Group Members: Coy, Jason, Timmy
3/2/2016

Testing Methodology

A series of tests were run to determine the differences between the Slim Chance algorithm and the FreeBSD generic algorithm.

Part 1: Creating Memory Stress

To force memory scarcity, we reduced the machine's memory to 256MB. This is very close to FreeBSD's minimum requirement of 64MB. Since the system memory is so low it will force the OS to start paging out memory to disk.

To stress this memory, we ran the *stress* program. This is a FreeBSD package that we installed using *$ pkg install stress*. The following command was executed, which forces the pageout daemon to wake up and begin scanning pages.

```
$ stress-ng -m 1 --vm-bytes 1G
```

- The –m flag spawns N workers that constantly malloc() and free() data (we spawn 1 worker).
- The --vm-bytes flag tells the workers how many Bytes to malloc() (in our case 1GB).

Therefore, the stress command we run will constantly malloc() 1 Gigabyte of memory and consequently free() it on our 256MB system. Since 4x as much memory is allocated than exists in RAM, this forces the system to rely on paging to fulfill the malloc() requests.

Part 2: Collecting the Data

We formatted the logging statements in such a way that they could easily be extracted from the system log and manipulated in a more powerful data processing application (Microsoft Excel). The program gawk allowed us to extract specific parts of the system log. The following command was used to convert our system logs into a csv file (separated onto 2 lines):

```
$ tail -F /var/log/messages | gawk '$6 ~ /^PAGEOUTd/ {print $3 "," $7 "," $8 "," $9 "," \
            $10 "," $11 "," $12 "," $13 "," $14 "," $15 > "testx_log.csv";fflush(stdout);}'
```

Group Members: Coy, Jason, Timmy
3/2/2016

The command *$ tail -F /var/log/messages* prints the log messages as they arrive, and will switch to the newly created log file whenever the old one rolls over (which happens frequently when printing a line each second).

This output is piped into gawk which filters out lines that are not our data lines (we mark these lines with the tag PAGEOUTd). It then separates all of the values in the log with commas and prints them to a csv file.

We also saved the output of the *vmstat* call throughout testing using the following command (separated onto 2 lines):

```
$ vmstat -w 1 -H | gawk '($0 !~ "procs") && ($0 !~ "flt") {print strftime("%H:%M:%S") \
                              "," $4 "," $5 "," $6 > "testx_vmstat.csv";fflush(stdout);}'
```

$ vmstat -w 1 -H prints the output of *vmstat* once every 1 second. The –H flag specifies that numbers should be raw rather than human readable (1024 rather than 1K). This is piped into gawk which filters out the headers and prints a timestamp followed by the avm (active pages), fre(free pages), and flt(page faults) values to a csv file.

Group Members: Coy, Jason, Timmy
3/2/2016

Part 3: Data Analysis

Test 1:

Time to analyze the data! For the first test, we ran the stress test on both the generic and the custom paging algorithm for forty-five minutes. This resulted in about 10,000 outputs from *syslog* and *vmstat*. With a sample size this large, we are reasonably confident that these results are statistically significant. First off, we calculated some averages from the data received from the system log.

RAM: 256MB LOAD: 1GB

	generic	custom
average active scanned:	9138.54	5048.61
average inactive scanned:	1472.91	1638.35
average active:	38300.33	38329.14
average inactive:	3443.13	4080.07
average active->inactive:	753.90	850.51
average inactive->active:	0.49	0.50
average move to cache:	2584	815.27
average flush:	46.77	52.26

Some statistics are about the same, some are little different. Some things to note:

- The number of active pages is the same for both algorithms because they are using ALL memory available
- 45% more active pages scanned with generic algorithm
- 317% more pages moved to cache in the generic algorithm
- The custom algorithm moves 12% more pages from active to inactive

We also charted data from *vmstat*:

RAM: 256MB LOAD: 1GB

	generic	custom
average flts:	1009.71	1084.96
average active:	1871295.7	1872542.03
average free:	2606.25	2456.72

- The custom algorithm triggers 8% more page faults on average

Group Members: Coy, Jason, Timmy
3/2/2016

Test 2:

While these figures seem to prove that the generic algorithm is superior, a second test will be performed on different hardware. This time, our test system will be equipped with 4GB available RAM and *stress* will provide a 4GB load. This is less of an extreme test, but we will find out if generic still reigns superior. Here is another list of averages from excel:

RAM: 4GB LOAD: 4GB

	generic	custom
average active scanned:	107374	61900
average inactive scanned:	33131.15	31436.96
average active:	918746.6	919193.7
average inactive:	47722.9	47074.02
average active->inactive:	913.18	366.60
average inactive->active:	833.27	315.69
average move to cache:	5475	651.61
average flush:	38.14	42.04

Notice the average active pages of Test 2, it is ~900,000. Now look at the same statistic for Test 1, it is a mere ~40,000. This is because Test 2 is run with 16 times as much memory.

Data from *vmstat:*

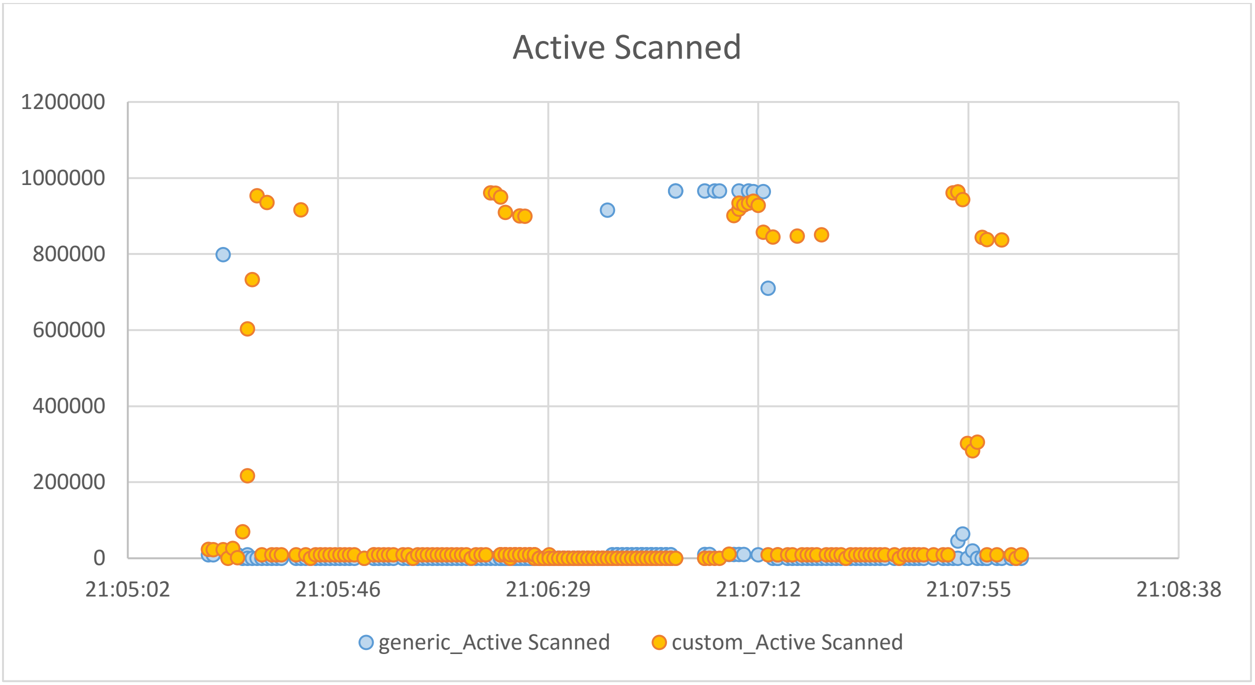
RAM: 4GB LOAD: 4GB

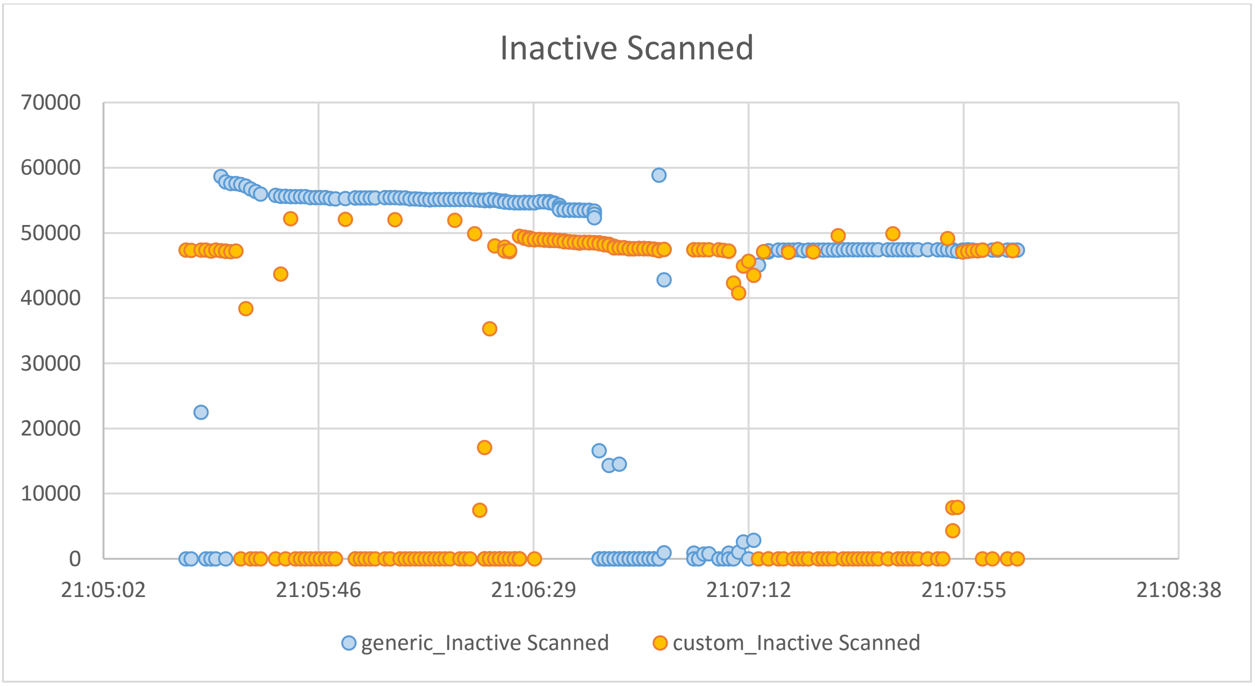
	generic	custom
average flts:	2576.65	2630.64
average active:	5157517.1	5084372.61
average free:	36249.96	40704.85

- With more memory available, the custom algorithm still triggers more page faults.

