Google I/O Trace Analysis

Vaishnavi Papudesi Babu
vpapudesibabu@hawk.iit.edu
Illinois Institute of Technology

December 4, 2024

Abstract

In designing and optimizing large-scale distributed storage systems, the full-resolution I/O traces are crucial, yet capturing them presents significant challenges. Google introduced Thesios, a methodology for generating synthesized I/O traces using down-sampled data from multiple similar storage servers, claiming high accuracy and applicability across diverse systems. Google has released synthesized traces generated from one storage server with one disk from their storage clusters fostering research on data center's storage system. In this study, we analyze these publicly available traces to evaluate their patterns and assess the transferability of Thesios-generated traces to environments with distinct hardware and configurations. We replicate similar workloads using open-source benchmarking tools and compare the synthesized traces against real-world traces. Based on our assessments, we see that the thesios synthesized traces closely mimic the behavior of normal disks as the latency pattern matches realistic patterns obtained from our experiments, hence showing its transferability.

Keywords: *I/O traces, Benchmarking, Thesios, Transferability, Storage systems, IOR, Darshan.*

1. Introduction

I/O trace analysis is essential for characterizing storage system behavior and informing the design of efficient and high-performing storage architectures. However, capturing full-resolution traces presents significant challenges [1], particularly in complex, large-scale systems with diverse hardware configurations, high volumes of I/O operations and the potential performance overhead introduced by trace collection mechanisms make systematic analysis difficult [2]. Google has recently proposed a methodology called Thesios [3] which can generate synthesized full-representative traces representing one disk using down-sampled I/O traces collected across multiple disks of similar disk characteristics attached to multiple storage servers, addressing the critical challenge of trace collection in complex storage environments. To facilitate research in data center storage systems, Thesios also released synthesized two-month-long traces for a single disk from three Google storage clusters. Although these traces provide an invaluable resource for studying storage behavior, it is crucial to evaluate their applicability and transferability to other systems. Understanding the strengths, limitations, and potential biases of these traces is essential before leveraging them in research aimed at optimizing or emulating storage systems in diverse environments. In this paper, we perform a thorough analysis of the Thesios traces to understand their characteristics and limitations. We also explore the transferability of these traces to other systems by deploying workloads on the disks with different configurations using IOR to generate synthetic workloads and Darshan to capture the resulting I/O traces to assess how well the Thesios traces generalize to diverse storage environments. The analysis and scripts used as part of this project are available in the git repository: CS546-Google-IO-trace

Our key contributions are as follows:

1. We provide an overview of Thesios, IOR, and Darshan to contextualize our study (Section 2).
2. We present a thorough analysis of the publicly available Thesios synthesized I/O traces (Section 3).
3. We design and implement an experimental framework to capture I/O traces, enabling a detailed comparison with Thesios traces to assess their applicability across storage systems (Section 4).
4. We present experimental results demonstrating the transferability of Thesios traces across systems (Section 5).

2. Background

In this section, we provide a brief introduction to Thesios, IOR, and Darshan, as we will extensively discuss about them in the subsequent sections.

2.1. Thesios

Thesios, as briefed in the introduction, Thesios [3] is a methodology designed to synthesize representative and counterfactual full-resolution I/O traces for storage devices and servers in distributed storage systems. Thesios enables accurate "what-if" analyses in the data centers by allowing them to test potential hardware or policy changes without disrupting production systems.

The Thesios methodology begins by collecting I/O operations on files at the storage server level leveraging a sampling service. It then groups similar servers and disks based on disk characteristics to produce representative traces. These traces are synthesized using a trace synthesizer, which combines the sampled data into high-resolution output representing single disk. The synthesizer also adapts to workload fluctuations over time, such as hourly or daily variations, by dynamically adjusting the traces it synthesizes. A trace reorganizer is used optionally to transform synthesized traces for higher-level entities into traces for lower-level entities to account for disk utilization and operational overhead such as queuing delays, burstiness, spatial and temporal locality of accesses and request reordering. This reorganizer allows synthesized traces to reflect the operational realities of lower-level storage entities without needing direct, high-resolution sampling at the disk level. It thus reduces sampling overhead while maintaining trace fidelity. To promote research in data center's storage systems, Google has released synthesized data captured from 3 different storage clusters which we will thoroughly study and analyze in section 3.

