Render Cost Driven Caching for Object-Based Media

Paul Dean
School of Computing and Communications
Lancaster University
Lancaster, U.K.
p.a.dean1@lancaster.ac.uk

Rajiv Ramdhany
BBC R&D
Media City, U.K.
Rajiv.Ramdhany@bbc.co.uk

Barry Porter
School of Computing and Communications
Lancaster University
Lancaster, U.K.
b.f.porter@lancaster.ac.uk

***Abstract*—The caching of popular content in fast-access memory has long been a staple of performance optimisation. A common assumption of cache protocols is that a cache miss generates a roughly equal level of work for any item. In this paper, we introduce a novel caching problem in which this assumption does not hold, in the emerging domain of Object-Based Media (OBM). Using OBM, a media experience is segmented and decomposed into individual objects, layers, and assets. These elements are then dynamically composed, during playback, by each user according to their preferences. In this domain, cache misses while generating a personalised media rendition will have highly varying costs depending on how expensive it is to render that segment of media. We analyse the challenges of caching based on render cost variability for object-based media delivery within self-adaptive systems, and present our work on a novel approach to cost-based cache partitioning and policy selection to reduce the total render cost. We present experimental results for the challenge of determining element retention in a cache, and identify a key challenge in determining retention rates via frame rate, request pattern, and GPU utilisation, creating a cost value per media object dependent on both velocity and render complexity. We examine different cache compositions using a single policy for uniform and non-uniform memory per object, with comparison to a recently proposed multi-queue implementation, studying the impact of partition adjustment. Our results demonstrate that cost-informed cache partitions produce a lower render cost for synthetic workload traces, lowered further through partition adjustment, despite a higher miss rate.**

I. Introduction

Traditional IP-based media delivery, from media streaming services like Netflix or Disney+, uses segmented video content with variable bitrates, using protocols such as DASH. This works by taking a final produced media entity such as a TV show, segmenting that media into (for example) 10-second chunks, and encoding a set of different bitrates for each chunk to support various client network characteristics. Each chunk at each bitrate has a URI and is delivered over HTTP. Media delivery in this paradigm is supported by content distribution networks, which operate distribution nodes around the world, such that clients pull data from their closest distribution node. Those distribution nodes internally use in-memory caches, which cache popular content to improve access times vs. loading that content from secondary storage such as disks.

The optimisation of caching becomes an increasingly complex problem operating at the data-centre scale handling terabytes of data and user requests [1]. Effective use of a cache's content is defined through a *hit ratio*: a hit allows previously-used data to be shared directly from fast memory with low-latency; whereas a miss forces an additional expensive via retrieval from disk. A range of optimisation techniques have been employed to aid in cache effectiveness; these include enhancing throughput in application-specific low latency workloads [2]; improving hit ratio of tiered caches within distributed systems [3]; or reducing access time to in-memory objects [4]; and approaches that are media-specific [5], [6].

In this paper, we consider the emerging media delivery paradigm of *object-based media* (OBM) and its affect on the caching optimisation problem. OBM is aimed at providing flexibility in how media experiences are composed, such that users can customise what they are watching from a rich set of permutations. To support this, OBM separates a media experience into its individual objects (such as actors or presenters, backgrounds, camera angles, and overlays) and those objects are combined in a customised way for each user in real-time. Each frame of an OBM experience is therefore dynamically derived and requires both a *compute* and *presentation* stage, compared to the presentation-only delivery of traditional media streams. A key enabling technology for OBM is the ability for some users to *offload* the compute needed to render some or all of a particular experience variant, in cases where their own device is not computationally powerful enough to perform that rendering in real-time. This may occur for high-demand media objects such as customised synthetic presenters, interpolated camera angles, or picture-in-picture scenarios where multiple concurrent video decoders are needed. For content delivery systems, this offload creates a compute-bound environment rather than a primarily IO-bound one: in an on-demand way, content-delivery systems must load content for requested offloaded media objects, render the frames of those objects to a video format, and deliver that video to a client.

In this context, the cache equation for content delivery nodes is fundamentally different. Typical systems for traditional media use a simple recency- or frequency-based approach to cache the portion of video segments which have most recently or most frequently requested. These approaches assume that the cost of a cache miss is roughly equal for any miss; in this context, maximising hit-rate is equivalent to minimising overall cost. In OBM with compute offloading, the cost of cache misses is highly variable depending on the rendering complexity of the media objects that clients are requested a remote render for. In this context, maximising overall hit-rate is no longer equivalent with globally reduced cost; instead, we have a far more nuanced optimisation landscape depending on the cache-miss costs of individual entities.

