Paging Measured Across Virtualization Software*

Brandon Lara
Louisiana State University
Baton Rouge, Louisiana
blara4@lsu.edu

ABSTRACT

Virtualization remains an integral part of the computing landscape. While RAM, Motherboard buses, and storage have come a long way, the speed of data manipulation is still easily eclipsed by modern Central Processing Units and Graphics Processing Units. Understanding the complex nature of how kernel and virtualization design decisions impact the speed of memory manipulation is crucial to getting the most out of our hardware. Further, companies frequently employ cloud service providers to run virtualized machines handling large data processing tasks such as machine learning. These tasks require a large memory space to operate and thus are dependent upon the memory management system. Further, the specifics of the hardware used is hidden, along with the underlying Operating Systems that service the requests. In this work, I attempt to test the differences in paging performance between Windows and MacOS virtualization using VMWare products (VMWare Fusion on MacOS and VMWawre Workstation on Windows).

In the course of this research, I find that VMWare Workstation shows a steady increase as paging increases. VMWare Fusion showed a much wider variance in the amount of paging occurring under these stress tests. Timed performance benchmarks showed that VMWare Fusion has a far larger variance in increase in slow down relative to the level of paging occurring. A third experiment was conducted to verify these results with data not reported from inside the virtual machine. These results demonstrate the importance of hardware and software decisions related to virtual machine memory management that extends beyond paging algorithms, and highlights the need for research that examines the performance of virtualization software.

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

SOSP '97 El Paso, Texas USA

1. INTRODUCTION

Memory paging has been an ingenious attempt to make the most out of the limited DRAM memory available inside a computer. It has been successful to this end, but its performance has been the subject of many hours of research. Over time, researchers have devised new paging schemes. Some ideas aim to reduce the amount of pages required through data sharing (in the case of library code or data sets) or compression. Other ideas try to ensure as few page misses occur as possible by ensuring that the pages most likely to be referenced are always in memory. In the context of virtualization, three systems are at play. First, the scheme of the Virtualized Opearting System decides which data to keep around. Second, the hypervisor maintains this memory, and third, the Host Operating System maintains the hypervisor's memory. Understanding how paging works, and its performance is a subject of much difficulty to many senior undergraduates here at LSU. Adding the complexity of three systems all with their own memory management schemes, much of which is proprietary and closed source creates a daunting task for any researcher. I aim to side step much of this complexity by taking measurements of the virtual machine's paging rate and performance under paging stress. Specifically, I will be testing Kali Linux's virtualized performance under VMWare Workstation on Windows and under VMWare Fusion on MacOS when subjected to a stress test designed to induce paging.

2. BACKGROUND

When referring to Paging, this paper means the act of swapping out 4K pages of data from main memory to save space. This process allows the computer to functionally have more memory than DRAM space. Some of the infrequently used data can be kept on disk until it is needed. The effect is that Operating System designers have made near limitless promises on the amount of memory program designers can have. This memory granted to program designers is called virtual memory, and consists of some chunks of memory that are in main memory, and other chunks that are in secondary storage. Programs rarely require that much memory at any one time, so storing some of the data in storage until is needed makes a small trade off in time (the time required to handle a page miss and then read in the new page or pages) for increased program flexibility. Large data processing programs can exist because of the virtual memory spaces that far exceed the actual main memory in size.

When a page is not in memory, overhead is incurred by going to get the page in question from storage, read it into

main memory, and potentially decide to kick out another page to make space for the new incoming page. Depending on how often this process occurs, this could be quite costly. Many research hours have been spent on trying to minimize both the frequency and cost of this operation. Windows, for example, attempts to compress pages in main memory before moving them to storage, because undoing the compression is faster than retrieving them from storage. Many researchers have attempted to come up with new schemes for kicking out pages that are unlikely to be needed again for a while. The simplest scheme is LRU, or Least Recently Used. This simply kicks out the page that was least recently used. More complex schemes attempt to make predictions about which pages are likely to be used soon, and kicks out pages deemed unlikely to be used again. Imagine a program that calls the printf function once in the Stdio C library. If the program only calls the function once, the library code is no longer required in memory. This is a somewhat bad example because many programs including the kernel, will likely want the Std C libraries to be in memory, but this is the sort of complex determination about paging that could be made.

Virtualization complicates this picture, because the memory management is now handled by the hypervisor, which in turn makes requests to the host operating system. The paging scheme is still decided by the virtualized operating systems implementation, but overhead is incurred by the translations made by the hypervisor and the host operating system. In this research, I explore this added layer of complexity with a simple benchmark test and attempt to measure this added overhead.