The method claims high accuracy in its synthesis, particularly achieving 95-99.5% precision in request counts, and 90-97% accuracy in disk utilization demonstrated through evaluations in real-world Google data centers.

The limitations of thesios include the inability to model feedback loops where user behavior changes due to system optimizations, imprecision in capturing tail latency for extreme cases, and challenges in accurately evaluating burstiness in smaller time frames. However, the current prototype cannot fully evaluate the impact of hardware changes on write operations and requires adjustments for simulator parameters like system overhead modeling. Thesios is flexible and generalizable to other domains, such as key-value stores or memory systems, but its application outside the storage server/disk context is unvalidated.

2.2. IOR

IOR [4] is a parallel I/O benchmark that can be used to test the performance of parallel storage systems using various interfaces and access patterns. IOR uses MPI for its execution on multiple nodes, it is possible to control the number of processes each node will use. IOR supports many backend drivers that can be selected by the IOR API (-a) option. The default API is POSIX.

An IOR file is organized as a sequence of segments that represent the application data, each processor holds an evenly divided part of the segment called a block. The process with rank 0 gets the first block and the process with rank 1 gets the second block and so on. The parallel I/O layer re-assembles each processor's block into a single segment as viewed in an IOR file. Each block is further divided into transfer units which can be useful in emulating strided access patterns. The transfer unit chunks directly correspond to the I/O transaction size, which is the amount of data transferred from the processors memory to disk for each I/O function call.

2.3. Darshan

Darshan [5] I/O characterization tool is a lightweight HPC I/O characterization tool that instruments HPC jobs and collects their real-time access pattern tracing and profiling developed by Argonne National Laboratory. It can collect I/O information from various I/O libraries such as HDF5, MPI-IO, and POSIX I/O and helps to understand I/O characteristics of application including I/O access patterns, sizes, number of operations, etc. To trace the detailed I/O information, the Darshan eXtended Tracing (DXT) tool can be enabled to collect the access pattern such as, process ID, accessed file offsets, request size, and I/O timestamps. DXT first stores the access pattern of an individual file in memory and then writes it to a log file after the application finishes its execution.

Darshan has two components, darshan-runtime and darshan-util. Darshan-runtime [5] to collect the performance data on a target system and Darshan-util[4] to analyse the data collected by the darshan-runtime. Darshan runtime library initializes itself by intercepting MPI_Init(), and creates the Darshan log file including the job metadata and at application shutdown time, Darshan core intercepts MPI_Finalize() and collects all the instrumented data to be written to the log file in the path defined in the darshan configuration time.

Darshan-util has a collection of tools for parsing and summarizing log files produced by the darshan instrumentation module. The command line utility darshan-parser can be used to obtain a human-readable, text- format output of all information contained in a log file. Darshan-job-summary tool helps generate PDF summary report of the entire log file. More information on other utilities can be found in the Darshan-util documentation.

3. Thesios data analysis

In this section, we analyse the synthesized I/O traces shared from Google's storage servers using the Thesios methodology. Before analysing the traces, it is important to understand the storage servers from which the traces have been captured. While the exact hardware specifications of the storage servers are not disclosed for privacy reasons, it was mentioned that the underlying storage system comprises a hybrid infrastructure of Hard Disk Drives (HDDs) and Solid-State Drives (SSDs), with SSDs functioning as a caching layer. The I/O samples were captured using RequestSampler, an existing telemetry system that records remote procedure calls (RPCs) from various applications serviced by Google's distributed storage system.

It is important to note that these traces are collected at the storage server level. While capturing the traces, the traceReorganizer (server simulator) preserved the original latency for cache hits and adjusted latency only for cache miss scenarios (disk reads) simulating the disk-level traces. All writes and some reads were fulfilled using the server's writeback buffer cache.

