WoW-IO: a Gaming-Based Storage Trace Generator for Edge Computing

Oleg Kolosov, Tom Herman, Ido Zohar, Gala Yadgar
Computer Science Department, Technion
{*kolosov,gala*}*@cs.technion.ac.il,* {*tomhe,idozo*}*@alumni.technion.ac.il*

***Abstract*—Workload traces play a crucial role in the design and evaluation of storage systems. They are used for evaluating system performance and optimizing aspects such as data access speeds, energy consumption, and load balancing. Most publicly available traces were collected in cloud environments, which limits their ability to represent workloads in user-centric and highly disaggregated settings such as edge systems.**

In this paper, we present WoW-IO, an open-source object trace generator based on the popular video game 'World of Warcraft'. WoW-IO uses, as input, logs of in-game avatar information from the game's servers. We describe the principles and assumptions used in the design of WoW-IO, how the generated traces can be used to evaluate various aspects of an edge storage-system design, and how the trace format can be extended to reflect additional details and complex usage scenarios.

I. Introduction

In the process of designing and optimizing a system, the characteristics of the expected workload play a crucial role. Workload traces from similar systems—and from applications that are expected to run on the system—provide valuable insights into how it will perform and how it can be improved. We focus on *storage traces*, which represent the user I/O requests from a storage system: get and put requests from object storage systems, or read and write commands in the case of a block interface. Such I/O traces are commonly used to design, optimize, and evaluate file-system organization [1], [2], caches [3], [4], [5], energy consumption [6], [7], [8], and device firmware [9], [10].

We are interested in an emerging storage-system model in the context of *edge computing*—the collection of resources placed at inter-operating *edge nodes* such as cellular towers and hotspots. This paradigm is increasingly adopted as a distributed storage backend for various IoT applications [11], [12] and as a caching layer for cloud services [13], [14], [15]. Representative storage traces are essential for designing effective edge storage systems.

Unfortunately, currently available I/O traces have been collected on corporate application servers (e.g., Microsoft [16]), cloud object storage (e.g. IBM [17] and Alibaba [18]), in-memory cache clusters (e.g., Twitter [19]), key-value backing stores (e.g. [20]), and mobile phones [21]. These traces capture the aggregate I/O workload as it arrived at the device, often including only the block address or object ID, I/O type (read/write/put/get), and timestamp. Some traces include anonymized process and file context [22], [20], but these are insufficient for deriving *user-level information* such as the number of currently active users, their physical locations, their I/O requirements, and the interactions between them. This information is important for designing and optimizing disaggregated systems and services, and especially edge-based systems. Nevertheless, in the absence of edge-based storage workloads, most current traces used for evaluating edge systems are either synthetic [23], [24] or were collected in datacenter settings [25], [26].

In this paper, we address this glaring gap between available traces and the workloads of representative edge users and applications. We focus on multiplayer online gaming—a prominent use case for edge computing. Gaming is a multi-billion dollar industry, with a billion active users worldwide. The rise in the popularity of video games is attributed to the increasing affordability of gaming consoles and mobile devices and the accessibility of the internet. Currently, online games are hosted on remote central servers, sometimes located thousands of miles from the end users. This potentially impairs latency and the overall Quality of Experience (QoE). While regional servers have been deployed to enhance QoE [27], their coverage area (e.g., North America) still displays latencies above 50ms, which poses a significant burden on QoE [28]. The edge is envisioned to mitigate these issues by acting as a local gaming server [29]. For example, a local edge system could host a gaming service, dedicated exclusively to players within the same city. This localized approach ensures that most in-game interactions are processed on the edge, thereby reducing the need to route data through distant cloud servers.

Unlike autonomous vehicles, which are the "classic" example of an edge use-case, gaming is ready for near-term implementation and significant revenue-generation potential. This makes it an ideal application for reflecting edge storage workloads. However, currently available gaming-based traces do not contain storage-related information: they focus on player behavior [30], [31], [32] or on other system components such as network and CPU [33], [34].

Our main contribution in this paper is **WoW-IO**—a new object-trace generator based on the popular massively multiplayer online video game *'World of Warcraft' (WoW)*. The generator receives, as input, a trace collected during a previous study on in-game behavior [30]. We leverage the user-level information in the input for generating I/O requests that reflect the user's activities within the game. In the following, we describe the design of the generator and how we used the resulting traces to study the behavior of edge storage systems

©2024 IEEE. Personal use of this material is permitted.
Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

(a) Original map (Kalimdor in red) (b) Colored zones (c) Rectengular (d) Avatar density

Fig. 1: Transformation of Kalimdor into a rectangular-shaped continent with rectangular zones.

in our recently published work [35]. In addition, we show how the resulting traces can be used directly as input for evaluating caching, locking, queuing, routing, and other storage-related mechanisms. We also show how the generator can be extended to augment the traces with representative distributions of user locality, differentiated services, etc. WoW-IO is available as an open-source project at github.com/olekol33/WoW-IO.

II. The Datasets

The game. World of Warcraft (WoW) is a popular online multiplayer game that has been enjoyed by millions of players worldwide since its launch in 2004. In the game, users create *avatars* (characters) to explore the virtual universe, complete quests, and battle other players or in-game creatures. The universe includes three continents, each divided into zones. Zones are several (virtual) square kilometers in size and contain small and large cities, as well as other instances such as forests and mountains. Avatars move within and between zones by walking, using various transportation means (such as boats), or via portals. These game structures are common to current MMORPGs [36], including the current version of WoW, which has more than 7 million active users worldwide [37].