In this paper we make the following contributions:

- We demonstrate that the use of cache hit-ratio as a metric is insufficient in minimising compute cost in offloaded OBM scenarios;
- We introduce a split cache design which supports the division of total cache space into bins for different render cost;
- We demonstrate that finding the ideal bin count, and per-bin size, results in significant reductions in total compute cost at offload sites;
- We show that finding the ideal bin-count and per-bin size is a novel dynamic problem in need of real-time adaptation or optimisation processes.

Our evaluation is conducted on a mixture of synthetic data, which demonstrates the theoretical nature of the problem, and on real-world traces from a large media streaming organisation. In future work we intend to examine the novel problem we identify in this paper to develop potential solutions.

II. Related Work

The improvement of caching has remained a prominent area of research within computing systems, for small- and large-scale deployments [1], [7]. Optimisations have been proposed to widely varying aspects such as improving cache hit ratio [7], lowering the latency of requests [8], reducing internal communication overhead [9], or attack prevention [10]. Specific optimisations to caches are dependent on workloads and properties which can be characterized by traffic pattern, time-to-live (TTL), popularity distribution, and size distribution [1]. The delivery of streamed media is one recent focus on cache optimisation due to the difficulty in understanding both the composition of the workload [5] and the unpredictable characteristics of the workload, compared to the rendering of large streams for an individual device [11]. Further difficulties include creating accurate workload traces to show the effects on a range of cache performance metrics such as hit rates within a workload sequence [12].

In this section we focus on research that considers improvements via hit ratio and cache latency. Improvements to caching effectiveness can lower client request latency [8], lower computation through reuse [1], [13], [14], and through reducing load in large systems we may gain lower operating costs and energy usage [15].

One approach to improving caching is to take a server-centric view and minimise load on a server. One metric to improve caching is cache hit rate, which provides insight into the effectiveness of a cache as objects from a workload are periodically stored, creating a store of objects which are no longer recomputed. However, there is difficulty in ensuring the cached contents present are representative of the task popularity and the resources saved from caching an object, as minor changes in cache allocation can cause large changes in the hit rate [7]. Previous work on the caching of media content for end users has highlighted difficulties in attempting to generalise object contents to explore uniformity, an even more difficult task within OBM as requests are now split among individual parts of a single frame rendered in different locations [5].

Operations performed on content shared to a large user pool also present unique challenges with caching, as a small number of requests in a workload may create peaks in network usage: 3% of requests can account for almost half of all data downloaded in certain traces [11]. However, through a large client pool, there may be an increase in the average temporal locality of cached objects as requests may also be co-located to maintain a higher percentage of sharing among client requests [9], [16]. Evaluating a cache's accuracy presents another layer of difficulty as computing a hit ratio for varying object sizes results in weak boundaries which may be unusable in closed training data sets and a need for an approach to learning in real-time to sporadically form a previously unseen composition of tasks [17].

Adaptation of eviction methods within caching has previously been performed to minimise latency. This is due to workloads changing over time and the deployment of resources affecting the Time-To-Live of cached content. Previous approaches have explored synthesising data locality and miss penalties to inform current decisions, relying on dynamically switching policies to maintain a lower latency average [4]. Further optimisation may be made to individual policies for workload-specific improvements, such as the generalisation of caching policies in place to exploit handling of a specific property to policy, for example, uniform cache object size with LRU Generalisation. However, within OBM the task type and cached data object may vary widely and a generalised approach may not effectively cover all permutations found.

Learning approaches have been utilised to aid in the adaptation of a caching approach, with model-based approaches relying on specific information available for all workloads and synthesising metrics for decisions from the combination of historical run time information with job dependency graphs [18]. Offline learning approaches, using purely machine learning approaches have previously been explored, avoiding the creation of heuristics from policies such as LRU and introducing custom metrics for a specific element of cache optimisation, for example forming a good decision metric from ML-defined boundaries to enforce eviction [19].

III. Approach

End-user player software for traditional streamed media is essentially comprised of a HTTP client with a single video decoder instance, together with some bitrate adaptation logic to choose between different bitrates of each video segment. OBM-based experiences entail a much wider variety of render types: they may require multiple video decoders, animation format renderers, image manipulation such as chroma-keying to layer presenters over backgrounds, or more advanced neural rendering transformations.

