To What Extend Does Windows Virtual Memory Affect the Performance of Containerized Applications

Zhen Guan (202191382)

***Abstract*—abstract**

Contents

I. Introduction

Virtual Memory (VM) is a feature found in contemporary operating systems, such as Windows. It shifts inactive memory pages to external storage when the physical memory is limited, allowing the system to handle more intensive workloads without requiring additional memory.

However, because of the substantial difference in performance between modern solid-state drives (SSDs) and high-speed memories, swapping can cause a noticeable reduction in performance. This performance gap is especially significant on devices that use hard disk drives (HDDs).

Containerization technologies like Docker provide isolated runtimes for applications to ensure that the environment is always correctly set up. However, this also means that containerized applications typically require more memory to run.

This paper measures and discusses how Windows VM impact the performance of containerized applications, based on Docker over Windows Containers, in two scenarios:

- Physical memory is (nearly) full and swapping happens frequently,
- Physical memory is sufficient and swapping happens occasionally.

A. Windows Containers

"Windows Containers" is a containerization technology developed by Microsoft that allows applications to be packaged and deployed as containers on modern Windows systems. Just like cgroup-based Docker on Linux, Windows Containers provide an isolated environment for applications to run, with their own file system, registry, and process space.

Docker also provides a Windows version that runs on Windows Containers which does not require CPU virtualization. This paper runs testcases on this version of Docker.

B. Java Microbenchmark Harness (JMH)

JMH is a Java-based framework that provides tools for writing and running microbenchmarks (small tests that measure the performance of individual code snippets or methods). One of the advantages of JMH is that it addresses common issues encountered when benchmarking, such as jitter, warm-up time, and accurate result measurement. JMH provides various benchmarking modes and output formats and is easy to integrate with popular IDEs and build tools.

Developers and researchers use JMH to optimize their Java code and gain insights into Java performance. It can be used to benchmark a variety of Java performance aspects, including memory, CPU, and I/O operations.

This paper uses JMH to measure the performance of the test program running in containerized applications.

II. (Unfinished)

III. Results

TABLE I: macOS Native, Swap frequently

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	69.635	± 2.741
Memory: Sort	avgt	5	1.050	± 0.142
MySQL: 10k Insert	avgt	5	1,952.006	± 30.874
Redis: 10k Insert	avgt	5	190.445	± 5.775
Redis: 10k Delete	avgt	5	184.787	± 4.754

TABLE II: macOS Docker, Swap frequently

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	54.378	± 0.410
Memory: Sort	avgt	5	1.044	± 0.082
MySQL: 10k Insert	avgt	5	10,693.103	± 7,050.951
Redis: 10k Insert	avgt	5	568.259	± 15.851
Redis: 10k Delete	avgt	5	605.792	± 78.055

TABLE III: macOS Native, Swap occasionally

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	68.817	± 1.207
Memory: Sort	avgt	5	1.058	± 0.102
MySQL: 10k Insert	avgt	5	1,985.267	± 215.062
Redis: 10k Insert	avgt	5	189.270	± 8.254
Redis: 10k Delete	avgt	5	181.761	± 6.805

TABLE IV: macOS Docker, Swap occasionally

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	55.153	± 4.105
Memory: Sort	avgt	5	1.004	± 0.154
MySQL: 10k Insert	avgt	5	6,712.840	± 125.200
Redis: 10k Insert	avgt	5	616.555	± 8.441
Redis: 10k Delete	avgt	5	568.797	± 20.346

TABLE V: Windows Native, Swap occasionally

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	193.250	± 4.070
Memory: Sort	avgt	5	1.617	± 0.236
MySQL: 10k Insert	avgt	5	25,114.043	± 3,496.287
Redis: 10k Insert	avgt	5	5,712.345	± 266.694
Redis: 10k Delete	avgt	5	5,539.225	± 820.510

TABLE VI: Windows Docker, Swap occasionally

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	50.112	± 5.183
Memory: Sort	avgt	5	1.388	± 0.120
MySQL: 10k Insert	avgt	5	53,700.293	± 1,081.152
Redis: 10k Insert	avgt	5	2561.764	± 266.891
Redis: 10k Delete	avgt	5	1554.308	± 210.247

TABLE VII: Windows Native, Swap frequently

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	197.929	± 28.401
Memory: Sort	avgt	5	1.762	± 0.242
MySQL: 10k Insert	avgt	5	22,761.322	± 3,923.135
Redis: 10k Insert	avgt	5	5,987.062	± 912.081
Redis: 10k Delete	avgt	5	6,109.372	± 871.125