Avatar-history dataset. Of several WoW-derived datasets [33], [38], the *World of Warcraft (WoW) Avatar-History dataset (WoW-Avatar)* [30] is the largest one available. It was collected by sampling a game server in Taiwan every 10 minutes over a period of 3 years, from 2006 to 2009. The dataset consists of 138,084 *samples* with gameplay data on 91,605 avatars. Each sample contains information about all currently active avatars, their gameplay timestamps, locations (continent and zone), and game attributes such as level and profession. Some avatars belong to *guilds*, indicating collaboration between their users. Figure 1(a) shows the game map at the time the trace was collected.

A. From avatar history to storage requests

The premise of our trace generator is that the data accessed by the user's game client is directly connected to the avatar's in-game behavior. Specifically, segments of the map surrounding the avatar's current location, as well as information about other avatars in its vicinity, are required for creating the game's display. As the avatar moves around the universe, the relevant map segments are retrieved to reflect the avatar's new environment. Thus, the I/O generated by a user depends on their avatar's location and mobility, as well as on the avatars of other currently active users. Nevertheless, there are some significant gaps between the information provided in the avatar histories and the information required for a meaningful I/O trace. Below, we describe these gaps and the assumptions we rely on to bridge them.

Objects. I/O traces represent accesses to data blocks or objects, which are not included in WoW-Avatar. We assume that each avatar, guild, and map location is represented by an individual object. The objects hold all the information required by the game client, e.g., special artifacts within a location, moves available to an avatar, skills of guild members, etc.

Requests. We assume each avatar belongs to a single user, which periodically generates `PUT` and `GET` requests for the object representing its avatar and `GET` requests for the objects representing its guild, location, and other avatars in this location. In addition, writes are made by the 'system' to update location and guild objects. We assume a constant request rate for these objects, although this assumption can be easily changed.

Location granularity. In WoW-Avatar, locations are given in the granularity of zones, where the three continents are divided into 229 zones of different sizes and shapes. A realistic gaming workload should include I/O requests generated from the interaction between avatars. The given zone granularity is too coarse to distinguish between locations in which an avatar is alone or near other avatars. Thus, we artificially divide the map into *cells* of size 60×60 meters (186K cells in total). For simplicity of implementation, we also transform the zones into simple rectangular areas whose size corresponds to the size of the original zone. For example, Figures 1(b) and 1(c) show the zones of the Kalimdor continent and the respective rectangular zones used by our generator.

We further assume that cities and instances (e.g., temples) are distributed randomly within their zones and that avatars are more likely to be in these points of interest than in

outer locations. This assumption is based on the structure of the game, where such locations provide more opportunities for quests and interaction between avatars. As a result of the original locations and our assumptions, the distribution of avatar locations in the I/O trace resembles a natural exponential popularity distribution. For example, Figure 1(d) depicts the average number of avatars in each cell in Kalimdor in a specific sample. It demonstrates that most avatars are located in the capital, with the remainder distributed across the continent.

Avatar mobility. The granularity of the avatar histories in terms of sample frequency (every 10 minutes) and location (zone only) is insufficient for representing detailed mobility within the universe. Thus, we interpolate per-second avatar locations based on their zone locations. For each sample, we choose a random location within the avatar's zone according to the assumptions explained above. We assume that avatars move at a constant speed between locations within the same zone, via a Manhattan path. If an avatar moves between zones in consecutive samples, we assume it uses a portal, and that it spends half of the time (between samples) in each zone. Note the avatars are always located in the zones specified in WoW-Avatar, and that the granularity is enhanced to add a more realistic aspect to the traces.

Generality. The objects in focus are not unique to WoW. In gaming, avatar locations, state, and inventory, are known as *state data*. Modifications of such data are executed by an in-memory database in real-time and backed on the disk [39]. In multiplayer online games, state data is crucial in ensuring proper gameplay, as it is exchanged in real-time between active users and involves many gameplay aspects. The traces generated by WoW-IO thus represent the fundamental functionality common also in other MMORPGs where users control avatars within a virtual world. We discuss possible extensions in terms of scale and game design in Section III.

B. I/O trace generation

Weekly server maintenance and occasional sampling issues caused 13.3% of samples to be missing over the 3-year collection period [30]. To ensure continuity within the trace, we divide the samples into continuous time periods we term *scenes*. Each scene is further divided into 10-minute *traces*. If the time delta between two consecutive samples is more than 25 minutes, then a new scene begins. Scenes shorter than a day are discarded. The dataset contains 158 scenes in total, each starting at virtual time 0.

WoW-IO creates a representation of the universe based on 'World of Warcraft: Classic' [40]—a recreated version of the original game. We extracted zone sizes, cities, city types (minor, major, or capital), location instances (such as temples), and remaining areas (such as forests) from the WoW Classic map. The universe-creation step distributes the cities across each zone, such that city sizes vary from one to three cells.

The object trace is generated for each scene independently. Within a scene, the avatar's zone is determined from its sample, and its location within the zone is determined based on a configurable distribution. In our default settings, If the avatar's zone is the same as its zone in the previous sample, then it remains in the same location with a probability of 0.5. Otherwise, it moves to other locations, such as major or minor cities, with a probability linear to the location size.

```
User_ID, Timestamp, Object_ID, Type
A_1859, 0.0, AO_1859, PUT
A_1859, 0.0, GO_2, GET
A_1859, 0.0, LO_k_89_268, GET
A_1859, 0.0, AO_1721, GET
...
A_1859, 180.0, AO_1859, PUT
A_1859, 180.0, GO_2, GET
A_1859, 180.0, LO_k_89_269, GET
A_1859, 180.0, AO_1721, GET
...
A_1859, 390.0, AO_1859, PUT
A_1859, 390.0, GO_2, GET
A_1859, 390.0, LO_k_90_269, GET
A_1859, 390.0, AO_1721, GET
A_1859, 390.0, AO_225, GET
```

Fig. 2: An object trace sample of a user with avatar ID 1859. The full trace includes similar requests for all active users.

