Understanding Flash-Based Storage I/O Behavior of Games

Adnan Maruf*, Zhengyu Yang†, Bridget Davis†, Daniel Kim†,
Jeffrey Wong†, Matthew Durand†, and Janki Bhimani*
*Knight Foundation School of Computing and Information Sciences, Florida International University
†Memory Solution Research Lab, Samsung Semiconductor Inc., USA

2021 IEEE 14th International Conference on Cloud Computing (CLOUD) | 978-1-6654-0060-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CLOUD53861.2021.00068

***Abstract*—Computer games are an extremely popular but overlooked workload. Cloud-gaming has been one of the biggest buzzwords in the gaming industry throughout 2020. The rapid growth of the video gaming industry and the diverse set of popular video games available today raises increasing concern to properly understand its I/O characteristics to improve their performance and design better gaming servers and consoles. To the best of our knowledge, this is the first attempt to systematically measure, quantify, and characterize the organization of game data into files, back-end storage access patterns, and the performance of gaming workloads. We explore the I/O behavior of 14 recent and famous games, producing a series of observations coming from measurements done on a real setup.**

I. Introduction

The video game industry has been growing rapidly in recent years with a total of $159.3 billion in 2020 and is estimated to reach $164.6 by 2022 [1]. In 2020, there are currently around 2.5 billion active gamers and more than 400 video game developing companies worldwide [2], [3]. Cloud-gaming has been one of the biggest buzzwords in the gaming industry throughout 2020 with many giant companies into cloud gaming services such as Google's *Stadia*, Nvidia's *GEFORCE NOW*, Amazon's *Luna*, Sony's *PlayStation Now*, and Microsoft's *XCloud* [4]. For most gaming service providers, with an increasing demand globally, it has become very challenging to deliver a qualitative user experience along with high performance.

From the computational viewpoint, games are high CPU, memory, and storage demanding workloads [5]. But, in contrast to its demand, there exists very few works [6], [7], [8], [9] that understands the characteristics of games. Storage system designers have recently shown interest in customizing I/O subsystems for the gaming workloads to enhance the system performance [10]. As the underlying storage devices play an important role in the system performance, modern video gaming consoles have replaced Hard Disk Drives (HDD) with faster Solid State Drives (SSD) [11]. Therefore, as the world is shifting the bulk of its gaming workload needs onto high-performance SSDs, it is essential to analyze I/O characteristics of the gaming workloads on high-speed flash-based storage and understand the overall performance implications to fully utilize the available cloud-storage resources.

[1]This work was initiated during Adnan Maruf's internship at Samsung Semiconductor Inc. This work was partially supported by National Science Foundation Award CNS-2008324.

In this paper, we mainly ask the following three research questions:
RQ1: *How to design a systematic methodology to analyze the I/O characteristics of the PC games?*
RQ2: *What is the file organizational structure and backend storage access patterns of different games?*
RQ3: *How to measure and analyze game performance?*

PC games are not frequently used as workloads for system research, and therefore there are few resources available to guide PC games developers. To the best of our knowledge, this work is the first attempt towards a new direction of game-as-a-service (GaaS). We are the first to collect I/O traces of different popular games and analyze the game I/Os characteristics. In total, we played 14 different popular games developed by more than 40 developers and collected game data totaling up to 1 TB for playing them once. We focus on the single-player games played on a gaming PC configured to match the popular consoles' hardware configuration.

We investigate the organization of game data into files to find the temporal locality within the files. Then we analyze the backend storage access patterns of the games to find common trends in the game I/Os. We also compare the game I/O behaviors with the public block-level I/O traces released by Microsoft Research Cambridge (*MSR*) [12], and I/O traces collected from block storage system at Alibaba Cloud (*AliCloud*) [13]. Furthermore, we analyze the end-to-end performance of the cumulative stack while playing games. Finally, based on our observations, we discuss intuitive ways to design optimized storage solutions and better games to improve the overall gameplay experience. We present our most interesting findings summarized into 8 key observations.

From our analysis, we observed several significant common properties and characteristics across all the games. We observe that large proportions of I/O activity happen in the initial gameplay period, games are highly read-intensive, have high spatial locality, and perform bursty I/Os. We also observed a high gap between the average and the peak performance, large I/O queue length, and low storage bandwidth utilization. In the next sections, we first study the related works in section II, discuss the experimental methodology in section III, present our observations in section IV, and finally, in section V, we discuss storage and game design guidelines for improving storage performance on game workloads.