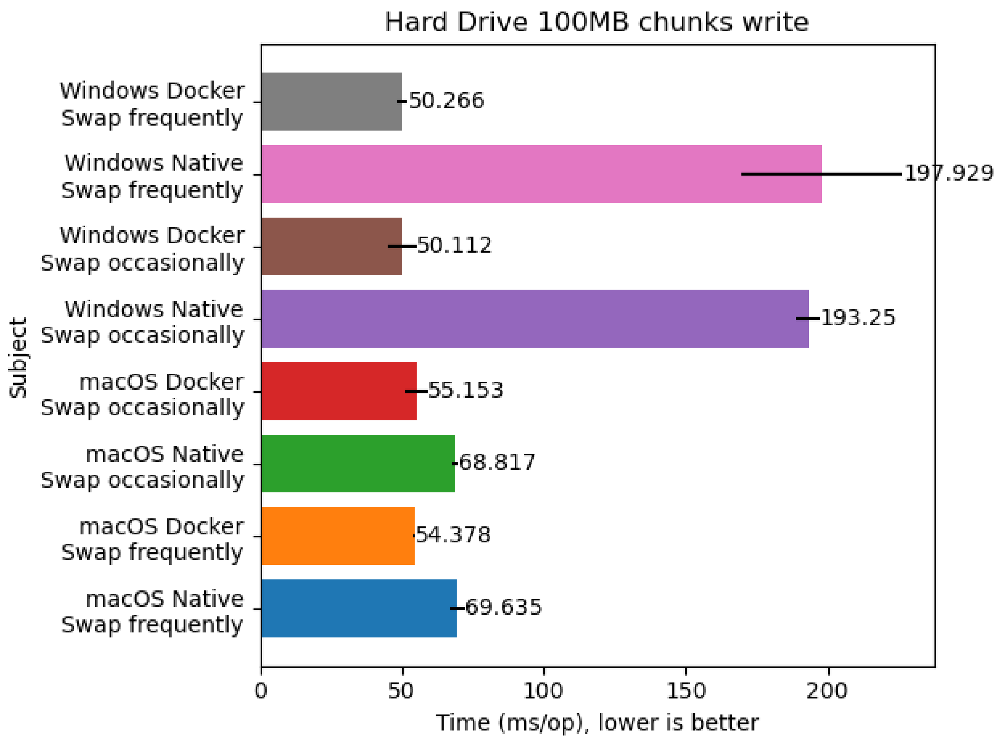

Fig. 1: Hard Drive: 100MB chunks write

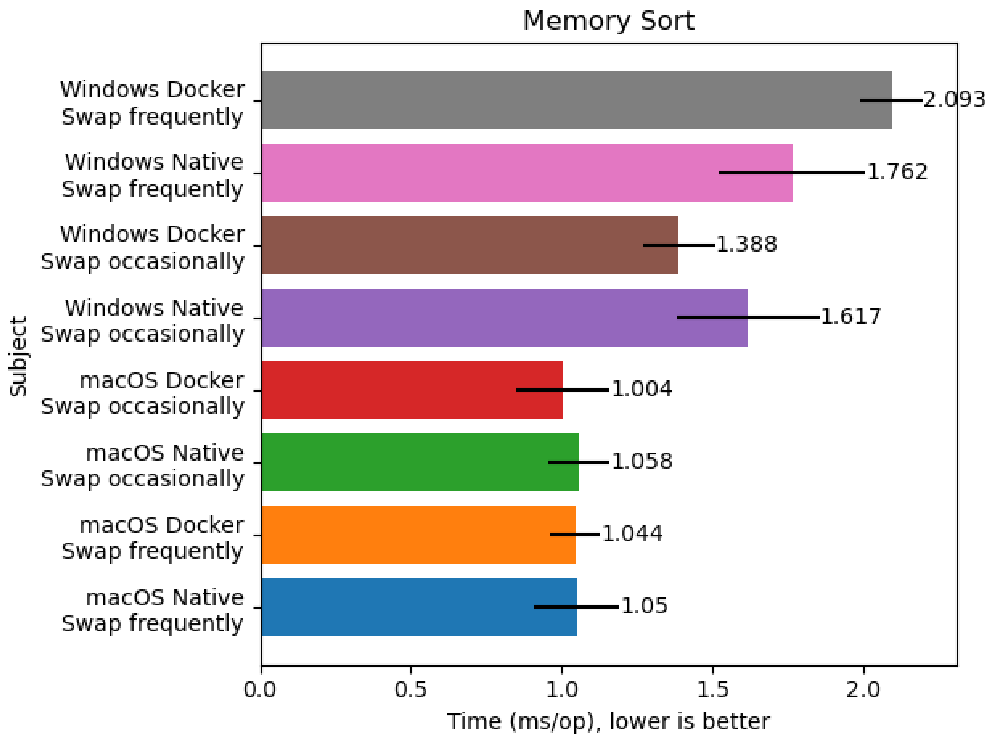

Fig. 2: Memory: Sort

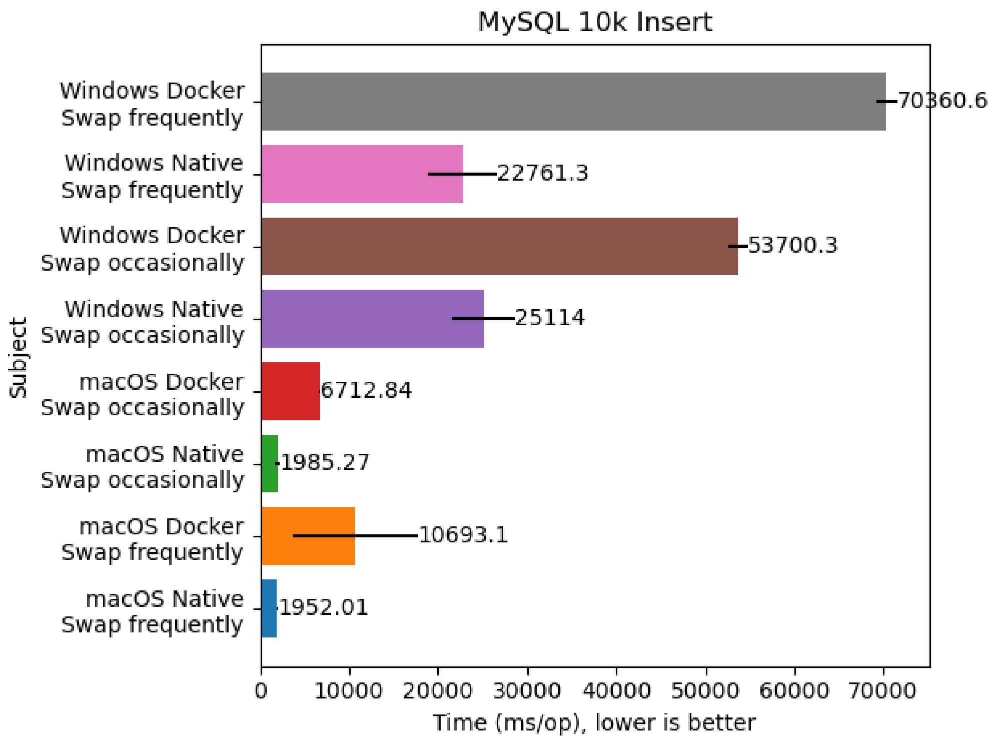

Fig. 3: MySQL: 10k Insert