3. MOTIVATION

Computer performance and responsiveness are measured through many metrics. In this research, we are concerned with the performance of the machine to perform large data tasks. The bottleneck of memory based tasks is typically the secondary storage, and being forced to keep some of the Main Memory in storage (IE through paging) means that retrieving it incurs a significant overhead. The amount of paging decides how many times the computer is forced to go to secondary storage to retrieve the needed paging. A higher rate of paging means that the program is waiting for its memory to be paged into RAM instead of continuing to compute and process the data. By quantifying the rate of paging, one could make claims about the performance slow down they could expect from secondary storage overhead.

Additionally, Virtualization software plays a small role in this process. Virtualization software is meant to act as a "Hardware" layer created in software. Hardware assisted paging through the Translation Lookaside Buffer must now be managed by the virtualization software. This means that performance of paging is now closely tied to software decisions. Measuring the performance of a virtual machine under high paging stress across multiple different virtualization software would provide data on the effectiveness of certain virtualization decisions.

Lastly, the Host Operating System must manage the virtualization software. Measuring the performance of the Virtual Machine software under high paging stress could offer insight into how effective the decisions made by the host operating system are to virtualization.

Additional research could aim to further quantify the effect of host OS and virtualization software on Virtual Machine performance by including more tests of more softwares and varied host machines

Table 1: MacBook Specifications

MacOS Monterey Version 12.0.1
CPU
2.3 GHz Intel Core i5
Memory
8 GB 2133 MHz LPDDR3

Table 2: PC Specification

Windows 10.0.19045
CPU 5 GHz Inteli7 12700K
Memory 32 GB LPDDR4

4. THE EXPERIMENT

In this research, I aim to test the effect paging has on performance under different virtualization software. To this end, there are three separate experiments. One measuring the rate of paging under VMWare Fusion and another measuring the rate of paging under VMWare Workstation. The second experiment was done using the same paging stress test, but measuring the speed of writing a large file to storage. The final experiment measures the real world time versus the virtual machine reported time to complete experiment two. The experimental setup is broken up into 4 sections:

- `Virtual Machine`: The Specifications of the Kali Linux Virtual Machine
- `Hardware`: The computers used to Virtualize Kali Linux
- `Code`: The stress test and data collection procedures taking place inside the virtual machine
- `Methods`: The process and precautions taken when performing the experiment

The virtual machine being tested is a standard VMWare image that can be obtained from the official Kali Linux website. The details of the machine will be covered in the next section.

4.1 Virtual Machine

The virtual machine used is an instance of Kali Linux with 2 GB of Memory and 4 processors. The virtual machine is given 80 GigaBytes of memory and runs off an external Solid State Drive. The Kali Linux version is 2022.1 Kali-rolling.

4.2 Hardware

The experiment was run on two separate machines. One is a MacBook running MacOS, and the other is a Custom Built Desktop PC running Windows 10. The full specifications are in Table 1 and 2. The elephant in the paper is the discrepancy between the two machines. It is natural to ask if it is valid to compare these two devices in benchmarking tests. This will be discussed in the methods section, but the

tests and data collected are carefully selected to overcome this issue, and each measurement will be defended regarding this concern. It is important to point out that it is rather difficult (and now impossible due to apple silicon) to get a Mac Machine and a comparable Windows machine. This is a complication that would need to be tackled regardless of the machines used, but I concede that the gap between these two machines is exceptionally large.

4.3 Code

I created a simple stress test that allocates a 0.122072 GigaByte sized chunk of space separated into page sized chunks. The program then scans through each page, writing data to each thus inducing paging:

4.3.1 The Stress Test

```
struct page {
    int buf[1024];
} typedef page;

int main() {
    // 61036 pages to makes 0.122072 GB
    page* virtualMemory = malloc(sizeof(page)*61036);

    int i = 0;
    int j = 0;
    while(1) {
        virtualMemory[i].buf[j]++;
        if (i == 61036) {
            i = 0;
            j++;
            if (j == 1023) {
                j = 0;
            }
        }
        i++;
    }
}
```

The stress test allocates 0.122072 GigaBytes of memory (61036 pages) This is the amount of memory I was able to allocate without causing crashing during my testing. As will be shown in the results section. This is enough memory to induce paging. More memory is actually allocated because the heap requires meta information about the data stored, but the allocation is roughly 0.122072 GigaByte. The program then loops over the allocated pages and writes to each. This induces paging roughly every three operations. Paging occurs for every virtualMemory[i].buf[j]++; execution, but the if statement must be checked each loop, and i must be incremented each loop.

While Kali Linux was subjected to the above stress test, I ran vmstat to collect information about paging. In a separate experiment, I measured the time taken to write a large file (1 GigaByte) to storage.