We assume a key-value store data structure whose entries are accessed by user requests. Requests are `GET` and `PUT` operations addressing avatar, location, and guild objects. In each virtual second, each user performs a `GET` operation on its guild and location objects, as well as on the objects of avatars in its location and guild. Additionally, each user performs a `PUT` of its avatar object. The system also updates location and guild objects via `PUT` operations: location objects when an avatar moves to a new location, and guild objects when avatars join or leave a guild. Guild updates are more frequent for larger guilds, as they depend on the ratio of guild members to total avatars.

Trace entries are formatted as `<User_ID, Timestamp, Object_ID, Request Type>`. For example, `<A_1859, 390.0, LO_k_90_269, GET>` indicates user A_1859 reading its location object at time 390, when it is located at (90,269) in Kalimdor. Figure 2 presents a partial trace for this user. At each time slot, the user writes its avatar object and reads the objects of location, guild, and other avatars. Initially, the user is located at position (90, 269), then it moves at times 180 and 390. It reads the object of avatar A_1721 who is in the same guild, and the object of avatar A_225 who is at the same location (90, 269) at time 390. At each arrival of an avatar to a new location, the location object is updated by the system. For example, `<SYS, 390.0, LO_k_90_269, PUT>` will be generated for location object (90, 269) at time 390.

In summary, due to the inherent gaps between the avatar history dataset and the information required for I/O trace generation, WoW-IO naturally includes several configurable parameters. These include the time delta between scenes (25 minutes), the I/O request rate (1 per second for each object type by each avatar), read-only trace, the probability to be in each location type, and the sizes of the cells (60×60 meters) and cities (1-3 square cells). Some of these parameters

are passed as command-line arguments, while others can be easily modified in a configuration file. Different parameters will reflect different distributions and different levels of detail, as we explain in the next section. The trace does not indicate request sizes, however, such information can be added with a simple code modification.

C. Trace characteristics

Figure 3 shows the main characteristics of the resulting object traces. Scene durations, depicted in Figure 3(a) range between 1 and 34 days, with a median of 3.8. Figures 3(b-c) show the CDF of `GET` and `PUT` operations per second in the first 60 minutes of each scene. The `GET` request rate ranges between 100 and 30K requests per second. The `PUT` request rate ranges between 40 and 670 requests per second. Although the requests are synthetically generated, note that their rate reflects the number of avatars and their behavior; more requests are generated if avatars are in the vicinity of one another, or if they are in the same guild. To account for scenarios with higher read intensity, such as those involving more active users, WoW-IO includes a configurable parameter to control the number of `GET` operations per object per second.

Figure 3(d) shows the CDF of the unique objects requested in the first 60 minutes of each scene. The number of objects ranges between 2340 and 53.4K. In all scenes, more than 80% of requests are to 1% of unique objects. In addition, our analysis showed less than 1% of objects are requested only once (omitted for brevity). Object popularity is determined by avatar presence in popular locations, such as capitals and major cities. More specifically, in a location hosting n avatars, there will be n^2 requests to their objects. Whenever the location is highly popular, the majority of requests are made to the same small group of objects.

Figure 3(e) shows a CDF of the number of unique avatars in the first 60 minutes of each scene. This number ranges between 50 and 900, which was representative at the time the trace was collected, but might be too small for evaluating current systems. The number of users can be scaled synthetically. One possible approach is to super-position several independent scenes, creating a denser object trace. Another alternative is to clone the samples; the avatar locations would still be different thanks to the randomized locations within the zone. We leave these extensions to future work.

Finally, Figure 3(f) shows a CDF of the `GET`/`PUT` ratio in the first 60 minutes of each scene, ranging from 3 to 62. Consistent with the read-heavy nature of real game workloads [41], WoW-IO traces exhibit similar characteristics. This figure shows another benefit of WoW-IO. Similarly to common cloud trace generators [42], WoW-IO offers a range of read intensities. Such variability enables the analysis of system performance under diverse load conditions, thereby offering a more comprehensive performance evaluation.

Figure 4 shows the object popularity distribution in log scale for the first 60 minutes of scene 1, whose characteristics are detailed in table I. The y-axis indicates the number of object requests and the x-axis shows the objects sorted by the number of requests, i.e., popularity rank. The `PUT` distribution starts with a flat line, reflecting the writes each user makes to update its avatar in each time slot. The `GET` distribution shows only a few objects with exceptionally high popularity, with up to 80K requests, and a majority of objects with very low popularity, often with no more than a few requests. This is a pattern of a heavy-tailed distribution, where a small fraction of the objects are highly popular. This is a common feature in many real-world object popularity scenarios [43]. A similar distribution pattern is observed in all WoW-IO scenes.

Scene	Avatars	Objects	Gets per sec	Puts per sec	% Requests to top 1% objects
1	233	21,119	1,827	182	90.3%
2	604	28,082	20,450	466	81.3%
3	250	13,153	4,575	184	83.1%
4	310	32,748	2,778	234	93%
5	903	53,157	21,079	595	89.6%

TABLE I: Evaluated scenes.