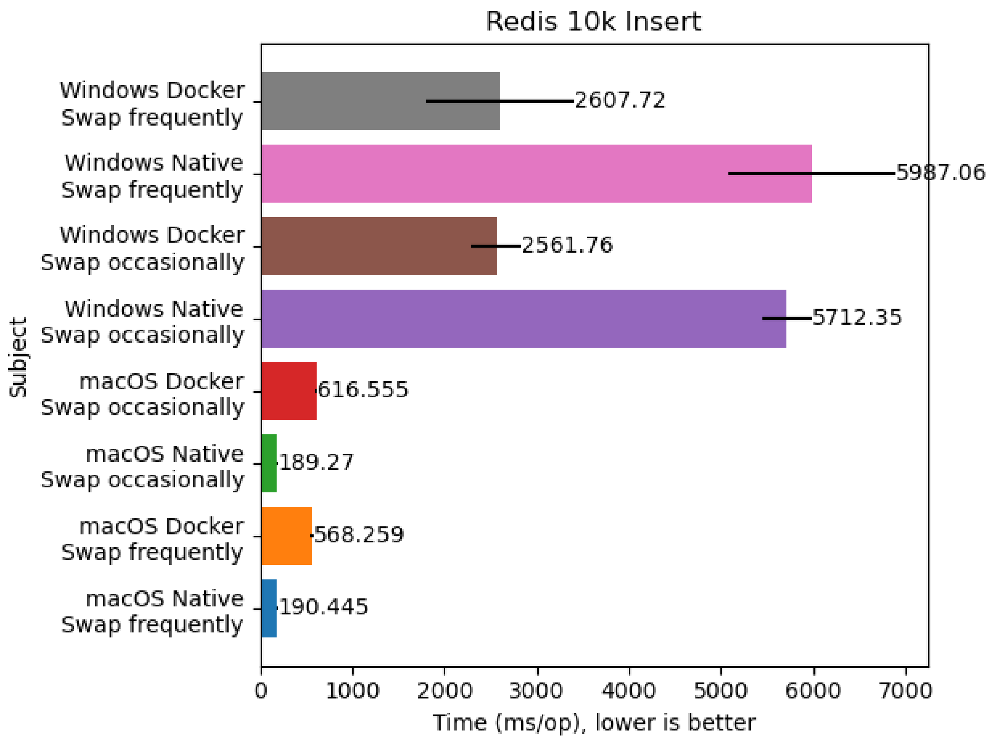

Fig. 4: Redis: 10k Insert

TABLE VIII: Windows Docker, Swap frequently

Subject	**Mode**	**Iter**	**Time**	**(ms/op)**
HardDrive: 100MB Write	avgt	5	50.266	± 1.828
Memory: Sort	avgt	5	2.093	± 0.105