2159-6190/21/$31.00 ©2021 IEEE
DOI 10.1109/CLOUD53861.2021.00068

II. Related Work

As storage devices play an important role in the system performance, there have been numerous works collecting and analyzing the I/O behavior of different workloads in large-scale storage systems, high-performance computing (HPC), and mobile applications [14], [15], [16], [17], [18]. However, there is no study characterizing the I/O of video game workloads. There have been several works in the area of the game and multi-media streaming workloads and traces [19], [20]. However, these game traces do not focus on the storage I/O, and the works on the multi-media streaming workloads didn't collect or analyze any game workload. To analyze game workloads, [21] generated synthetic workloads for multiplayer online games. Although these synthetic workloads can help understand and design new systems specifically for gaming, these do not reflect the actual I/O behavior of the game workloads.

III. Methodology

In this section, we discuss our game software architecture, gameplay workloads, experimental setup, trace collection method, and the key I/O properties on which we focused.

A. Gameplay Workloads

We choose 14 most popular games spanning across 5 different genres. We collect storage I/O traces while playing these games and perform in-depth analysis to derive interesting observations. Table I shows the details of these games with the release year, genre, game installation size, and the duration of the gameplay trace. Although the gameplay time for all the games is not equal, as each of the games has a different game load time, selection process, and runtime before it ends, our collected traces capture the significant I/O behavior. This also leads us to some very interesting findings, as discussed in our observations.

B. Experimental Setup and Trace Collection

We played the selected games on a local machine running on Windows 10. Our gaming machine has AMD 3900X Gen-3 Ryzen CPU with 12 cores; clock speed is 3.8 GHz with 4.6 GHz boost. We used Nvidia RTX 2070 SUPER, 9 TELOPs 1.6 GHz GPU. The system has 16 GB DDR4 (3200 MHz) memory and 2 TB of flash-based NVMe storage using Samsung 970 EVO Plus SSDs.We tried to have a similar hardware configuration as popular gaming consoles, e.g., Microsoft Xbox Series X [22].

For trace collection, we used the Windows Performance Recorder (WPR) [23] along with Windows Performance Analyzer (WPA) [24]. We used the WPA tool to create the data tables of the Event Tracing for Windows (ETW) events captured by the WPR tool. Then only the storage I/O events were filtered for our analysis. Each of the games is played three times to ensure correctness and reproducibility.

TABLE I
List of the selected games with the release year, genre, game size, and gameplay duration.

Game ID	Game Name (Year)	Genre	Size (GB)	Duration (Min)
01	Assassins Creed: Odyssey (2018)	AA	98.5	19.17
02	Control (2019)	AA	41.8	64.21
03	Shadow of the Tomb Raider (2018)	AA	35.3	46.29
04	Wolfenstein (2019)	FPS	42.8	35.54
05	Metro Exodus (2019)	FPS	71	15.05
06	Final Fantasy XV (2016)	AA	86.1	60.02
07	Call Of Duty (2019)	FPS	186	12.71
08	Mechwarrior (2019)	FPS	43.2	66.31
09	Deliver Us The Moon (2018)	A	7.8	115.84
10	Forza Horizon 4 (2018)	CR	76.2	73.35
11	Grand Theft Auto V (2013)	AA	89	9.14
12	Battlefield V (2018)	FPS	88.7	35.71
13	Fortnite (2017)	FPS	81.3	17.44
14	Minecraft (2020)	S	0.39	13.19