III. Research Case Studies

In this section, we explore research case studies that leverage the WoW-IO framework for evaluation.

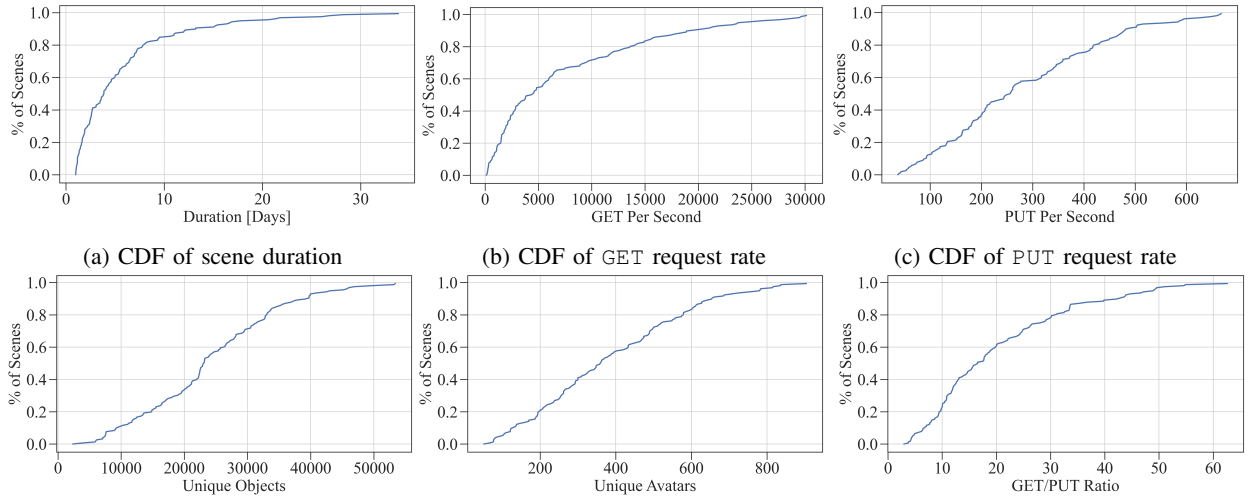
A. Edge system model

WoW-IO played a crucial role in generating workloads for our research on edge storage systems [35]. In [35], we studied the gaps between a theoretical model for edge storage and a simulated edge system. This system, designed to store data in various redundancy schemes (namely, replication and erasure coding), is focused on serving user `GET` requests. Requests are served in collaboration between edge nodes to ensure high availability and low latency.

We used WoW-IO to create a workload with 1000 users at 100 nodes. We followed the guidelines of the workload composition methodology described in [44] to create a storage workload with end-users, the objects they request, the timing of these requests, as well as the edge nodes and the specific objects they store. We used a combination of WoW-IO requests with synthetic data (the locations of users and object replicas) to create the complete workload for our experiments.

We selected two of the longest contiguous scenes, each reflecting a demand distribution spanning slightly less than a month. Detailed information on these periods is given in Table II. In each scene, we extracted the requests of the 1000 most active users. This resulted in approximately 2900 objects in each scene. Next, we calculated the average request rate in the scene, and used only the traces whose average request rate was within 0.8 to 1.2 of the scene's average rate. This selection was required for evaluating the specific model in the paper, and resulted in 2504 traces in total.

Unique features of WoW-IO traces. The analysis in [35] consisted of several datasets: small and large synthetic traces and WoW-IO traces. The synthetic traces featured a Poisson arrival process and multiple Zipf distributions, which are common in storage traces [43]. However, when we compared the synthetic traces to WoW-IO traces, we observed the following distinct characteristics.



(a) CDF of scene duration (b) CDF of GET request rate (c) CDF of PUT request rate

(d) CDF of number of objects (e) CDF of number of avatars (f) CDF of the GET/PUT ratio

Fig. 3: Main characteristics of the 158 I/O traces generated from the WoW-Avatar scenes.

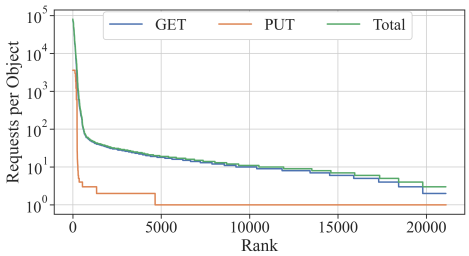


Fig. 4: Object request distribution in scene 1 in log scale.

Bursty demand. Synthetic I/O traces are often smooth, without spikes in demand for specific objects. Such traces are often generated with Poisson arrival process [43]. In practice, many modern applications, and especially video games [41], feature short but intensive bursts of demand for specific objects. Even community-standard storage benchmarks such as YCSB [42] fail to adequately mimic this hotspot behavior [45]. WoW-IO traces contain such bursts, for example in large avatar gatherings, where each user reads the objects of the avatars in its location. Bursts are responsible for temporary queue overflows, which require requests to be dropped. We learned that the behavior of such bursts is difficult to predict and reduces the theoretical model's accuracy.