3.1. Thesios Trace structure and fields of interest

The synthesized I/O traces include various fields that capture detailed I/O operations, such as: **filename, file_offset, application, c_time, io_zone, redundancy_type, op_type, service_class, from_flash_cache, cache_hit, request_io_size_bytes, response_io_size_bytes, disk_io_size_bytes, start_time, disk_time, simulated_disk_start_time, simulated_latency.**

For the purpose of our analysis, the following fields were of primary interest:

1. io_zone: Categorizes data as WARM, COLD, or UNKNOWN.

2. op_type: WRITE and READ operations.

3. service_class: Identifies request priority with categories such as THROUGHPUT_ORIENTED, LATENCY_SENSITIVE, and OTHER.

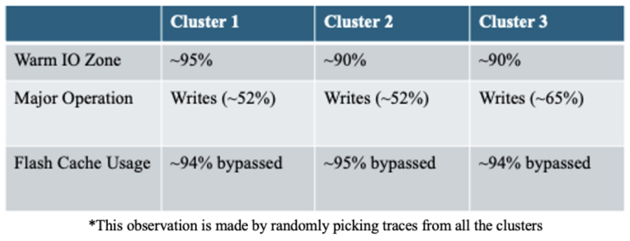
4. from_flash_cache: Indicates if the request was served from flash cache (1 for yes, 0 for no).

5. cache_hit: Whether the request is served by server's buffer cache : -1 for writes, 0 for cache misses, and 1 for cache hits during reads.

6. request_io_size_bytes and response_io_size_bytes: Represent the size of I/O requests and responses, respectively.

7. simulated_disk_start_time: Reflects the simulated start time of disk-level operations for cache misses.

8. simulated_latency: Simulated latency for disk-level operations in the event of a cache miss.



	Cluster 1	Cluster 2	Cluster 3
Warm IO Zone	~95%	~90%	~90%
Major Operation	Writes (~52%)	Writes (~52%)	Writes (~65%)
Flash Cache Usage	~94% bypassed	~95% bypassed	~94% bypassed

*This observation is made by randomly picking traces from all the clusters

Figure 1. Comparison of IO zone, operation and flash cache use in clusters

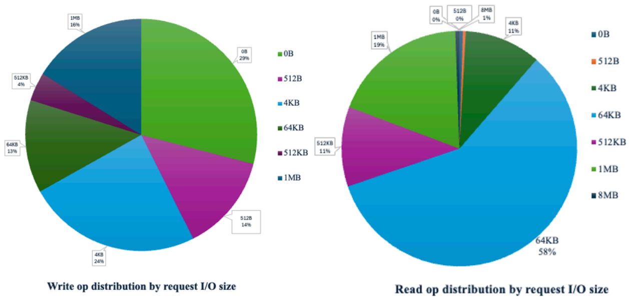


Figure 2. IO size distribution by operation type

3.2. Traces Analysis

The simulated I/O traces were generated over a 60-day period. The traces were captured from three clusters, each representing a largest disk category in the storage cluster. In each cluster, approximately 100 trace files are available with each file containing 40MB of data, amounting to 240 GB of data per cluster for the entire period. This comprehensive dataset offers insights for analysing workload characteristics. Fig. 1 shows the approximate spread of data in terms of IO Zone, major operation and flash cache usage. Though each cluster represent separate disk, it is worth noticing that the I/O operations on each cluster are write dominant and the files accessed belong to warm zone as well as they were not served from flash cache.

For further analysis, we selected cluster 1 which represents one disk whose requests belonged to throughput oriented service class(75%). To analyze the write and read distribution from the traces, we have picked 10 samples a day randomly from the traces, so a total of 600 samples that were loaded into a data frame. The traces contain 1,040,613 different request IO sizes ranging from 0 Bytes to 9MB. To simplify analysis, we created a new column called 'request_io_size_bucket' to group these request sizes into predefined buckets of 0 B, 512 B, 4 KB, 64 KB, 512 KB, 1 MB, 8 MB. Each request IO size was assigned to its nearest corresponding bucket for further analysis.