C. Game Properties and Performance Metrics

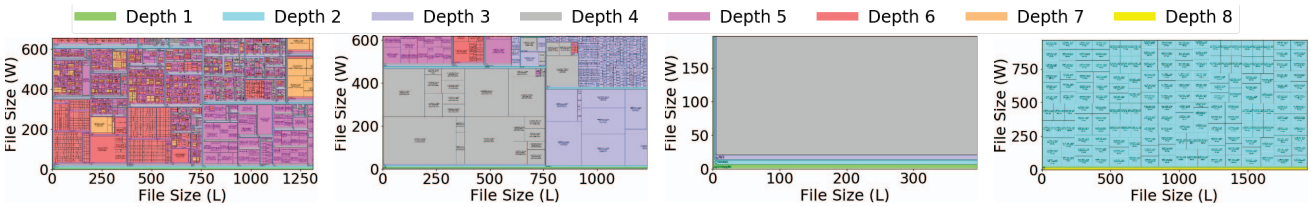
Our carefully selected game properties and performance metrics can be categorized into three types: i) *Game File Properties*, ii) *Game I/O Properties*, and iii) *Performance Metrics*. In the *Game File Properties* category, we analyze the number of files and file sizes on the storage device created by the games using [25] and the percentage of the accessed portion of the accessed files, i.e., the ratio of the total access and the total size of the accessed file. For *Game I/O Properties*, we analyze the total number of read and write I/Os requested by a game, the percentage of the read and write I/Os, the distribution of the various I/O sizes observed during the gameplay, I/O access histogram of every game. In the *Performance Metrics* category, we analyze average and peak throughput (IOPS, i.e., I/O operations per second) and bandwidth(MBPS, i.e., megabytes per second), application-level I/O queue length, end to end I/O latency, and tail latency. We used end-to-end I/O latency and latency interchangeably in this paper which is defined as *latency* = *waitingTime* + *operationTime*. Tail latencies are known as the upper percentiles of the latency distribution, e.g., the 99-th and 99.9-th percentiles.

IV. Results and Analysis

In this section, we present our results and analysis of the I/O performance while running gaming workloads. We discuss how gaming clients' data is organized and accessed within the files, the backend storage access patterns of different games, and the end-to-end performance of the cumulative stack, including the application server and backend storage.

In particular, in this section, we ask and find answers to the following interesting research questions:

- *Q1:* How do different games pack their data into files?
- *Q2:* What amount of data is touched within the accessed files?
- *Q3:* How do the games access the backend storage?



(a) Final Fantasy XV (b) Forza Horizon 4 (c) Deliver us the moon (d) Call of Duty

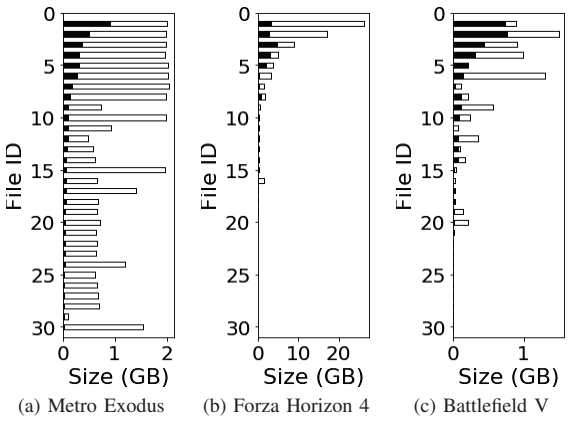
Fig. 1. **File sizes and the number of files used by games vary significantly. Each file is represented by a rectangle and file size is proportional to *L*W*, specifically, *1L*1W* denotes 100 KB file size. Different colors denote different depths of the directories.**

- *Q4:* Is there any locality in data access?
- *Q5:* Is there any common I/O pattern across games?
- *Q6:* How is the observed storage performance for the games?
- *Q7:* What are the reasons for performance limitations?

Observation 1: *Some game developers make explicit thoughtful design choices to efficiently pack data into files.*
The average file sizes of the gaming workload range from 40 KB to 685.89 MB for different games, which is larger than 189 KB, the average sizes of the general purpose files reported in [26]. In Fig. 1 we show the number of files and file sizes of the four games that represent the file properties of most other games. In the figure, each rectangle represents a file, and the file size is presented by its area (i.e., Length(L)*Width(W)). Each *1L*1W* area represents 100 KB file size. The larger the area, the larger the file. Different colors denote the depth of the directory. As we can see from Fig. 1, the number of files each game generate differs a lot. *Final Fantasy XV* uses very high number of files (i.e., 65,000). Interestingly, none of the games accesses all the files it initially generated during a gameplay. Most of the games access less than 60% of the total number of files generated. We mainly identify four different types of file packing. First, some games uses large number of small files (e.g., see Fig. 1a), second, games such as *Forza Horizon 4* (see Fig. 1b) has a mix of different sized files, third, other games such as *Deliver Us The Moon* (see Fig. 1c) combines most of its data into a single very large file, and finally, some games maintain many equally sized files (e.g., see Fig. 1d).