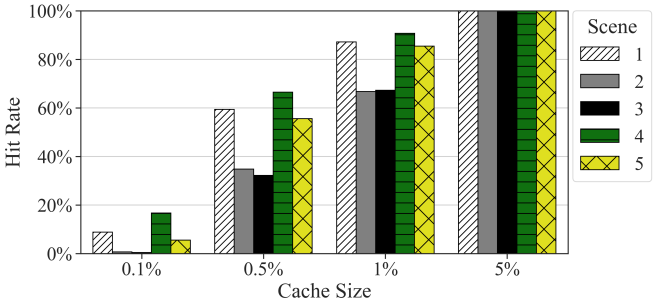
Inactive objects. The first scene in table II contains 2913 objects, but only an average of 1350 objects were read in any 10-minute trace. This means half of the objects stored in the system were inactive in each scene. We observed a similar pattern in all the scenes. Real gaming workloads also exhibit similar behavior, in which only a part of the game data is accessed [41]. In WoW-IO, location objects are read whenever avatars are located in these locations. If at a certain period, no avatar is present at a location, the location object is not read but still might be stored in the system. The need to allocate storage space for such temporarily "frozen" objects was not demonstrated in the synthetic workloads, where each object in the trace was accessed at least once.

Realistic object popularity. WoW-IO traces were more skewed than the synthetic workloads. For example, observing the 20% most popular objects in each trace showed that they were accessed 2.5× more in the WoW-IO traces. The reason is the high percentage of inactive objects in these traces, combined with higher popularity of the most popular objects. This more skewed demand contributed to the existence of hotspots in the nodes containing the most popular objects. Analysis of other video games also showed similar behavior, where some data tend to be much more popular than other [41] We learned such hotspots further contribute to dropped requests and more sophisticated object placement and redundancy techniques are required to mitigate the hotspot effect.

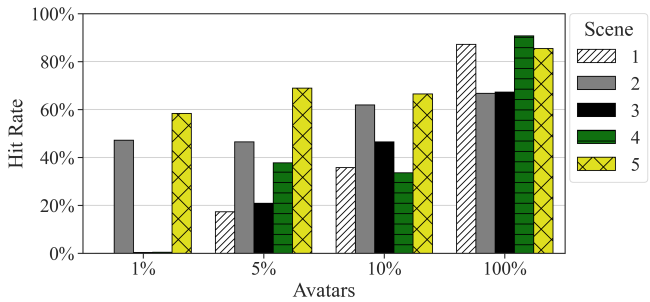
B. Edge caching

Perhaps the best example of the importance of I/O traces is their central role in the design and optimization of cache-management algorithms. This case study represents a hypothetical research and not a concrete experience. As a proof-of-concept, we implemented a basic cache simulator with least-recently used eviction and used five representative I/O traces as input. Our simulator handled GET and PUT requests for objects of unit size. We used the first 60 minutes from each scene, with the first minute dedicated to cache warm-up. We executed the simulator with several cache sizes, given as a percentage of the number of unique objects in each trace. The characteristics of these traces are summarized in Table I.

Figure 5(a) depicts the hit rate with different cache sizes. As expected, the hit rate is 100% for a cache size of 5%. Recall that in each second, users request objects representing the avatars in their vicinity and in their guilds, which results in repeated reads of similar objects. Caches smaller than 1% achieve significantly lower hit rates. We repeated this experiment with the first 60 minutes of all 158 scenes, and a cache size of 1% of the objects in each scene. Figure 6 shows the CDF of the resulting hit rate. While the lowest hit



(a) Cache size as a percentage of the number of objects



(b) Percentage of remaining avatars from each original trace

Fig. 5: LRU cache evaluation.

Period	Date	Duration	Included traces	Unique objects	Selected users	Mean request rate
1	6/2007	27 days	1621	2913	1000	14326 reqs/sec
2	3/2008	21 days	883	2875	1000	11611 reqs/sec

TABLE II: Traced periods generated with WoW-IO for [35].

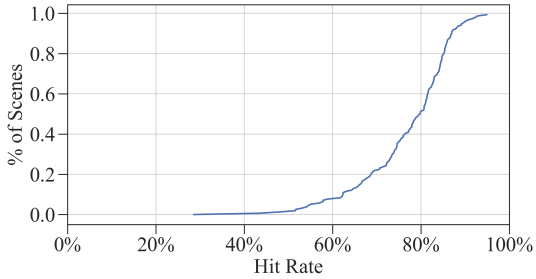


Fig. 6: Hit rate CDF of all scenes.

rate is 29%, half of the scenes have a hit rate of more than 80%. The difference between scenes can be attributed to the variations in the number and behavior of the active avatars.

An edge system topology can be viewed as a tree, where each leaf is an edge node. Each node serves the demand of the user population in its range, while the collection of all edge nodes serves the entire demand of all users which comprises the system's workload [46]. In the next experiment, we show how the WoW-IO workloads can be used to evaluate a cache in a single edge node. We randomly removed some of the avatars in each scene, which also reduced the number of objects and accesses to them in the resulting I/O traces. We repeated this process with different numbers of avatars removed. Figure 5(b) depicts the hit rate in a cache size of 1% of the remaining objects in each trace.

The results confirm that the high skew in object popularity is the result of the avatars' distribution across the map locations and guilds, as users read objects of avatars with whom they share a location or guild. They also indicate that increasing the number of avatars is unlikely to change this distribution dramatically. The trace generator can be configured with different distributions of avatar locations (see Section II-A) to generate traces with different skew in object popularities.

This is the simplest example of using the generated object traces for cache evaluation. More complex algorithms might also consider the data-sharing patterns between users or prefetching possibilities, e.g., of adjacent map locations. These optimizations rely on the object semantics, which are typically missing from anonymized public traces. Cache coherency mechanisms can be evaluated by extending WoW-IO to generate a mix of read and write (or get and put) requests.