Our OBM player software is therefore composed of a suite of rendering components, with one component for each type of content that the player can render [20], [21]. Each of these rendering components can run either on the user's device, or can be offloaded to edge or cloud resources to stream rendered frames back to the client for presentation. Rendering components that have been moved to server-side resources may benefit from caching of rendered outputs, in cases where multiple clients watching the same content have offloaded the same rendering component to that server.

The server implementation is realised as a virtual 'resource container' in Docker image. It receives client requests to offload render tasks, processes those requests and delivers rendered frames; it can adapt between different cache policies depending on the mixture of tasks it is being asked to do, from simple ones such as rendering subtitles to far more intensive tasks such as rendering digital human presenters. For the purpose of this paper our client-side is kept in a single, manually-chosen composition, in which selected render tasks are offloaded to selected remote sites, and our server-side containers provide multiple caching policy implementations, such as LRU and FIFO.

Fig. 1 shows a subset of the components used by the resource container and client, focusing on the elements for which relevant variation is available for our study (in reality both systems are composed of over 30 components). Fig. 2 illustrates track offloading, as the client requests render tasks while streaming an OBM media experience, whereas the server-side allocates resources for those tasks and assigns them to specific containers. This allows the client to dictate when computation can be usefully offloaded and submitted as a task to the wider OBM delivery system.

Our experiments in this paper use a flexible weather forecast media experience as an example, which is composed of a map, weather iconography, presenters, time-of-day imagery, and subtitles, all of which have degrees of configurability. Our resource container architecture is designed to allow many client requests to be concurrently serviced to offload these entities for remote rendering. It allows for there to be differing compositions for the completion of differing tasks, such as rendering subtitles or on-screen graphics. Each separate resource container may then differentiate its chosen caching policy, adding a layer of runtime-interchangeable components. Our study in this paper considers three different render tasks: subtitles, an animated map, and an onscreen time-image graphic overlay. We then use a range of different caching policies alongside a variety of different client workloads to understand caching characteristics.

A. Traditional Cache Replacement Policies

We use a set of traditional cache policies as a starting point, and examine their characteristics when subjected to OBM streams which have partially or fully offloaded compute. We also introduce a customised cache policy in which render cost (or more generally *cache miss cost*) can be explicitly modelled.

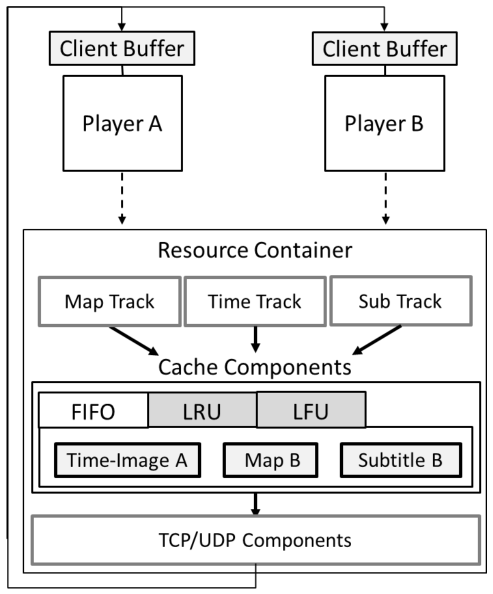

Fig. 1. A simplified overview of component composition for a Resource Container, including caching policy components.

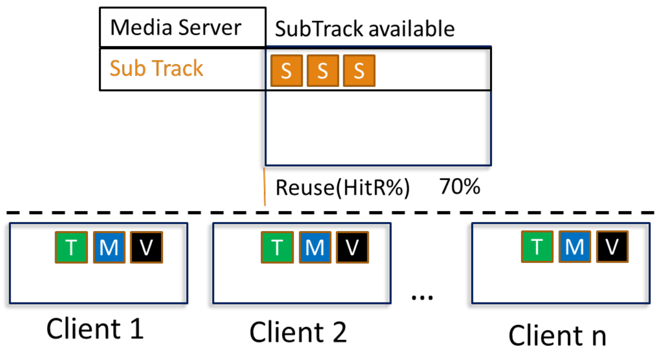

Fig. 2. Example architecture, with client tracks offloaded to a Resource Container.