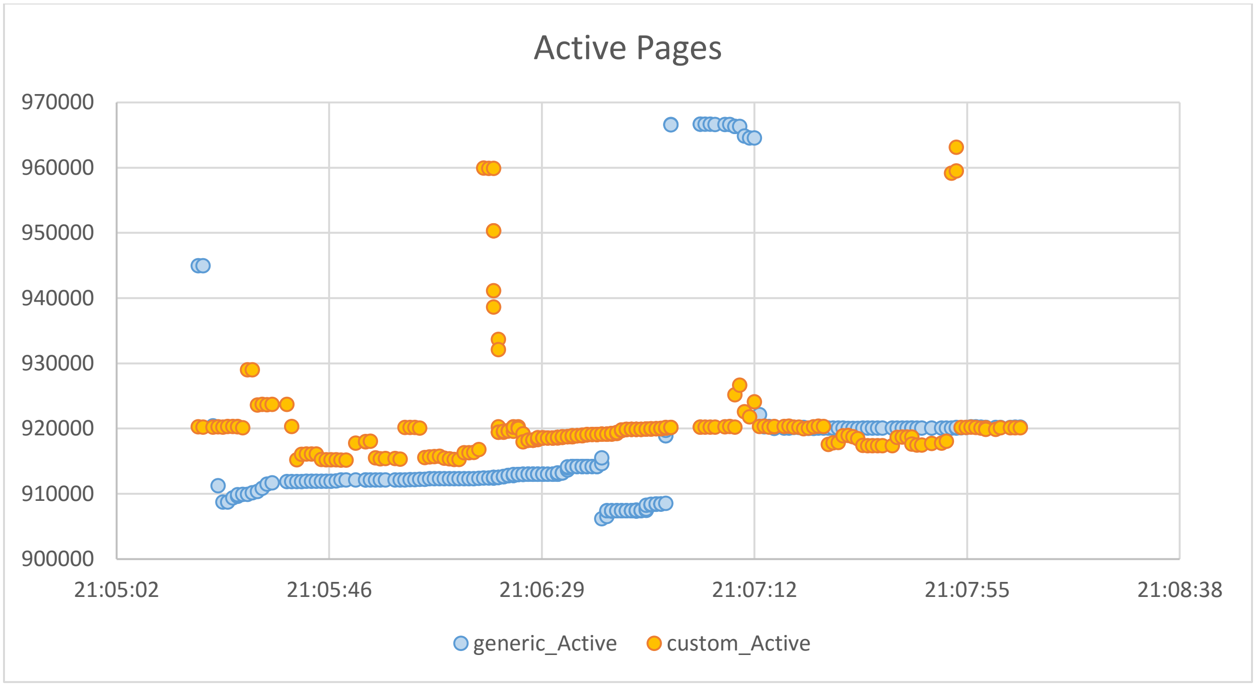
Group Members: Coy, Jason, Timmy
3/2/2016