C. Concurrency control

Concurrency control mechanisms such as locking and lock-free protocols must be verified rigorously. Their correctness is proved by formal methods, but their effect on the system's performance depends on the workload it serves. Specifically, the pattern and frequency in which shared data objects are accessed by individual users determine the time spent waiting for locks or retrying aborted transactions. Benchmarks and traces are thus often used for debugging the protocol implementation and evaluating its performance. For example, I/O traces were used to evaluate a mechanism for eventual consistency [47] and for a consistent-hashing key-value store [48]. In the context of online multiplayer games such as WoW, previous works used proprietary [49] and synthetic [50] traces to evaluate their architectures and models. WoW-IO can serve as a publicly available input for such evaluations.

D. Additional uses

In the current version of WoW-IO, we focused on modeling the game's map, user locations, and user mobility in an easy-to-use framework. The resulting I/O traces include the minimal attributes for basic evaluation purposes. Several trivial extensions can be applied to create richer trace formats. For example, an attribute indicating an object's size can be added, where the size of an object will depend on its type (location or avatar) and properties (e.g. number of quests in a city or an avatar's skill and guild members), or change dynamically [51]. Each avatar representation can be expanded into multiple objects, for example, to represent the avatar's inventory. *Game data* objects can be added to include world appearance and avatar metadata (name, race, appearance, etc.) [39]. Objects' time-to-live (TTL) can add an aspect of expiration time to objects, allowing the system to remove them. Request priorities

can be added to reflect the importance of different operations. More sophisticated (though easy-to-implement) extensions can generate more objects for "busy" locations (e.g., add quest objects to the city locations), or have avatars with higher skill move faster or generate more object updates (i.e., write requests). WoW-IO is published as an open-source project to allow these and similar extensions by the community[1].

IV. Related Work

The absence of edge workloads has been observed and discussed in detail by Kolosov et al. [44], and, to the best of our knowledge, remains a challenge. To address this challenge, they proposed *workload composition*, i.e., joining existing datasets to obtain attribute combinations that are otherwise unavailable. They demonstrate the creation of a comprehensive edge workload describing Wikipedia 'browsing sessions' of users riding New York City (NYC) taxis. The resulting edge workload is a combination of multiple datasets: NYC hotspots represent edge nodes, Wikipedia pages represent objects and their popularities, and NYC yellow taxi dataset represents users and their locations. Our use of the WoW-Avatar traces in WoW-IO can be viewed as another instance of workload composition, where we combine information from real gaming traces with synthetic distributions of (in-game) map locations and (physical) user locations.

Toczé et al. [52] characterize edge use cases and identify representative workload classes for a realistic edge benchmark. Similar to our approach, they focus on user-level information by deriving workloads directly from edge applications. However, they do not generate storage-related (I/O) traces. TPCx-IoT [53] is a benchmark for measuring the performance of IoT gateway systems, and is based on synthetic workloads. Riotbench [54] is an IoT benchmark that uses workloads derived from observations on smart cities and smart health. Their approach is similar to ours: the derived workloads are augmented synthetically to include more aspects.

Storage trace generators are commonly used as benchmarking tools for storage-system components such as caching, mapping schemes, storage devices, databases, and file systems. Typically, these tools generate synthetic workloads to reflect various desired behaviors, such as object popularity, read/write intensity, and trace duration.

The YCSB [42] benchmarking suite is extensively used for cloud services and key-value stores. It includes Zipf-based and uniform workloads that mimic web and cloud environments like websites or email services. YCSB supports client deployment across multiple machines for benchmarking distributed systems [55]. The traces generated by WoW-IO can be used for similar purposes by mapping the requests to distributed users within the system [35].Flexible I/O (FIO) [56] is another popular workload generator. Though it creates synthetic workloads, it can used to replay real traces to replicate real-world I/O patterns. Filebench [57] is a file system and storage benchmark tool. It creates synthetic file system workloads and can be used to test the performance of file systems and storage devices. In addition to read and write, it supports a range of operations, including file creation, opening, closing, and deletion. These tools were not designed to represent edge-system workload distributions and do not include related user-level information.

Tools like blktrace [58], SystemTap [59], and DTrace [60] capture I/O events for trace analysis and creation. However, they primarily record device activity and do not directly represent user behavior. To the best of our knowledge, WoW-IO stands out as the only object trace generator built on actual gaming user actions, thus filling a crucial gap in realistic workload generation for edge use cases.

Gaming-based traces are commonly used to evaluate user in-game behavior [30], [31], [32]. The information that they capture cannot be directly used for evaluating the systems aspects of video games. Namely, such analysis entails knowledge of in-game operations with respect to the analyzed system elements, such as storage, memory, or network. Suznjevic et al. [33] monitored the network traffic of six individual users to derive the effects of specific game actions, Chen et al. [61] analyze the effect of network quality on gamers' quality-of-experience (QoE), and Patro et al. [34] implemented a measurement framework to evaluate battery and CPU utilization of a massively multiplayer online game on a mobile phone. The traces in these studies include information about gameplay time and in-game user actions. While reflecting the user-level information, they are unfortunately inappropriate for reflecting the storage requirements of their clients. In addition, edge systems are expected to serve thousands of users in parallel, however, these workloads do not reflect such demands.

V. Conclusions

To the best of our knowledge, WoW-IO is the first gaming-based I/O trace generator for edge storage systems. Derived from real user gameplay, WoW-IO enables the creation of diverse traces with realistic attribute combinations. It addresses the scarcity of edge workloads and facilitates practical edge research. As an open-source tool, we expect community involvement and future work to further expand its applicability.