4.4 Methodology

In each section, I defend the set up and measurements taken from each experiment.

4.4.1 The First Experiment

The first experiment consisted of running variable instances of the stress test above and collecting data from vmstat while they are running. In the experiment, I started by running one stress test for 2 minutes and 30 seconds, collecting vmstat data every 30 seconds. I then repeated this experiment with two stress tests, and so one up to five stress tests. Five stress tests is the chosen limit because at six the virtual machine would crash.

A sample vmstat output from the experiment

```
Entry:
  1994208 K total memory
  1114540 K used memory
   299508 K active memory
  1033252 K inactive memory
   350700 K free memory
    84436 K buffer memory
   444532 K swap cache
   998396 K total swap
   123100 K used swap
   875296 K free swap
     1780 non-nice user cpu ticks
        0 nice user cpu ticks
     2747 system cpu ticks
    64415 idle cpu ticks
      633 IO-wait cpu ticks
        0 IRQ cpu ticks
      541 softirq cpu ticks
        0 stolen cpu ticks
  1006227 pages paged in
   151796 pages paged out
     1338 pages swapped in
    31162 pages swapped out
   330607 interrupts
   523234 CPU context switches
1669456103 boot time
     2264 forks
```

For memory management, we are only concerned with "Pages Paged In", "Pages Paged Out", "Pages Swapped In", "Pages Swapped Out". In this research, I have chosen to only measure pages paged in. The decision is somewhat arbitrary between pages paged put and Paged Paged In. Pages paged out refers to any time that a page is written to disk, and pages paged in refers to any time a page is retrieved from memory. Pages paged in captures routine paging, but also captures when new processes are started as its image must be paged in to start the process. Pages paged out captures routine paging, but also captures when pages are reclaimed because they are no longer used. No processes were created during the tests, and the measurements taken are concerned with rate of pages paged in, and percentage increase as the number of stress tests increases.

4.4.2 The Second Experiment

The second experiment consisted of running variable instances of the stress test (just like the first experiment). While running the stress test, I measure the time taken to write 1 GigaByte of data to storage.

For each test, when increasing the number of stress tests running, the host machine would be powered down and left off for one minute. This is done to ensure that the DRAM is sufficiently cleared. Upon powering on the machine, VMWare would immediately power up the virtual machine, and the next test would begin.

In the first experiment, I am collecting the amount of paging via vmstat. This will be used to calculate the rate

of paging per second. Rate of paging is not affected by the power of the machine. The rate of paging is a measure of how many page misses occur per second. A fast CPU would incur the same amount of page misses, only faster. In the second experiment, I am measuring the time required to write a large file (1 GigaByte) to storage. A faster CPU will perform this task much faster, and thus perform better in this experiment. To accommodate for this fact, I have measured the rate of performance time increase. Rather than measuring the time between these very different machines, we will measure how much worse they get as the paging stress increases. This allows the measurements to be reflective of performance of software rather than of performance of hardware.

4.4.3 The Third Experiment

The third experiment came about during the course of running the two previous experiments. I noticed a discrepancy in the time reported by the virtual machine to complete experiment two and the real world time that the experiment had been running. I conducted a new experiment to test and verify that this discrepancy was real and not human error. I repeated the second experiment three times timing each run with a stopwatch. This creates some level of human error, as I can not start the timer exactly when the test begins, but this research is only concerned with large discrepancies in execution time.

5. RESULTS

The results section discusses the three experiments separately. The first experiment measured the rate of paging. The second experiment measured the performance under increasing paging stress. The third experiment measured real world execution time contrasted with reported virtual machine execution time. In the final section, I discuss the implications of these results together: What they mean for systems testing under virtualization, and how virtualization overhead should be viewed in experimental setups.

5.1 Rate of Paging

5.1.1 VMWare Fusion

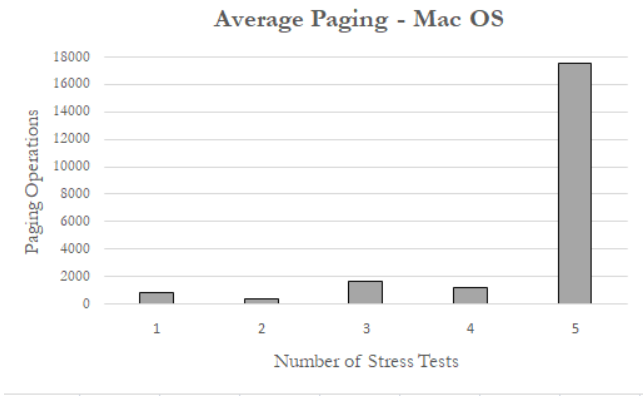

Figure 1:
Average Paging under increasing stress tests on VMWare Fusion

In the first experiment, I measured the rate of paging per second. Each experiment ran for 2 minutes and 30 seconds with 30 second intervals in between paging data collection. Figure 1 shows the paging per second under one, two, up to five stress tests for VMWare Fusion. The difference in paging per second at five stress tests is much larger than the other jumps. This implies that there is some limit where paging starts to greatly increase. Further testing could aim to parse this process and further nail down when and why the amount of paging starts to greatly increase.