From the fig.2 , we can see that the requests are reasonably spread into these buckets. Notice that 29% of the write requests are of size 0 bytes, which we expect to be for the

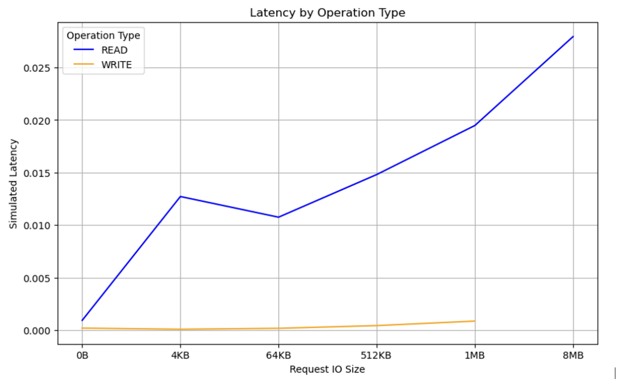


Figure 3. Latency by IO size

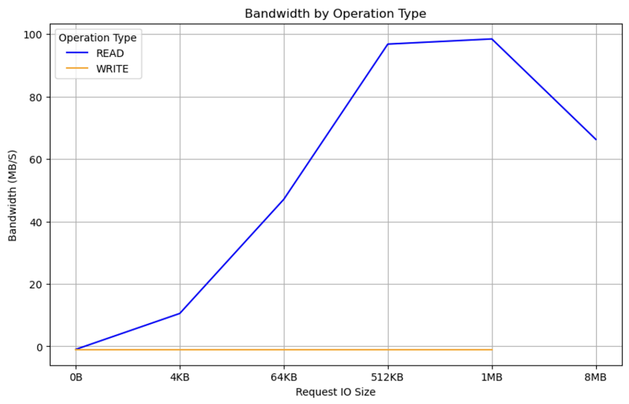


Figure 4. Bandwidth by IO size

traces that update file metadata. We also observed that from the overall traces, around 15.54% of requests are on the 0 byte size. From the request size spread, we could see read and write requests are more on smaller IO sizes specifically under 64KB compared to larger IO sizes.

3.3. Thesios Latency Analysis

The latency of the IO requests by request IO size in the fig. 3 shows the latency graph for the read and write operation types. The latency for write operations are comparatively very less because all the write operations are served by server's buffer cache. The latency on IO requests of read type increase with the increase in IO size reasonably.

3.4. Thesios Bandwidth Analysis

We have calculated bandwidth based on request IO size and the disk time parameters obtained from the traces. The fig 4 shows the bandwidth spread for read as well as write requests separately. The write request bandwidths are always -1 because all the write operations are performed on the server's buffer cache only. The read request graph demonstrates significant scaling with the increase in IO sizes. In conclusion, the synthesized traces closely resemble real-world scenarios, showcasing a fairly distributed range of IO sizes and latency behavior that aligns with a typical disk performance. Latency increases with increase in IO size, reflecting expected behavior of most disks. This suggests that the traces effectively mimic realistic workload patterns. However, it is worth noting that the traces contain relatively less requests at larger IO sizes like 8 MB.

4. Our experiment Design and Implementation

The primary objective of this study is to compare the performance, latency, and bandwidth of Thesios disk against multiple storage devices including Non-Volatile Memory Express (NVMe), Solid-State Drive (SSD), and Hard Disk Drive (HDD). This comparison will help identify correlations in performance metrics and provide insights into the similarity and variability across different storage systems. As a result of this experiment, we aim to evaluate the transferability of I/O traces provided by Thesios to other storage environments. The performance of a storage system vary significantly based on its hardware configurations, making it a critical factor in an experimental design. While the hardware specifications of the disks where Thesios I/O traces were captured are unknown, we know that it originated from the largest disk available on the storage server, which is an HDD. In this study, we utilized multiple storage devices, including NVMe, SSD, and HDD, to benchmark and analyze their performance to ensure we perform a robust comparison of storage disks against thesios disk. NVMe is known to deliver the highest speeds, followed by SSD, and finally the slower HDD.

4.1. Libraries required

The libraries required for this benchmark include Python, IOR (version 3.3.0), GCC Runtime (version 11.3.0), and MPICH (version 4.1.1), Darshan runtime and Darshan utility (version 3.4.6) to analyze and profile I/O performance. Python was utilized for working with PyDarshan, a component of the Darshan utility. The IOR benchmarking tool relies on GCC for testing, and since IOR is run with MPI, the MPICH library is required. These tools and libraries were essential for conducting the experiments and analyzing the I/O traces effectively.