In traditional caching, the main differentiator of caching algorithms is the *eviction policy*. A cache is configured to have a maximum memory occupancy (such as 1GB), and every time an element is requested the cache is checked to see if that element is present. In an OBM context this cached entity is a rendered time segment of one or more media objects, and a cache miss results in a render of that segment at the server. After a cache miss the result is then added to the cache. At the point of adding an element to the cache, if the cache contains a volume of data that would places it above its memory limit when the new element is added, an *eviction policy* is triggered to remove existing items from the cache until there is sufficient space for the new item.

Our first traditional caching eviction policy is First-In-First-Out (*FIFO*), which replaces the oldest cached entry within the cache with the most recent element decision. Newly-added items are simply placed at one end of the cache list, and evicted items are taken from other end. No changes in ordering are made when a cache hit occurs (i.e., when a requested item is found to be in the cache and is therefore used in a response to a client request).

Our second policy is Least Recently Used (*LRU*), which is a commonly-used approach for caching within distributed systems and traditional media delivery [5]. In LRU, new items are added at one end of the cache list, and evicted items are taken from the other end, in a similar way to FIFO. However, when a cache hit occurs, that element is moved to the new-items end of the cache list, making it stay in the cache for longer. This maintains a cache dependent on temporal proximity, thereby relying on requests maintaining close temporal proximity to gain effective cache use.

Our third cache replacement policy is Least Frequently Used (*LFU*), which promotes elements to stay in the cache longer depending on their popularity (rather than recency). LFU uses a hit counter on each cached item to record its access frequency; when a replacement decision is needed the content with the lowest frequency count is removed. This creates a caching policy which can potentially maintain a cache set which covers a larger (but discontinuous) temporal window, based on individual entity popularity.

Finally, we use an implementation of a recently-proposed high-performance caching policy for traditional caching contexts, called S3-FIFO [22]. This policy divides the total cache space into three separate FIFO queues, and moves items between those queues depending on their access patterns. When an item first enters the cache it is placed in a 'small' queue, and only if it is accessed a second time does it enter the 'main' queue; items are preferentially retained in the main queue if they have accessed multiple times up to a cap.

B. Cost Aware Caching Policies

Beyond the above traditional cache policies, we also introduce novel policies which have the potential to account for variable costs of cache misses. In contrast to the general assumption in traditional caching that each cache miss has a roughly equal cost (e.g. to read an element from a disk), our novel cache policy supports per-item cache miss costings. This may accommodate our OBM scenario in which rendering different objects in offloaded compute may have very different costs. We experiment with two different cost-aware policies.

Our first, called Render Cost First-in-First-Out (*RC-FIFO*), demonstrates the effect a policy-level prioritisation of render cost on throughput and hit ratio. In RC-FIFO, new items are placed at the head of the cache list, and evicted items from the tail. The order is changed upon insertion by a render cost value, and placed at the head of the queue if equal to or greater than the current head entry's render cost, retaining high-cost entries for a longer duration.

Our second policy divides the total cache space into different cost bins, where each bin takes a portion of the available storage space. Each bin is assigned a cost range, where all items in that cache-miss-cost-range are placed into that bin. In our current design, eviction decisions are made within the specific bin that an item will be placed, such that each bin has a fixed size. Alternative designs could consider adjusting bin sizes dynamically to consider eviction across the entire cache due to metrics such as average popularity. While each bin could use its own eviction policy, for this paper we use LRU within each bin.

C. Trace Generation

The evaluation of cache effectiveness requires a list (or 'trace') of client requests to be run against the cache, allowing us to determine a hit ratio for that trace. We use two different sources of traces for our experiments: a synthetic one and a real-world one.

Our synthetic trace allows us to control variables of interest, such as temporal similarity windows between groups of clients, to examine basic caching characteristics. Our real-world trace then allows us to examine how these characteristics manifest in a more realistic setting.

Our synthetic trace is developed by using a real datacentre with real clients, but controlling the rate and volume of client startup to be within a particular distribution. From these clients we record the ordering of offload requests that arrive at our offload site. We then re-play this ordering of requests to different cache implementations in isolation to measure their effectiveness. The compute cost of each cache miss is based on real render cost ratios between different media objects in our flexible weather forecast presentation example.

Our real-world trace is a log of all client requests over a number of days during 2024 from the BBC iPlayer streaming service. Because this trace is extremely large (many terabytes), we sub-sample the trace at a uniform rate for our experiments.