5.1.2 VMWare Workstation

5.2 Rate of Paging

5.2.1 VMWare Fusion

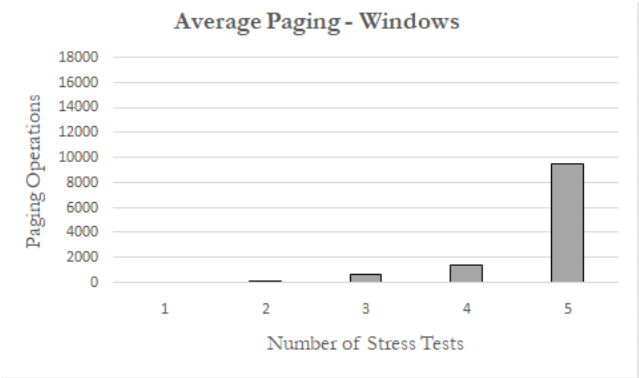

Figure 2:
Average Paging under increasing stress tests on VMWare Workstation

VMWare Workstation showed a similar trend to VMWare Fusion. The measurements from the experiment are seen in Figure 2. The rate of paging per second increased slightly and exploded at the 5th test. The difference is that VMWare Workstation's results show a significantly smaller jump in paging at five stress tests. VMWare Workstation four stress tests to five stress tests showed a 587.452% increase, while VMWare Fusion showed a 1336.65% increase from four stress tests to five stress tests. I believe that this increase is well beyond the variance expected in the rate of paging of standard operation of a Kali Linux Installation.

This is a very surprising result. I did not expect to find much variance in the amount of paging between the two systems. The rate of paging should remain the same as it is decided by Kali Linux. Further research and testing is required and encouraged to determine what accounts for this gap between the two software.

5.3 Performance Under Paging Stress

5.3.1 VMWare Workstation

The performance of the virtual machine was measured under one, two, up to five stress tests. Each test starts a timer and then writes 1 GigaByte to a file and saves the file. The timer stops and calculates the time taken at the end of this process. Figure 3 shows the time taken by VMWare Workstation on Windows. The time shows virtually no increase

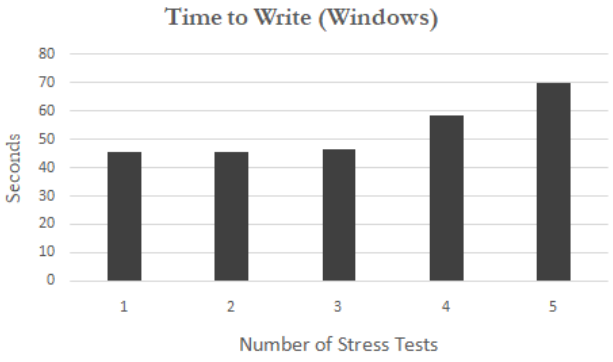

Figure 3:
Time to write 1 GB to storage under increasing stress tests on VMWare Workstation

as stress tests are added until the 4th and 5th stress test. The difference between one stress test and two stress tests was indistinguishable over the course of the experimentation, but there is a large jump at four and five stress tests. The jump in execution time for four stress tests is roughly 12 seconds and again the jump from four stress tests to five stress tests is roughly 12 seconds.

VMWare Workstation seems to perform well under stress, better than VMWare Fusion which showed a larger percentage increase in performance as well as a larger variance in performance across tests.

5.3.2 *VMWare Fusion*

The performance of VMWare Fusion shows a worse performance than VMWare Workstation. Figure 4 shows three graphs. Each one is a repeat of the full experiment. All three are shown because one of the tests was a large outlier. The first test had a significant increase in performance under five stress tests, but the other two experiments showed a less severe increase. It is unclear how this large discrepancy was created, but it is clear that paging performance can vary. To understand the nature of this variation and how wide of a variation can occur, more detailed and focused testing on this specific topic must be explored.