[1] github.com/olekol33/WoW-IO

References

[1] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," *Trans. Storage*, vol. 3, no. 3, Oct. 2007.

[2] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *USENIX Conf. on File and Storage Technologies (FAST)*, 2011.

[3] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *USENIX Conf. on File and Storage Technologies (FAST)*, 2003.

[4] G. Yadgar, M. Factor, K. Li, and A. Schuster, "Management of Multilevel, Multiclient Cache Hierarchies with Application Hints," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, pp. 5:1–5:51, 2011.

[5] T. Pritchett and M. Thottethodi, "SieveStore: A highly-selective, ensemble-level disk cache for cost-performance," in *Annual Intl. Symp. on Computer Architecture (ISCA)*, 2010.

[6] D. Zhou, W. Pan, W. Wang, and T. Xie, "I/O characteristics of smartphone applications and their implications for eMMC design," in *IEEE Intl. Symp. on Workload Characterization (IISWC)*, 2015.

[7] J. Mohan, D. Purohith, M. Halpern, V. Chidambaram, and V. J. Reddi, "Storage on your smartphone uses more energy than you think," in *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2017.

[8] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *Trans. Storage*, vol. 4, no. 3, pp. 10:1–10:23, Nov. 2008.

[9] P. Desnoyers, "Analytic models of SSD write performance," *Trans. Storage*, vol. 10, no. 2, pp. 8:1–8:25, Mar. 2014.

[10] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The unwritten contract of solid state drives," in *European Conf. on Computer Systems (EuroSys)*, 2017.

[11] M. Linaje, J. Berrocal, and A. Galan-Benitez, "Mist and edge storage: Fair storage distribution in sensor networks," *IEEE Access*, vol. 7, pp. 123 860–123 876, 2019.

[12] H. Jin, R. Luo, Q. He, S. Wu, Z. Zeng, and X. Xia, "Cost-effective data placement in edge storage systems with erasure code," *IEEE Trans. Serv. Comput.*, 2022.

[13] Y. Liu, Y. Mao, X. Shang, Z. Liu, and Y. Yang, "Distributed cooperative caching in unreliable edge environments," in *IEEE Conf. on Computer Communications (INFOCOM)*, 2022.

[14] R. Zhou, X. Wu, H. Tan, and R. Zhang, "Two time-scale joint service caching and task offloading for uav-assisted mobile edge computing," in *IEEE Conf. on Computer Communications (INFOCOM)*, 2022.

[15] G. Yadgar and O. Kolosov, "Data storage at the edge: Challenges and opportunities," *IEEE BITS the Information Theory Magazine*, 2024.

[16] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production Windows servers," in *IEEE Intl. Symp. on Workload Characterization (IISWC)*, 2008.

[17] O. Eytan, D. Harnik, E. Ofer, R. Friedman, and R. Kat, "It's time to revisit LRU vs. FIFO," in *USENIX Conf. on Hot Topics in Storage and File Systems (HotStorage)*, 2020.

[18] "Alibaba cluster trace," https://github.com/alibaba/clusterdata.

[19] J. Yang, Y. Yue, and K. Rashmi, "A large scale analysis of hundreds of in-memory cache clusters at Twitter," in *USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, 2020.

[20] G. Yadgar, M. Gabel, S. Jaffer, and B. Schroeder, "SSD-based workload characteristics and their performance implications," *ACM Transactions on Storage (TOS)*, vol. 17, no. 1, pp. 1–26, 2021.

[21] D. Zhou, W. Pan, W. Wang, and T. Xie, "I/O characteristics of smartphone applications and their implications for eMMC design," in *IEEE Intl. Symp. on Workload Characterization (IISWC)*, 2015.

[22] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production Windows servers," in *IEEE Intl. Symp. on Workload Characterization (IISWC)*, 2008.

[23] O. Kolosov, G. Yadgar, D. Breitgand, and D. H. Lorenz, "Pase: Proactive service embedding in the mobile edge," in *IEEE 43rd Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2023.

[24] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE Conf. on Computer Communications (INFOCOM)*, 2020.

[25] L. Gu, Z. Chen, H. Xu, D. Zeng, B. Li, and H. Jin, "Layer-aware collaborative microservice deployment toward maximal edge throughput," in *IEEE Conf. on Computer Communications (INFOCOM)*, 2022.

[26] R. Han, S. Wen, C. H. Liu, Y. Yuan, G. Wang, and L. Y. Chen, "EdgeTuner: Fast scheduling algorithm tuning for dynamic edge-cloud workloads and resources," in *IEEE Conf. on Computer Communications (INFOCOM)*, 2022.

[27] I. Jaya, W. Cai, and Y. Li, "Rendering server allocation for MMORPG players in cloud gaming," in *49th Intl. Conf. on Parallel Proc.*, 2020.

[28] Y. Xu, Q. Shen, X. Li, and Z. Ma, "A cost-efficient cloud gaming system at scale," *IEEE Network*, vol. 32, no. 1, pp. 42–47, 2018.

[29] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, "Improving cloud gaming experience through mobile edge computing," *IEEE Wirel Commun*, vol. 26, no. 4, pp. 178–183, 2019.

[30] Y.-T. Lee, K.-T. Chen, Y.-M. Cheng, and C.-L. Lei, "World of Warcraft avatar history dataset," in *ACM Conf. on Multimedia systems*, 2011.