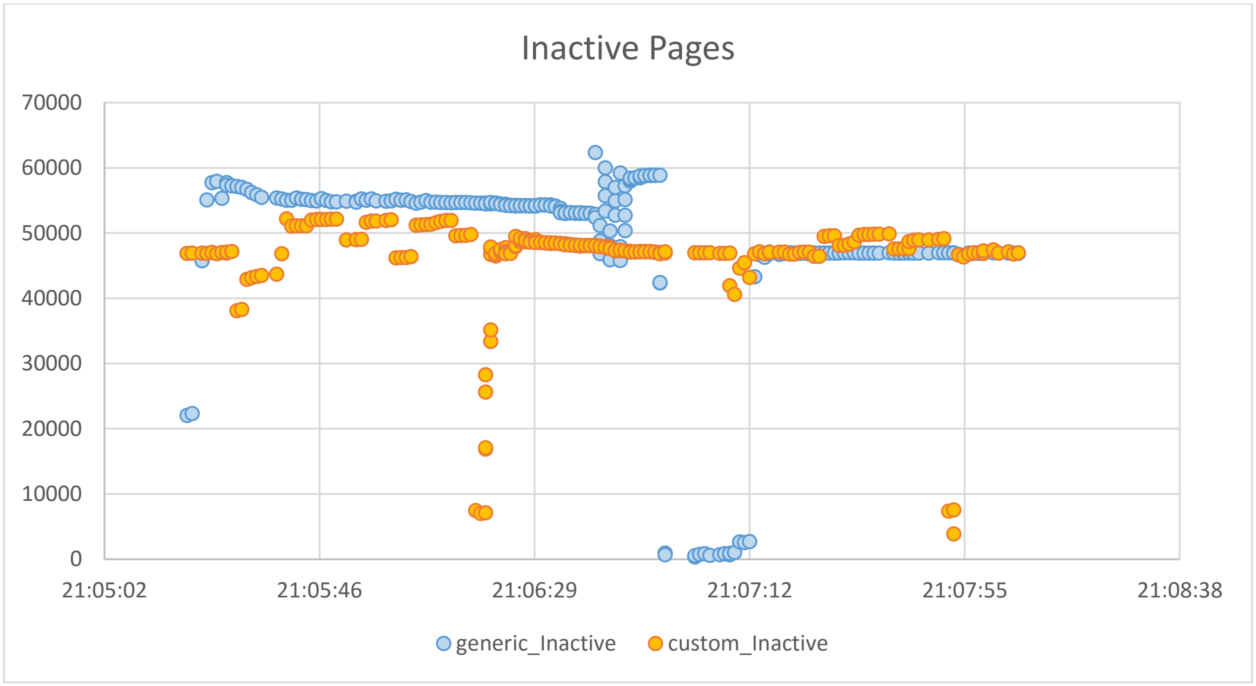
Part 4: Graphs



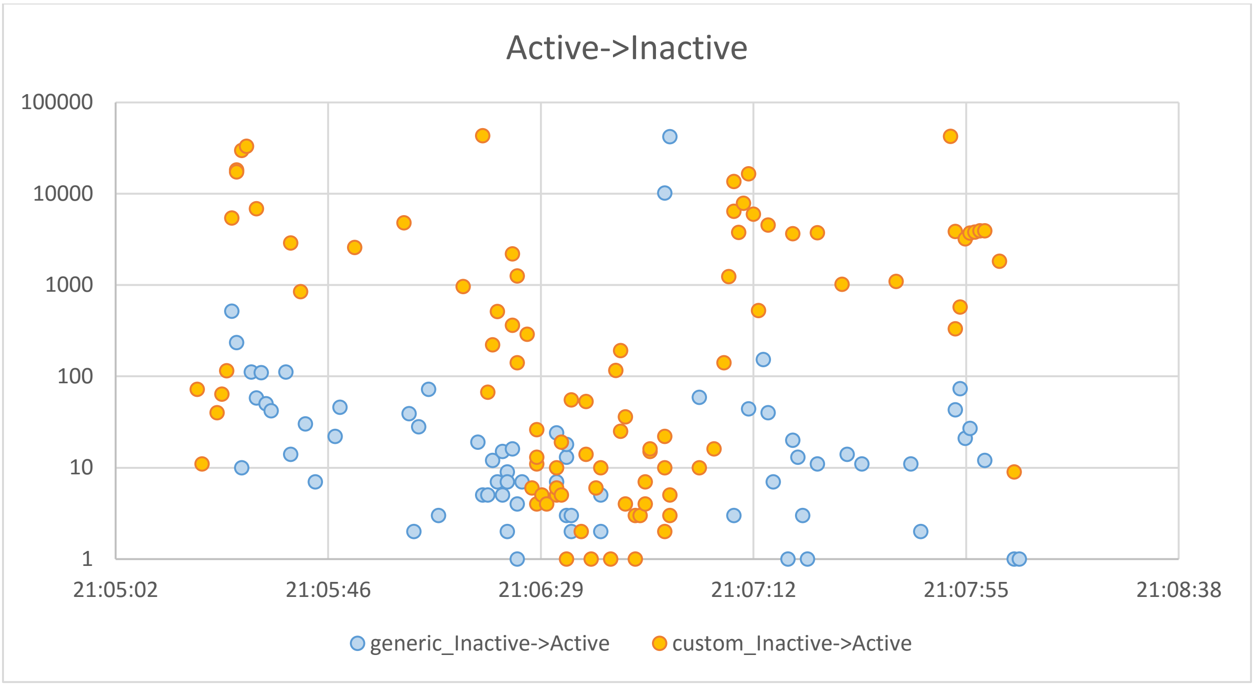