Excluding the outlier, VMWare Fusion shows an exponential increase. As each stress test is added the execution time is increased by a larger and larger amount. VMWare Fusion shows a percentage increase from increased the number of stress tests larger than that of VMWare Workstation. This implies that VMWare Fusion (or MacOS) makes different decisions about virtualization that could be affecting performance. These decisions could also account for the larger variance seen in the experimentation.

Neither figure 3 or figure 4 show an explosion at 5 stress tests the same way that figure 1 and figure 2 show. This might imply that increased paging does not directly affect the execution time of file IO.

5.4 Real World Time

5.4.1 *VMWare Fusion*

During the course of running the experiments, I noticed a difference in the reported run time of the virtual machine when virtualized by VMWare Fusion and the real world time that the experiment had been running. I decided to conduct a small third experiment where I timed the execution with a stopwatch and compared the reported time with the execution time reported by Kali Linux. The data is excluded for brevity, but there was no meaningful difference in reported execution time and real world execution time.

5.4.2 *VMWare Workstation*

The data is excluded, but VMWare Workstation also shows no difference in the reported time and the real world execution time.

6. CONCLUSION

It likely comes at no surprise that virtualization software and host operating system design play a role in the performance of a virtual machine. Still, the complexity of three of the most dense and difficult to understand pieces of software all interacting (Host Operating System, Virtualized Operating System, and HyperVisor) creates an environment that is difficult to analyze. Measurements, like the ones in this paper, help to pinpoint and determine friction points in the design of the three products that could be causing losses in performance. Further, Hardware support for virtualization makes separating hardware from the process even more difficult. When the complexity of a system encompasses two operating systems, chip and circuit design, and a HyperVisor, testing provides a robust avenue of research and can pin point researchers to problems in the design that should be reevaluated. In this research, I showed that VMWare Workstation (and Windows 10) performed better than VMWare Fusion in a narrow paging stress test. Further testing is required to determine how much the Host Operating system, HyperVisor, and CPU design play a role in these results. Finding the source of increased performance for VMWare Workstation could help to improve all systems design in the future.

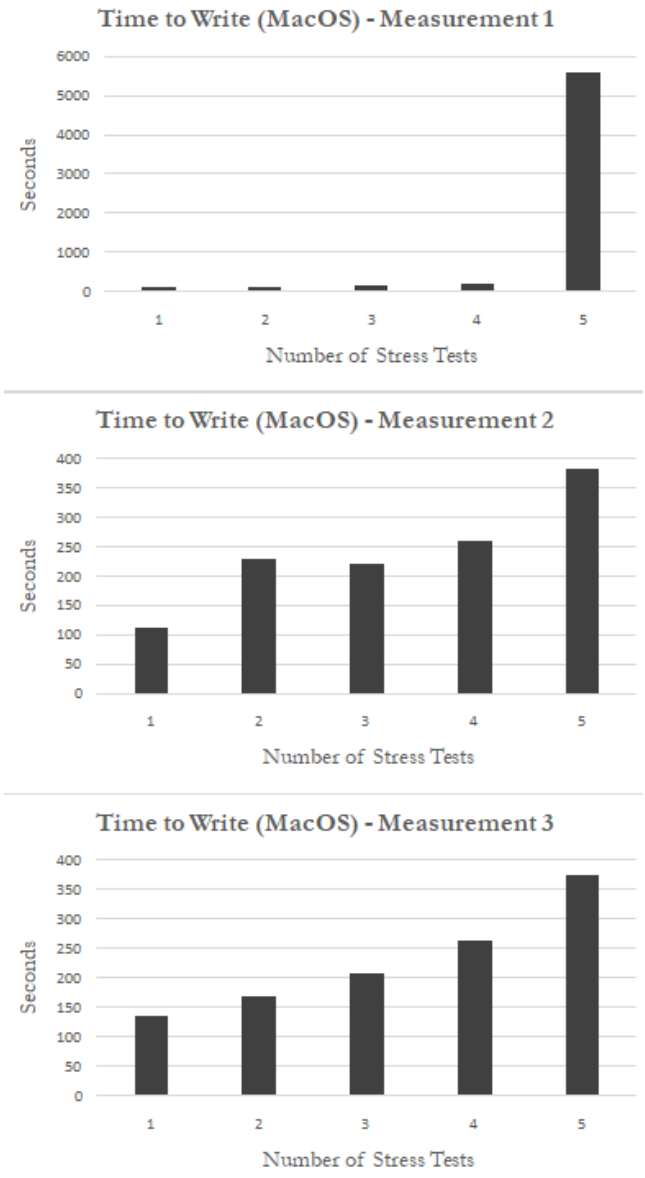

Figure 4:
Time to write 1 GB to storage under increasing stress tests on VMWare Fusion - three trials