MySQL: 10k Insert	avgt	5	70,360.582	± 1,210.955
Redis: 10k Insert	avgt	5	2607.718	± 801.732
Redis: 10k Delete	avgt	5	1693.477	± 315.247

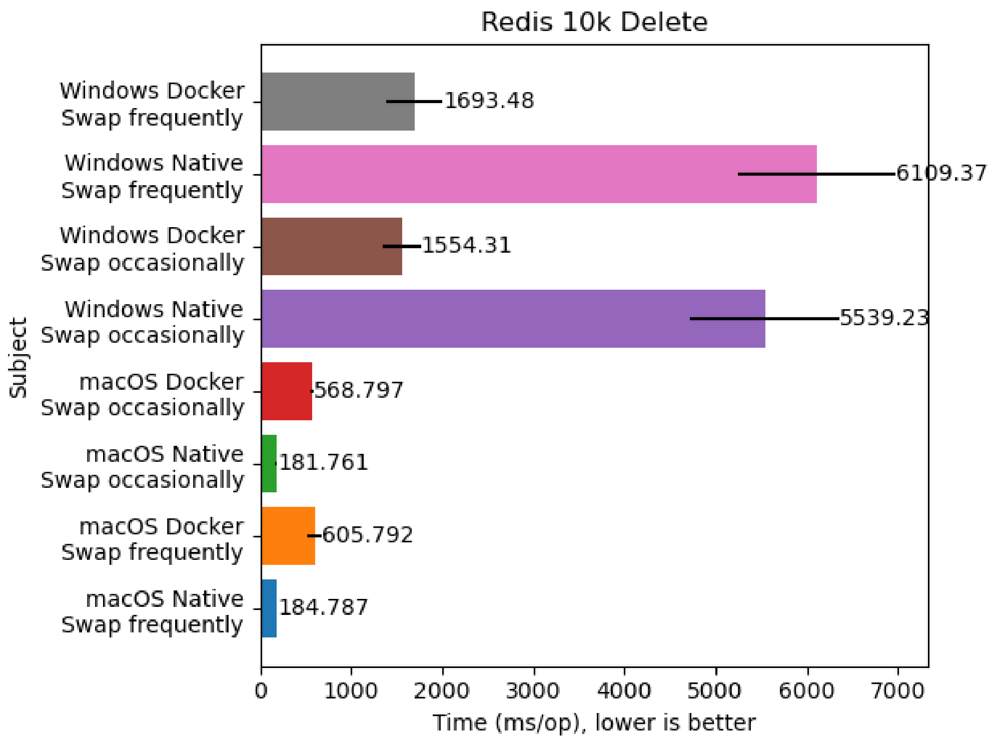

Fig. 5: Redis: 10k Delete