Packing most of the gameplay data into a single large file will allow to *mmap* that large file avoiding the multiple read *syscalls*, and can also use the page cache for caching the hot parts of that large file. However, if the main memory is not sufficient to entirely cache these large files, then a lot of page-in and page-out overhead may incur, deteriorating the overall performance of such games.

Observation 2: *Only a small amount of data is touched within accessed files.*
Although games do not use all the files initially created during gameplay, more importantly, games also do not use all the data within the accessed files. We observe that only a small portion of a file is being accessed by the games. For example, on average, *Metro Exodus* only uses 11.41% of data from the files it accesses. Fig. 2 shows the ratio of the amount of data used (solid portion of each bar) within the thirty most accessed files in three randomly picked games *Metro Exodus*, *Forza Horizon 4*, and *Battlefield V*. We observe a similar trend across all



(a) Metro Exodus (b) Forza Horizon 4 (c) Battlefield V

Fig. 2. **A very small portion of file data is touched within the accessed files. File accessed part (solid black) vs. entire file ratio for the top 30 most accessed files are shown. The file accessed part combines all unique I/Os of a file.**

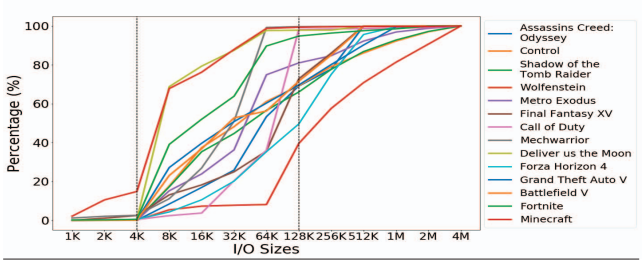
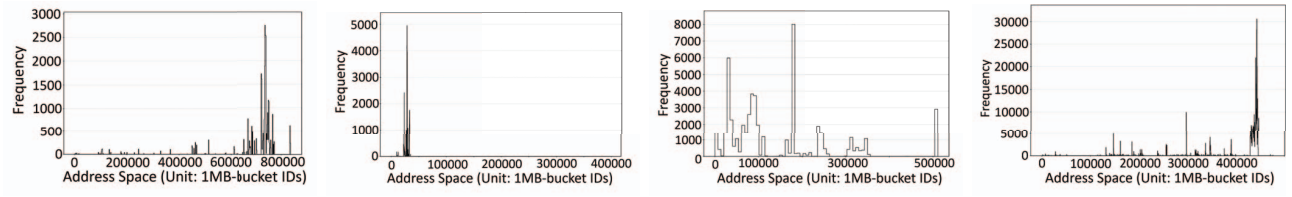


Fig. 3. **Most of the game I/Os are ranged between 4 KB - 128 KB. The number of I/Os smaller than 4 KB and greater than 512 KB is very low. I/O size distribution of all the games is shown.**

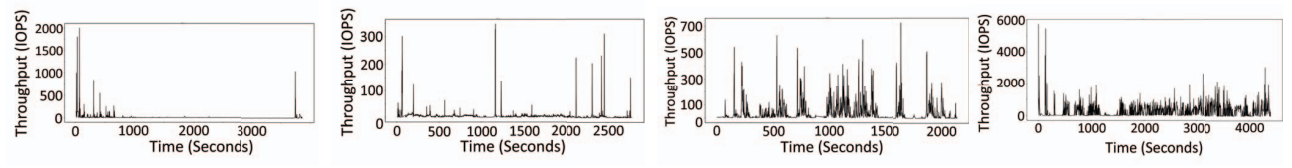
the other games. While running these games, we do not need to load/prefetch/cache the entire file into low latency access media (persistent memory, high-end SSD, or DRAM). The prefetching techniques, caching algorithms, and storage tiering data rearrangement mechanisms such as [27], [28] that works on the general design intuition, that most of the data within any accessed files is used, may not provide good performance for gaming platforms and new techniques that leverages data organization properties of games may be required to achieve optimal performance.