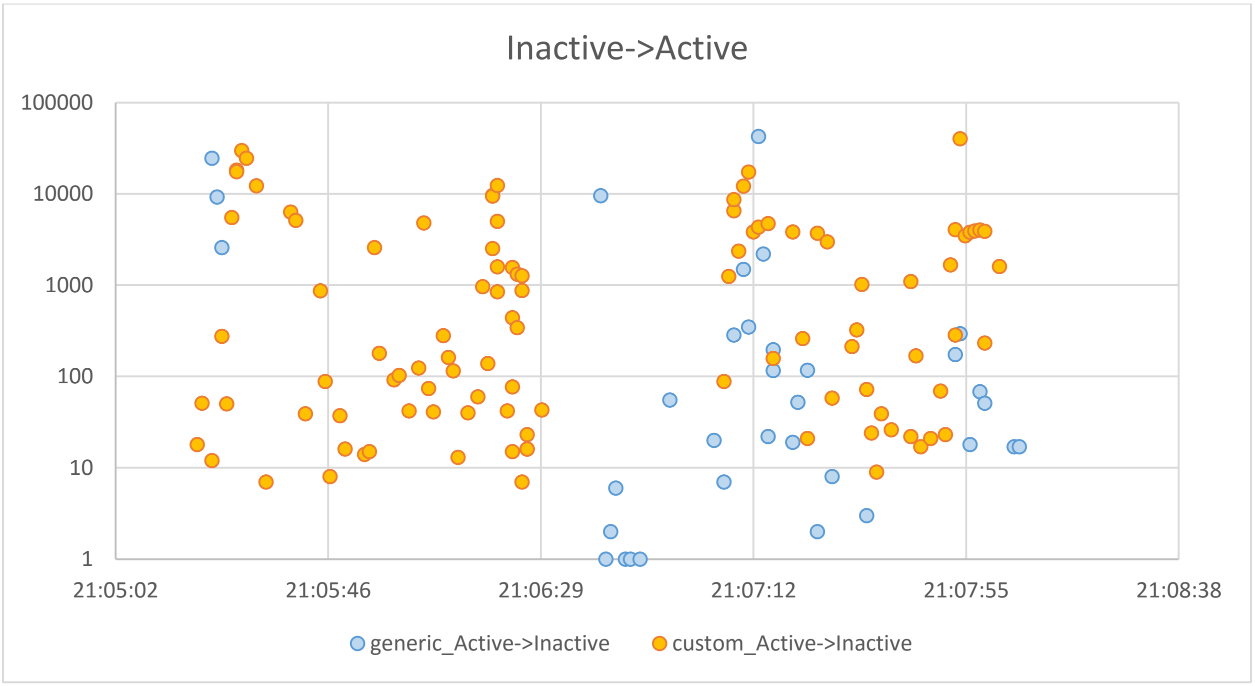
Group Members: Coy, Jason, Timmy
3/2/2016