4.2. Test Methodology

This experiment begins with enabling Darshan in the environment and running IOR with MPI to generate the test workloads. After the execution of MPI is complete, Darshan logs are generated at a predefined log path, from which the I/O traces are extracted. The detailed steps of the methodology are as follows:

4.2.1 Enabling Darshan

Darshan-runtime, an external library installed on the server, is used to capture detailed I/O traces. To enable Darshan, the following environment variables are set:

1. LD_PRELOAD=/path/to/libdarshan.so - Preload the Darshan library for capturing I/O traces.

2. DARSHAN_DEBUG=1 - Enable debug mode for Darshan.

3. DXT_ENABLE_IO_TRACE=1 - Enable the Darshan eXtended Tracing (DXT) module to capture detailed I/O information.

Darshan automatically begins capturing traces when the MPI execution starts and continues until the MPI_Finalize() function is called. After the workload generation is complete, the LD_PRELOAD environment variable is unset to ensure no other MPI traces are captured beyond the scope of the experiment.

4.2.2 Generating Workloads Using MPIRUN and IOR

To generate test workloads, the following modules are loaded on the experimental node:

1. IOR: Used for generating workloads with configurable file I/O patterns.

2. GCC-Runtime: Provides the compiler runtime support.

3. MPICH: Executes the workloads with MPI.

Different combinations of IOR parameters such as block size, segments, and transfer sizes are used to simulate challenging workloads. The workload configurations should align with the file sizes and patterns observed in Thesios data (as discussed in Section 3) in order to make a clear and unbiased analysis. Workload generation will be performed using MPI and IOR commands. For each request size in the predefined list [512 B, 4KB, 64 KB, 512 KB, 1 MB], a different combinations of transfer size, block size, segment should be performed to make the benchmarking more challenging. All steps, including loading parameters, libraries, and executing workload generation scripts, are automated for this experiment.

4.2.3 Processing Darshan Logs

Once the MPI workload execution is complete, Darshan logs are generated at a predefined log path. These logs are processed using darshan-parser to extract the I/O traces for further analysis. From the darshan log, we particularly look for the below POSIX traces and fields.

Write I/O Operation:

1. POSIX_BYTES_WRITTEN: Total bytes written.

2. POSIX_F_WRITE_START_TIMESTAMP: Timestamp of the first write operation.

3. POSIX_F_WRITE_END_TIMESTAMP: Timestamp of the last write operation.

4. POSIX_F_WRITE_TIME: Cumulative time spent on POSIX write operations, including fsync and fdatasync.

Read I/O Operation:

1. POSIX_BYTES_READ: Total bytes read.

2. POSIX_F_READ_START_TIMESTAMP: Timestamp of the first read operation.

3. POSIX_F_READ_END_TIMESTAMP: Timestamp of the last read operation.

4. POSIX_F_READ_TIME: Cumulative time spent on POSIX read operations, including fsync and fdatasync.

Latency and Bandwidth calculation: The extracted POSIX traces are used to compute key performance metrics for analysis. The formulas for calculating latency and bandwidth are as follows,

$$\textbf{Read I/O Latency} = \text{POSIX\_F\_READ\_END\_TIMESTAMP} - \text{POSIX\_F\_READ\_START\_TIMESTAMP} \tag{1}$$

$$\textbf{Write I/O Latency} = \text{POSIX\_F\_WRITE\_END\_TIMESTAMP} - \text{POSIX\_F\_WRITE\_START\_TIMESTAMP} \tag{2}$$

$$\textbf{I/O Bandwidth (in MB/s)} = \frac{\text{POSIX\_BYTES\_READ}}{1024 \times 1024 \times \text{POSIX\_F\_READ\_TIME}} \tag{3}$$

5. Evaluation

In this section, we discuss the experiment analysis comparing the latency and the bandwidth across three disks mentioned in section 4 against Thesios disk. We first present the test environment configuration details followed by experimental analysis.