Observation 3: *Game workloads are highly read-intensive.*
Most general purpose applications show a combination of reads and writes such as *AliCloud* workloads [13] show 25%



(a) Control (b) Wolfenstein (c) Call of Duty (d) Forza Horizon 4

Fig. 4. **I/O access histogram of four games representing the pattern of all other games. High spatial locality is observed across the games. Games access specific data regions very frequently.**



(a) Control (b) Shadow of the Tomb Raider (c) Wolfenstein (d) Forza Horizon 4

Fig. 5. **Runtime throughput of the games shows a bursty I/O pattern for a short period of time. Initial game load phases have the most I/O activities. These four games can represent all other games' runtime throughput performance.**

reads and 75% writes. However, interestingly we observe that most of the games are very read-intensive, except a few such as *Deliver us the moon*. Overall, on average, 99.29% of all the game I/Os are read I/Os. This means most of the games are mainly reading resource files during the gameplay and writing almost nothing to the disk. The games need to save the states of the current gameplay, and this requires writes which, for many of the games, are way smaller than the resource files these games read. These state information can be easily stored in the system memory during the gameplay, and according to each game's internal logic, they are either triggered periodically or manually written back to disk. Thus, persistent storage media such as NAND flash, which provides low latency access, but has the huge drawback of limited write cycles, are a perfect fit to be used within gaming storage servers and consoles.

Observation 4: *Most of the games use small I/O sizes ranging between* 4 *KB* - 128 *KB.*

To understand the I/O size distribution of the games, in Fig. 3, we plot the cumulative distribution function (CDF) of the I/O sizes for different games. We observe that most of the games have small sized I/Os. In *MSR* [12] I/Os smaller than 64 KB and in *AliCloud* [13] I/Os smaller than 32 KB are majorly observed. In games, the vast majority I/Os range between 4 KB - 128 KB. It is possible that game developers intentionally pick this I/O size range for better alignment to the memory page size of 4KB and huge page size. This indicates that to improve the I/O performance of games; we should pay attention to how to improve the latency of 4 KB - 128 KB from the storage since they are the vast majority of I/O sizes.

Observation 5: *Games show high spatial locality with the data stored in some specific storage regions being very frequently accessed.*

We investigate the spatial locality of I/O accesses and plot Fig. 4, with the x-axis as the disk address space (we use 1 MB bucket as the granularity, and thus x-axis is in 1 MB-bucket IDs), and the y-axis as the frequency of accesses to each bucket. Thus, Fig. 4 shows how many times each address (bucket) is being accessed (read + write). We observe that most of the games have a high concentration of I/O access in some address space. For example, in the Fig. 4d, the game *Forza Horizon 4* has very high access frequency in the address space ranged from $400,000$ to $500,000$ ($bucketID$). The frequency is 6x times higher than other locations for this game. A similar access pattern is observed in most other games. The observed high spatial locality for games is not common in the *MSR* [12] and *AliCloud* workloads where random I/Os are more common [13]. Integrating the program level knowledge of games with the storage device internal data organization such as Flash Transition Layer (FTL) algorithms for SSDs [29], and stream assignment techniques for multi-stream SSDs [30], will help in vastly increasing the storage device endurance and lifetime.

Observation 6: *Games perform bursty I/Os over time, with many of them depicting high I/O activities during the initial load phase.*

We further investigate the runtime throughput of the I/Os performed during the gameplay in Fig. 5. We observe that most of the games have some burst periods of high I/O activity during a short span of time. *MSR* [12] and *AliCloud* show low burstiness overall, but high short-term burstiness throughout the trace period [13]. However, many games have the bursty I/O phase in the beginning, such as the game *Control* is showing a burst of I/Os during the first 750 seconds. We anticipate this is due to the initial load phase of games when it reads the data from persistent storage into memory. However, many other games also show a burst of I/O activities at an intermediate time during the gameplay. For instance, *Shadow of the Tomb Raider*. Game instances with bursty I/Os might