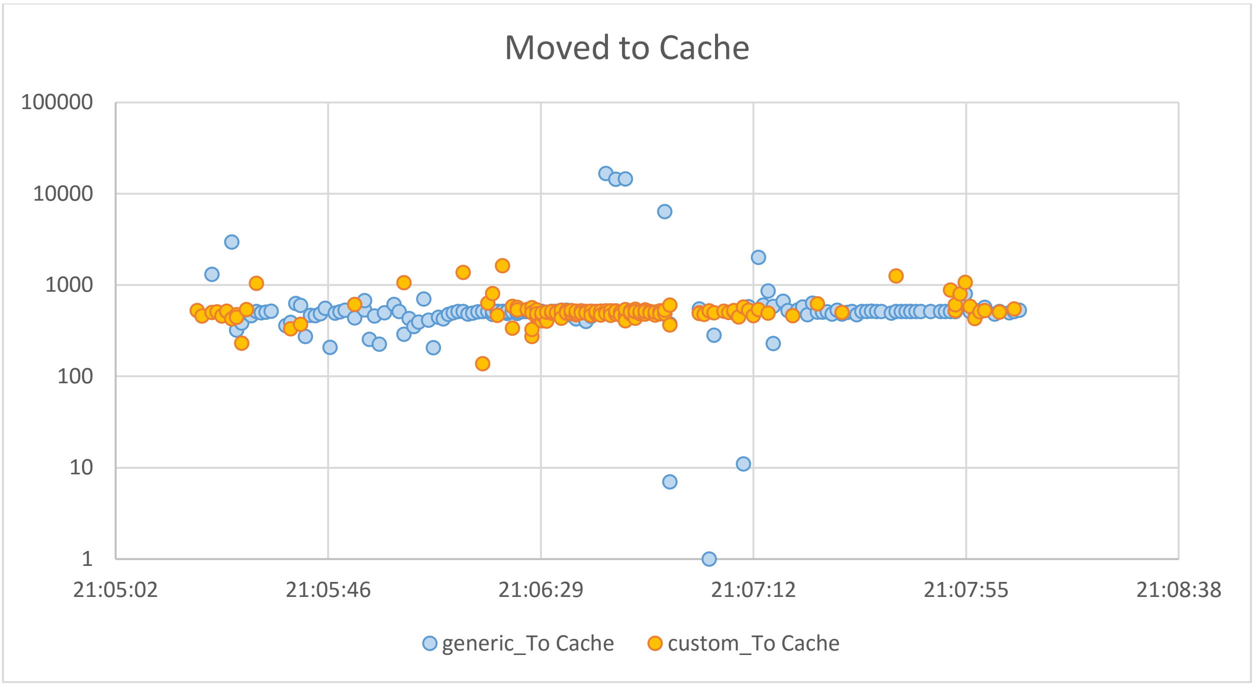


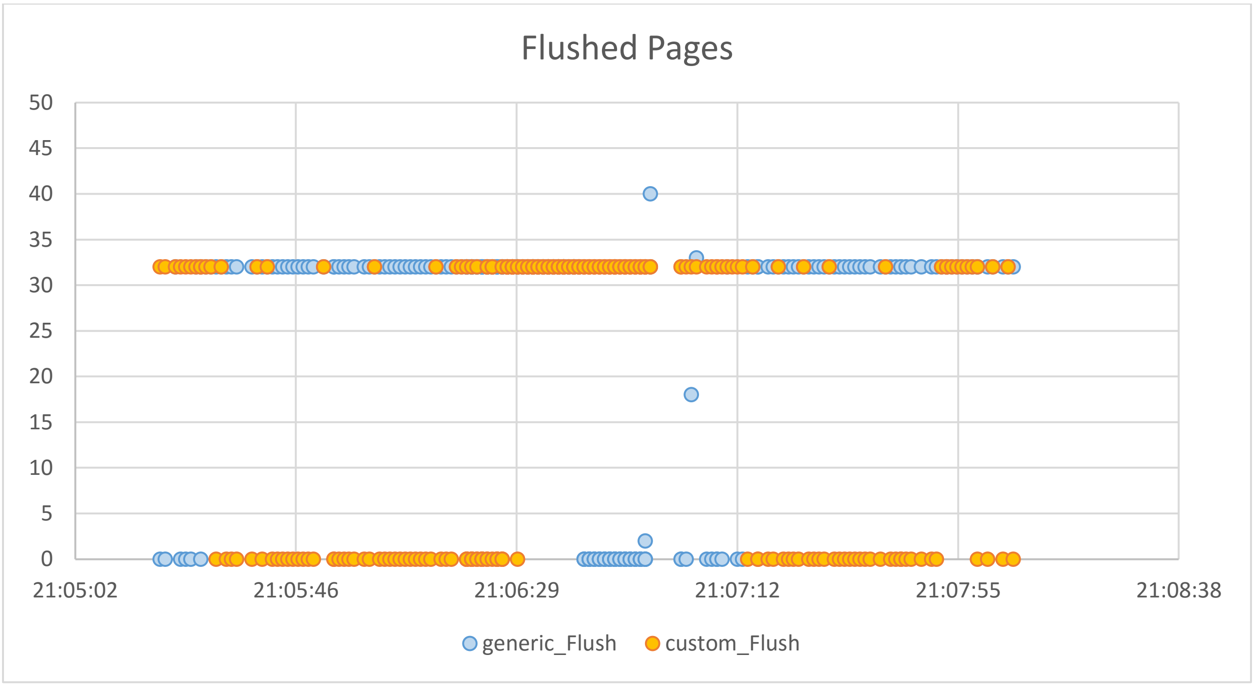
Group Members: Coy, Jason, Timmy
3/2/2016





Group Members: Coy, Jason, Timmy
3/2/2016


Moved to Cache
21:05:02
21:05:46
21:06:29
21:07:12
21:07:55
21:08:38
generic_To Cache
custom_To Cache


Flushed Pages
21:05:02
21:05:46
21:06:29
21:07:12
21:07:55
21:08:38
generic_Flush
custom_Flush