5.1. Test environment configuration

For our experiments, we used nodes in ares cluster that are configured with an Intel(R) Xeon(R) Silver 4114 CPU running at 2.20GHz (800MHz - 3GHz), 48GiB of system memory, and a 10 Gigabit Ethernet interconnect. Table 1

Type	Model	Interface	Capacity
NVMe	Samsung 960 Evo	NVMe	250GB
SATA SSD	Samsung 860 Evo	SATA	512GB
SATA HDD	Seagate LM049-2GH172	SATA	1TB

Table 1. Storage Devices Specifications

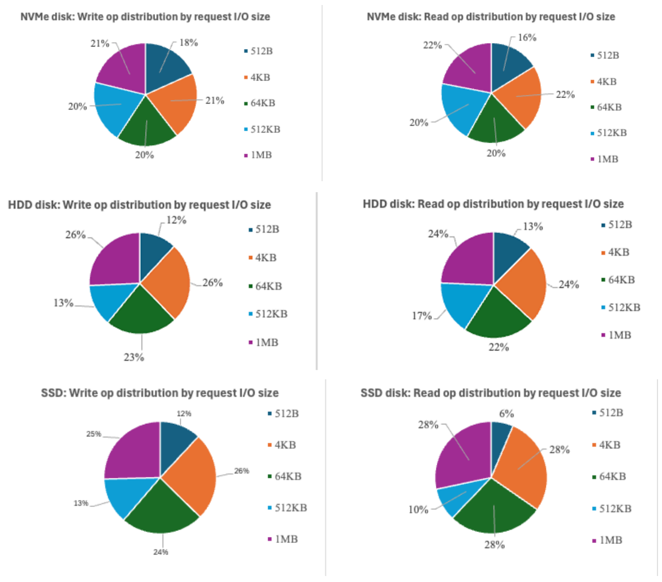


Figure 5. Evaluation workload distribution by IO size

shows the configuration of the disks that we use for the benchmarking. For each disk mentioned in Table 1, we have performed I/O benchmarking using IOR for the file sizes [512 B, 4KB, 64 KB, 512 KB, 1 MB] to fairly match the workload distribution of thesios. Fig 5 the distribution of the read and write requests across the disks. We generated reasonable workload to capture the traces sufficient for the comparison.

5.2. Latency comparison against thesios

The latency comparison for the write operations of the disks in this experiment with the Thesios disk is shown in Fig. 6. Across all the storage types, latency tends to increase with IO size. Thesios. Similar to Thesios, the latency of SSDs and NVMe disks increased with the IO size, showing the expected behavior. NVMe outperforms SSDs as expected but SSDs latency is higher than HDD which could be due to the block size combinations we used in the experiment. However, the latency difference between NVMe and SSD is less than 1.6ms. The low latency of Thesios suggests superior I/O optimization and workload handling compared to our disks.

Figure 7 shows the latency comparison for the read operations, comparing the latencies of various disks against the Thesios disk. The Thesios disk consistently shows