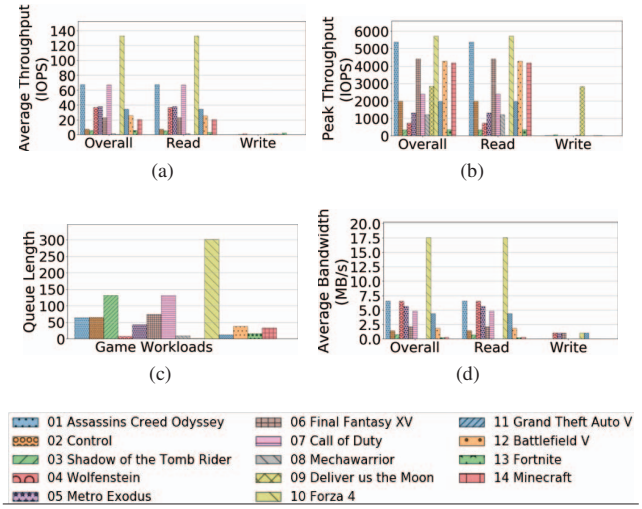


Fig. 6. **Average and peak throughput, maximum queue length, and bandwidth comparison. A high gap between the average and the peak is observed. Overhead in the software stack is reflected by the high application-level queue lengths.**

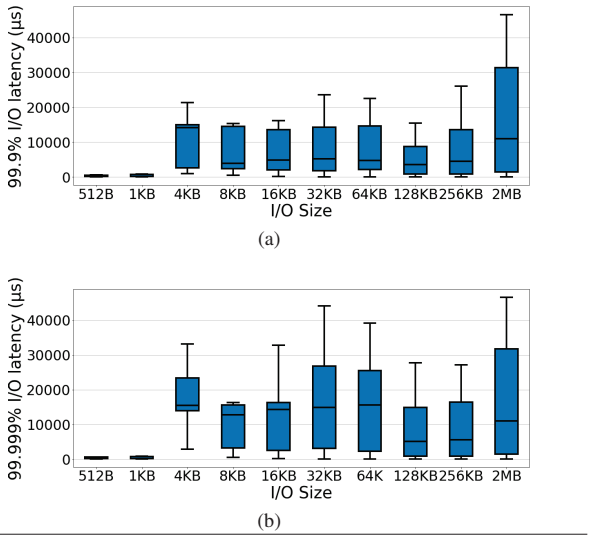


Fig. 7. **99.9% and 99.999% latency for different I/O sizes of all the games. I/O sizes used by the games play an important role in overall performance. 4 KB and larger I/Os suffer from high tail latencies.**

occupy a large amount of the memory resources during their bursty periods by evicting other cached data. Thus, if the data that is brought into the memory is not reused, then the hit rate might drop. We believe the caching techniques with the concept of short-term and long-term zones such as GREM [31] can provide sustained high hit rates for games.

Observation 7: *The high gap between the average and the peak performance, large I/O queue length, and low storage bandwidth utilization indicates that the software stack overhead dominated the overall performance.*

We compare the average and peak throughput (IOPS) in Figs. 6a and 6b. We observe a big difference between the average and the peak throughputs for all the games. For example, *Forza Horizon 4*, on average, performs 130 IOPS; however, at its best, it has the potential to perform $5,600$ IOPS. Thus, the difference between the average throughput and peak throughput across all games is very high, around 200x for some games such as *Final Fantasy XV*. To investigate it further, we study the application queue length on the host side during the gameplay. The higher the queue length is, the higher number of I/Os are waiting to be processed. Fig. 6c shows the maximum queue lengths observed for different games. The maximum queue length for most of the games reaches around 50. Finally, we see that although the peak bandwidth of the SSDs we use is $1,800$ MBPS, most of the games can only saturate up to 20 MBPS. Fig. 6d shows the average bandwidth of SSDs consumed by various games during the gameplay. This leads us to an interesting observation that although the storage device is not the bottleneck, high application-level queue lengths indicate that the software stack overhead of intermediate operating system layers is having a dominant impact on the overall performance during gameplay.

Observation 8: *I/O sizes that the games use highly impact the overall performance and latencies.*

Motivated by the above significant performance gap in terms of overall throughput, we next investigate the tail latencies for different I/O sizes that games use. We show the 99.9% and 99.999% latencies for different I/O sizes from various games in the Fig. 7. In alignment with our expectation from the characteristics of the database workloads with various I/O sizes, the tail latencies for larger I/Os (e.g., 2MB) are observed to be higher than that of smaller I/Os (e.g., 512B).

