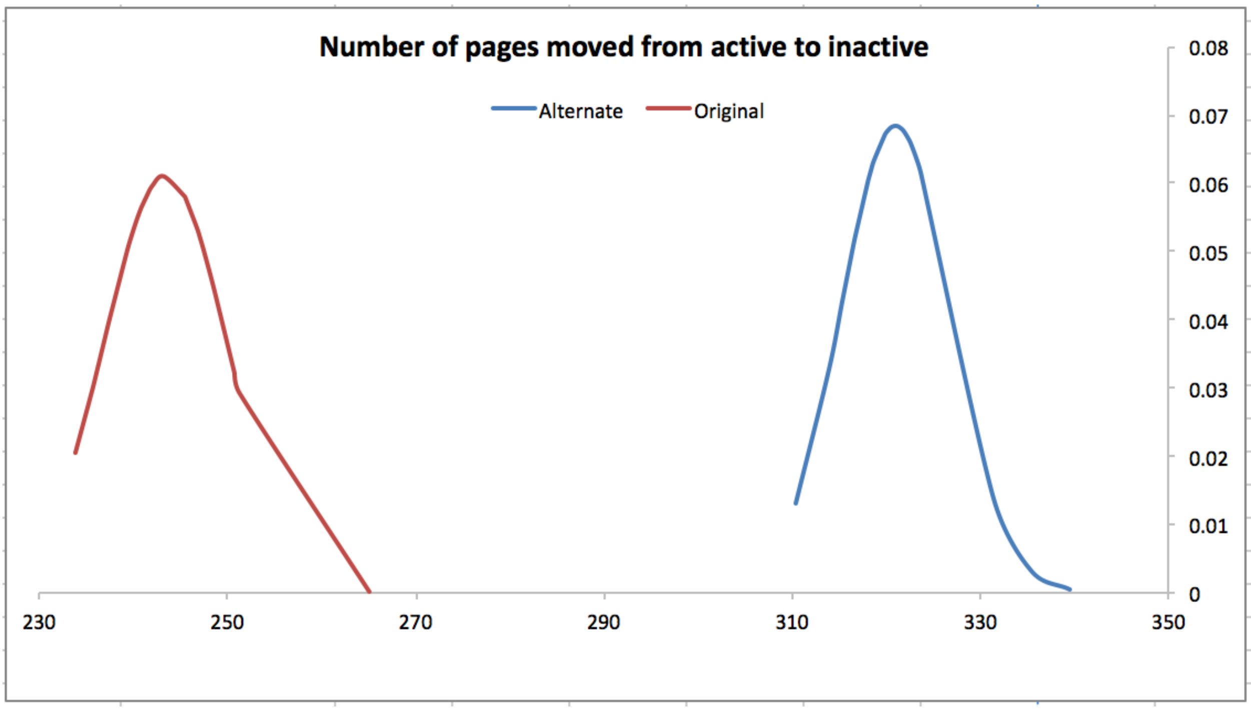
----------------- Number of pages moved from inactive to cache -----------------



	Average	Standard Deviation
Alternate	325.5853317	3.239815429
Original	255.0912825	3.52133843

Our modified algorithm shows a significant increase in the average number of pages moved from inactive to cache and it shows a greater variance as well. This is most likely a direct consequence of a higher number of pages in the inactive queue and a higher number of pages being moved around due to our less efficient algorithm.