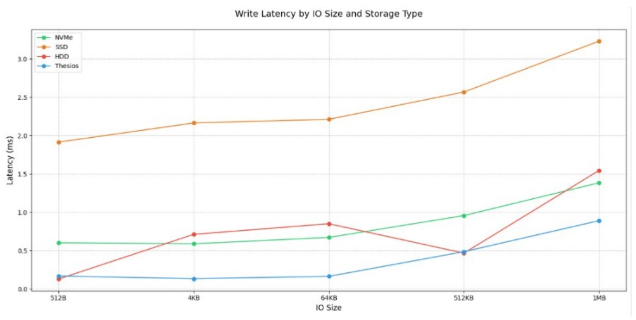


Figure 6. Write Latency comparison of test disks vs. Thesios

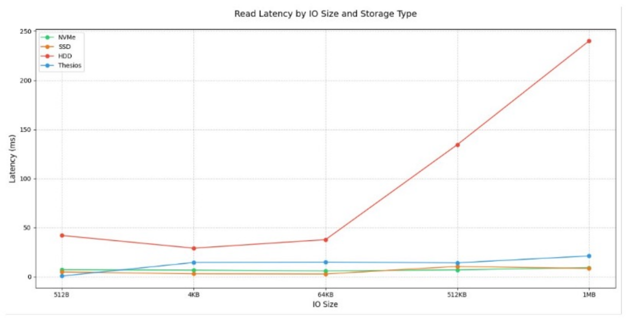


Figure 7. Read Latency comparison of test disks vs. Thesios

low latency, which suggests its high-performance hardware configuration. Latency of HDD is larger when compared to NVMe and SSD as HDDs are traditionaly slower than NVMe and SSD. The latency patterns for NVMe, SSD, and Thesios are similar, with all three storage types showing minimal and steady increases in latency as the IO size grows. However, Thesios demonstrates slightly higher latency compared to both NVMe and SSD across all IO sizes, although the difference is relatively small. In contrast, HDD latency is significantly higher, particularly at 1MB. This sharp increase could be attributed to the larger block sizes used in the experiments, which are known to heavily impact performance due to their mechanical limitations. Overall, while Thesios, NVM(e), and SSD offer comparable and efficient performance.

5.3. Bandwidth comparison against thesios

The Fig. 8. is the bandwidth comparison graph against all the disks. We observed that the bandwidth increases with the IO size for all storage types, indicating improved data transfer rates as larger blocks are processed. Thesios, NVM(e)exhibit similar bandwidth patterns, with consistent scaling as IO sizes increase. Among these, Thesios generally maintains a slightly lower bandwidth compared to NVM(e), it could be due to the high configuration or faster performance of thesios disk. The SSD bandwidth is notably high compared to Thesios, while its latency is lower. This

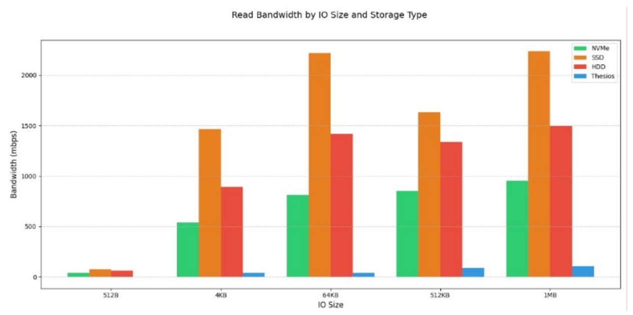


Figure 8. Bandwidth comparison of test disks vs. Thesios

could be because of the smaller dataset used in the experiments.

6. Conclusions and Future Work

In conclusion, the Thesios synthesized traces closely resemble the normal disks, as observed in our experimental analysis. These traces demonstrate consistent patterns in latency and bandwidth, and it aligns with the expected performance of various storage types. We can conclude that Thesios traces can reliably be used for further analysis and experimentation. However, the scope of this project is to analyze the transferability of the I/O traces released by Google as part of Thesios. In the future work, the project could be extended to analyze thesios methodology and its framework to check its adaptability for different storage servers as the methodology was implemented and validated against disks in Google storage server but the methodology was not validated against other storage systems such as key-value store. This would provide insights into the versatility of Thesios and its potential to simulate IO traces on diverse storage environments effectively.

7. Acknowledgements

This report is based on a thorough analysis and experimentation conducted using Google's Thesios data at Illinois Institute of Technology. The work was carried out under the guidance of Ms. Meng Tang, as part of the term project for the course CS546.

References

[1] G. R. Ganger, "Generating representative synthetic workloads: An unsolved problem," in *Proceedings of the Computer Measurement Group (CMG) Conference*, dec 1995. 1

[2] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/o performance challenges at leadership scale," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, p. 40, ACM, 2009. 1

[3] P. M. Phothilimthana, S. Kadekodi, S. Ghodrati, S. Moon, and M. Maas, "Thesios: Synthesizing accurate counterfactual i/o traces from i/o samples," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24)*, vol. 3, (New York, NY, USA), pp. 1016–1032, Association for Computing Machinery, 2024. 1, 2

[4] L. L. N. Laboratory, "Ior benchmark." https://github.com/chaos/ior, 2015. 2

[5] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular hpc i/o characterization with darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, (Salt Lake City, UT, USA), pp. 9–17, 2016. 2