V. Discussion

In this section, we discuss how storage solutions can be optimized for game workloads. Our discussion is based on the observations we made from our experiments and result analysis. Although our experiment testbed is a local machine, we believe the observations are very insightful to design efficient storage solutions for any gaming service. Here, we first discuss possible design guidelines for storage designers and then for game designers.

Thirteen out of the fourteen real games we experimented with are highly read-intensive. Thus, the storage stack for game workloads should be optimized for read workloads. Devices such as NAND flash are good to use for gaming workloads as they provide low latency read accesses and a long lifetime. Moreover, "zoning" techniques can also be used where "zone" can be either physical (e.g., grouping multiple SSDs into a zone) or virtual (e.g., zoning inside an SSD) to separate the read and write I/Os. As we have observed, most of the games use the I/O size of 4 KB to 128 KB; configuring the cache line size and access granularity of the storage devices will significantly improve the performance. This is also useful to keep in mind while choosing or configuring software techniques such as the "Prefetch mechanism", "Indexing algorithm", "Logging techniques", and "fine-grained concurrency control" [32]. As we have seen, most of the games use only a small amount of data from each accessed file. Thus, gaming applications exhibit a very good opportunity to be accelerated by sophisticated caching and tiering mechanisms

such as hierarchical system architectures [33]. We see that games perform bursty I/Os over time. Bursty workloads can temporarily evict the important data from the cache and increase the queue length due to the sudden increase of I/Os. Burst detection and throttling techniques can be used to handle the sudden spikes in the workload.

Developers can make the best use of the storage drives as the game developers have prior knowledge of the game workload behavior, e.g., correlation of the data, data required in the upcoming session based on the current gameplay, and possible identification of the hot/cold data. Additionally, the game load phase can be optimized in the following possible ways as we have observed that most of the I/O activity occurs during the load phase of many games because these games prefetch all the necessary data from the storage to the memory. i) invalidate the cache lines before loading the game data, ii) only cache the expected future re-access data to make the proper utilization of the memory, iii) instead of loading all data in the beginning, divide the load phase into multiple sessions and fetch data from the storage device in the background.

VI. Conclusion

This paper begins by posing a simple question for investigation: *"What is the storage I/O behavior of PC games?"* We conclude by observing that storage stack, including the intermediate memory and software techniques, can have a severe impact on the performance of the games. To the best of our knowledge, this is the first attempt to systematically measure, quantify, and characterize the organization of game data into files, back-end storage access patterns, and the performance of gaming workloads. We discovered many unexplored and surprising observations. Finally, we leverage the learnings from our observations into high-level design guidelines on *"How to improve game performance by designing optimized storage solutions and better games?"*

References

[1] (2020) Video game industry statistics in 2020. [Online]. Available: https://www.wepc.com/news/video-game-statistics/

[2] H. Narula. (2019) A billion new players are set to transform the gaming industry. [Online]. Available: https://www.wired.co.uk/article/worldwide-gamers-billion-players

[3] List of video game developers. [Online]. Available: https://en.wikipedia.org/wiki/List_of_video_game_developers

[4] K. Browning, "'crucial time' for cloud gaming, which wants to change how you play," *The New York Times*, 2021. [Online]. Available: https://www.nytimes.com/2021/07/01/technology/cloud-gaming-latest-wave.html

[5] Intel. Gaming System Requirements for 2020's Hottest Titles - Intel. [Online]. Available: https://www.intel.com/content/www/us/en/gaming/resources/system-requirements.html

[6] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: an open cloud gaming system," in *MMSys*, 2013, pp. 36–47.

[7] H. E. Dinaki, S. Shirmohammadi, and M. R. Hashemi, "Boosted Metaheuristic Algorithms for QoE-Aware Server Selection in Multiplayer Cloud Gaming," *IEEE Access*, vol. 8, pp. 60 468–60 483, 2020.

[8] J. Roca, V. Moya, C. González, C. Solís, A. Fernández, and R. Espasa, "Workload characterization of 3d games," in *IISWC*. IEEE, 2006, pp. 17–26.

[9] F. Zyulkyarov, V. Gajinov, O. S. Unsal, A. Cristal, E. Ayguadé, T. Harris, and M. Valero, "Atomic quake: using transactional memory in an interactive multiplayer game server," in *ACM SIGPLAN PPoPP*, 2009, pp. 25–34.