[31] M. Suznjevic, I. Stupar, and M. Matijasevic, "MMORPG player behavior model based on player action categories," in *IEEE Annual Workshop on Network and Systems Support for Games*, 2011.

[32] R. van de Bovenkamp, S. Shen, A. L. Jia, F. Kuipers *et al.*, "Analyzing implicit social networks in multiplayer online games," *IEEE Internet Computing*, vol. 18, no. 3, pp. 36–44, 2014.

[33] M. Suznjevic, O. Dobrijevic, and M. Matijasevic, "MMORPG player actions: Network performance, session patterns and latency requirements analysis," *Multimedia Tools and Apps.*, vol. 45, pp. 191–214, 2009.

[34] A. Patro, S. Rayanchu, M. Griepentrog, Y. Ma, and S. Banerjee, "Capturing mobile experience in the wild: A tale of two apps," in *ACM Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013.

[35] O. Kolosov, M. F. Aktaş, E. Soljanin, and G. Yadgar, "Theory vs. practice in modeling edge storage systems," in *IEEE Intl. Performance, Computing, and Communications Conf. (IPCCC)*, 2023.

[36] H. Wang, Z. Zhang, M. N. A. Khalid, H. Iida, and K. Li, "MMORPG evolution analysis from explorer and achiever perspectives: A case study using the final fantasy series," *Information*, vol. 12, no. 6, p. 229, 2021.

[37] "Latest World of Warcraft player count & subscription numbers," https://headphonesaddict.com/world-of-warcraft-player-count/.

[38] S. Shen, N. Brouwers, A. Iosup, and D. Epema, "Characterization of human mobility in networked virtual environments," in *Network and Operating System Support on Digital Audio and Video Workshop*, 2014.

[39] Z. Diao, E. Schallehn, S. Wang, and S. Mohammad, "Cloud data management for online games: potentials and open issues," *Datenbank-Spektrum*, vol. 13, pp. 179–188, 2013.

[40] "World of warcraft: Classic," https://wowpedia.fandom.com/wiki/World_of_Warcraft:_Classic.

[41] A. Maruf, Z. Yang, B. Davis, D. Kim, J. Wong, M. Durand, and J. Bhimani, "Understanding flash-based storage I/O behavior of games," in *IEEE Intl. Conf. on Cloud Computing (CLOUD)*, 2021.

[42] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *ACM Symp. on Cloud Computing (SoCC)*, 2010.

[43] K. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran, "EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding," in *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2016.

[44] O. Kolosov, G. Yadgar, S. Maheshwari, and E. Soljanin, "Benchmarking in the dark: On the absence of comprehensive edge datasets," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*, 2020.

[45] C. Claesen, A. Rafique, D. Van Landuyt, and W. Joosen, "A YCSB workload for benchmarking hotspot object behaviour in NoSQL databases," in *TPC Tech. Conf. on Perf. Eval. and Benchmarking (TPCTC)*, 2022.

[46] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. De Lara, "Cloudpath: A multi-tier cloud computing framework," in *ACM/IEEE Symp. on Edge Computing (SEC)*, 2017.

[47] F. Yan, A. Riska, and E. Smirni, "Fast eventual consistency with performance guarantees for distributed storage," in *IEEE Intl. Conf. on Distributed Computing Systems Workshops*, 2012.

[48] N. Zhao, J. Wan, J. Wang, and C. Xie, "GreenCHT: A power-proportional replication scheme for consistent hashing based key value storage systems," in *IEEE Symp. on Mass Storage Systems and Technologies (MSST)*, 2015.

[49] M. Assiotis and V. Tzanov, "A distributed architecture for MMORPG," in *ACM SIGCOMM workshop on net. and sys. support for games*, 2006.

[50] Z. Diao, P. Zhao, E. Schallehn, and S. Mohammad, "Achieving consistent storage for scalable MMORPG environments," in *Intl. Database Engineering & Applications Symp.*, 2015.

[51] J. Yang, Y. Yue, and K. Rashmi, "A large scale analysis of hundreds of in-memory cache clusters at Twitter," in *USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, 2020.

[52] K. Toczé, N. Schmitt, U. Kargén, A. Aral, and I. Brandić, "Edge workload trace gathering and analysis for benchmarking," in *IEEE Intl. Conf. on Fog and Edge Computing (ICFEC)*, 2022.

[53] M. Poess, R. Nambiar, K. Kulkarni, C. Narasimhadevara, T. Rabl, and H.-A. Jacobsen, "Analysis of TPCx-IoT: The first industry standard benchmark for IoT gateway systems," in *2018 IEEE 34th Intl. Conf. on Data Engineering (ICDE)*, 2018.

[54] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: An IoT benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017.

[55] D. Bermbach, E. Wittern, and S. Tai, *Cloud service benchmarking*. Springer, 2017.

[56] "fio - flexible I/O tester," https://fio.readthedocs.io/.

[57] T. Vasily, "Filebench: A flexible framework for file system benchmarking.; login," *The Usenix Magazine*, vol. 41, p. 6, 2016.

[58] "Block IO tracing," https://github.com/efarrer/blktrace/.

[59] F. C. Eigler and R. Hat, "Problem solving with systemtap," in *Proc. of the Ottawa Linux Symp.*, 2006.

[60] B. Gregg and J. Mauro, *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X, and FreeBSD*. Prentice Hall Professional, 2011.

[61] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?" *Communications of the ACM*, vol. 49, no. 11, pp. 34–38, 2006.