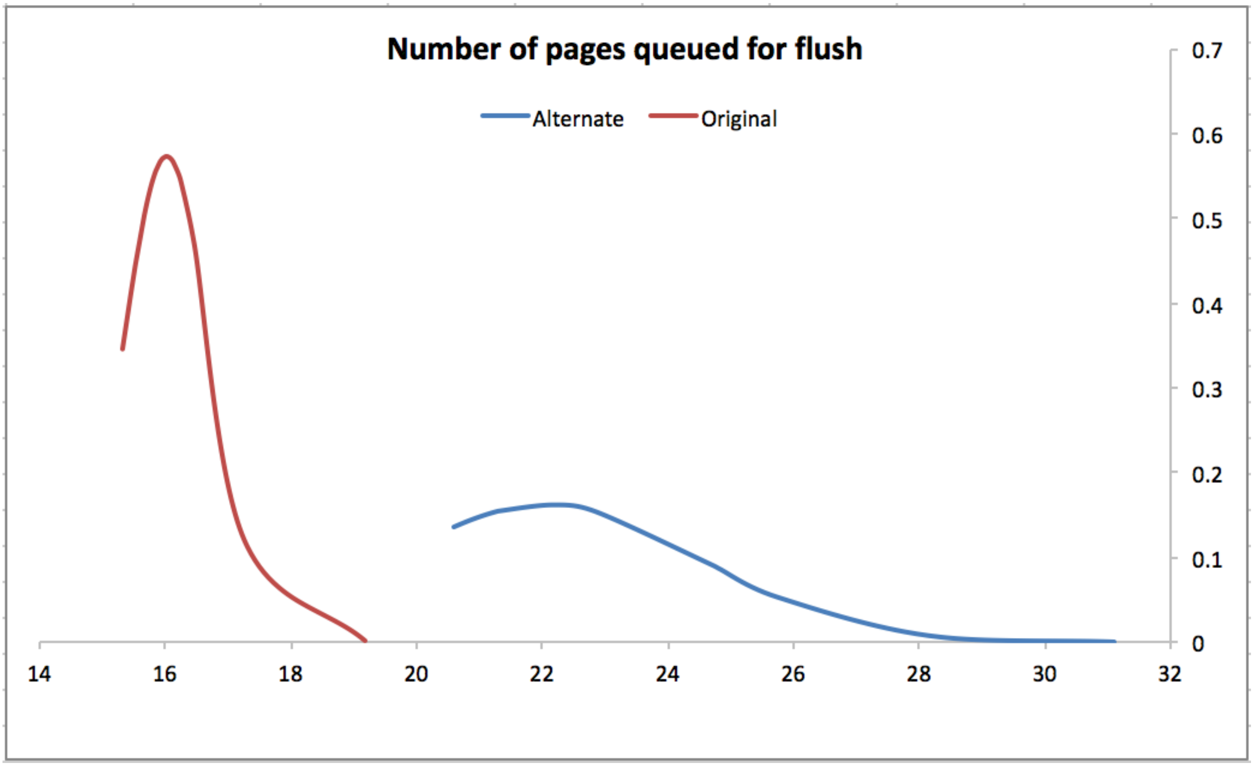
----------------- Number of pages moved from active to inactive -----------------



	Average	Standard Deviation
Alternate	321.0096215	5.826867948
Original	243.3797578	6.531476278

We have noticed that number of pages moved from active to inactive queue in the original FreeBSD is lower than the modified version. Since the activity count is divided by two it forces a page to move from active to inactive queue. In other words, they pages will have less chance to be revisited inside the active queue.

----------------- Number of pages moved from queued for flush -----------------



	Average	Standard Deviation
Alternate	22.06517778	2.46681841
Original	16.03340689	0.696947649

The original algorithm queues less pages for flush and is much more consistent with the number of pages it queues for flush. Our algorithm bonkered up the pages queued for flush because we have a less efficient method for keeping the pages in the active and inactive lists.

Concluding Remarks:

Our modified algorithm produced less desirable results in every category of testing that we ran. We hypothesized that our modified algorithm would be inferior to the default algorithm in FreeBSD. The default paging algorithm is more efficient in its movement of active to inactive pages, inactive pages to cache/free, has less page faults and page reclaims and moves less pages from queue to flush. Our findings support our original claim that the FreeBSD paging algorithm is more efficient than the one we implemented. Thusly, based on findings, our modifications do negatively affect the paging mechanism. Realistically, all of our changes have reduced the efficiency of the paging daemon.