[10] R. Priday. PS5 SSD is so fast, developers need to redesign their games. [Online]. Available: https://www.tomsguide.com/news/ps5-ssd-is-so-fast-developers-need-to-redesign-their-games

[11] R. Ramsey. PS5 SSD: Why It's Better Than HDD. [Online]. Available: https://www.pushsquare.com/guides/ps5-ssd-why-its-better-than-hdd

[12] S. N. I. Association *et al.* (2010) MSR Cambridge Traces. [Online]. Available: http://iotta.snia.org/traces/388

[13] J. Li, Q. Wang, P. P. C. Lee, and C. Shi, "An in-depth analysis of cloud block storage workloads in large-scale production," in *(IISWC)*, 2020, pp. 37–47.

[14] T. Patel, S. Byna, G. K. Lockwood, and D. Tiwari, "Revisiting I/O Behavior in Large-Scale Storage Systems: The Expected and the Unexpected," in *SC*. ACM, 2019.

[15] J. Bhimani, A. Maruf, N. Mi, R. Pandurangan, and V. Balakrishnan, "Auto-tuning parameters for emerging multi-stream flash-based storage drives through new i/o pattern generations," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[16] J. Lüttgau, S. Snyder, P. Carns, J. M. Wozniak, J. Kunkel, and T. Ludwig, "Toward Understanding I/O Behavior in HPC Workflows," in *PDSW-DISCS*, 2018, pp. 64–75.

[17] P. Liu, A. Maruf, F. B. Yusuf, L. Jahan, H. Xu, B. Guan, L. Hu, and S. S. Iyengar, "Towards adaptive replication for hot/cold blocks in hdfs using memcached," in *ICDIS*, 2019, pp. 188–194.

[18] J. Courville and F. Chen, "Understanding storage I/O behaviors of mobile applications," in *MSST*, 2016, pp. 1–11.

[19] Y. Guo and A. Iosup, "The Game Trace Archive," in *NetGames*, 2012, pp. 1–6.

[20] K. Cho, Y. Ryu, Y. Won, and K. Koh, "Virtual interval caching scheme for interactive multimedia streaming workload," in *ISCIS*. Springer, 2003, pp. 276–283.

[21] M. Lehn, T. Triebel, and R. Rehner, "On synthetic workloads for multiplayer online games: a methodology for generating representative shooter game workloads," in *Multimedia Systems*, vol. 20, 2014, pp. 609–620.

[22] (2020) Xbox series x. [Online]. Available: https://www.xbox.com/en-US/consoles/xbox-series-x#specs

[23] Windows performance recorder. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/test/wpt/windows-performance-recorder

[24] Windows performance analyzer. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/test/wpt/windows-performance-analyzer

[25] "SpaceMonger." [Online]. Available: https://github.com/seanofw/spacemonger1

[26] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," in *FAST 07*. USENIX, 2007.

[27] C. S. O'Connell, J. J. Berenson, and N. I. Rajkumar, "Method and system for file-system based caching," 2011, patent No. US 7024452 B2, Filed Jul. 15, 2002, Issued Apr. 4, 2006. [Online]. Available: https://patents.google.com/patent/US7024452B1/en

[28] B. A. Ignacio, C. Wu, and J. Li, "Warmcache: A comprehensive distributed storage system combining replication, erasure codes and buffer cache," in *GPC*, S. Li, Ed. Springer, 2019, pp. 269–283.

[29] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," *Acm Sigplan Notices*, vol. 44, no. 3, pp. 229–240, 2009.

[30] C. Choi *et al.*, "Multi-stream write SSD," *Flash Memory Summit*, 2016.

[31] Z. Yang, J. Tai, J. Bhimani, J. Wang, N. Mi, and B. Sheng, "GReM: Dynamic SSD resource allocation in virtualized storage systems with heterogeneous IO workloads," in *IPCCC*, 2016, pp. 1–8.

[32] J. Bhimani, J. Yang, N. Mi, C. Choi, M. Saha, and A. Maruf, "Fine-grained control of concurrency within kv-ssds," in *SYSTOR*. ACM, 2021.

[33] S. Zheng, M. Hoseinzadeh, and S. Swanson, "Ziggurat: A tiered file system for non-volatile main memories and disks," in *FAST 19*. USENIX, 2019, pp. 207–219.