IV. Evaluation

Our evaluation is conducted by playing our above traces against various cache policies on a single node. We constrained the maximum capacity of each cache implementation to be 2% of a given trace in bytes (this is comparable to related work in this area [22]). In each experiment we measure hit ratio, as in traditional caching, but also measure the total cost of cache misses (the work needed to render frames of an experience). Compared to traditional caching objective functions, depending on the scenario, in OBM offloading it is possible that a *lower* hit ratio actually yields a better score for total cache miss cost, depending on which cached items are missed or hit and their associated relative costs.

A. Synthetic Trace

We begin with our synthetic trace, which allows us to create controlled conditions to examine particular effects.

In order to form our synthetic trace we initially deployed one server node containing a single resource container 2, and 5 client nodes each deploying 100 instances of a client. Each client instance composes 3 offloaded media object tracks, resulting in 1,500 concurrent user request streams to the media server for the entirety of their requested viewing experience.

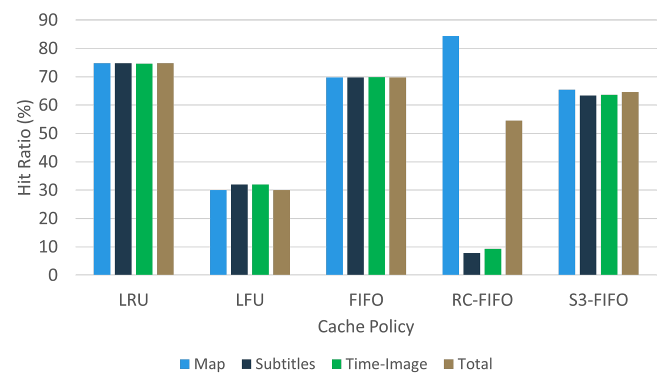

Fig. 3. Synthetic Trace Cache Policy Comparison

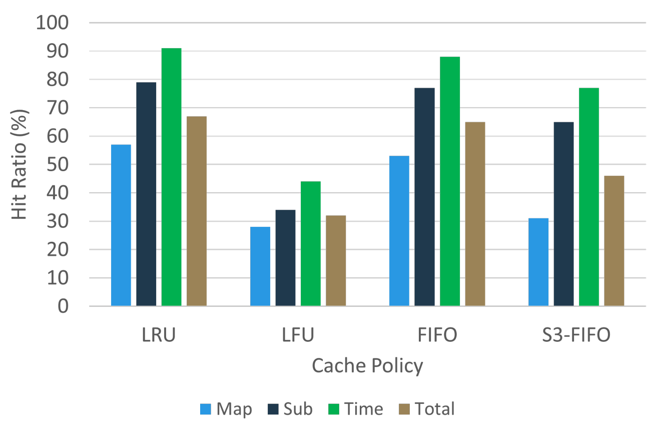

Fig. 4. Track Render Cost Split Cache

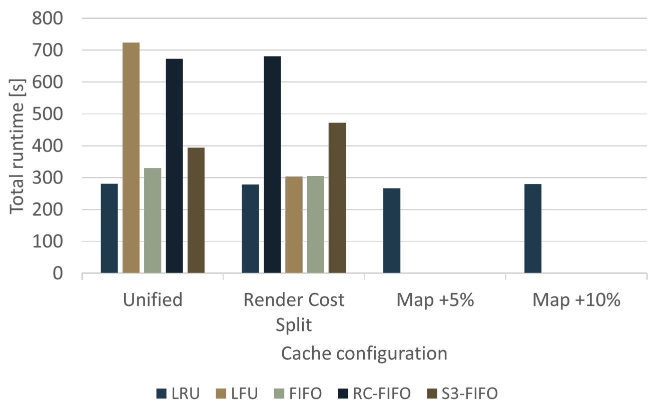

Fig. 5. Total Runtime for Unified and Partitioned Traditional Cache Policies

We recorded the time of arrival, time entering the cache, track, render time, and workload end time. This recorded sequence of operations is stored as a trace which we can then re-play in controlled conditions.

We subjected our traditional cache policies LRU, LFU, FIFO, and S3-FIFO, and our two render-cost aware policies, RC-FIFO and a cost-partitioned LRU, to the produced synthetic trace. Performance was measured by recording the start time for workload entries, the time spent within function calls to get / put in the cache, and the time to render a cache miss. The recorded metrics provide an end-to-end view of all cache operations summarised as total runtime and hit ratio(%).

Starting with hit ratio, Fig. 3 shows traditional caches alongside our RC-FIFO policy; the latter produces a large disparity in hit ratio across all tracks, as the prioritisation of resources to a single track starves other track operations. We see an improvement of 10% for map track's hit ratio vs. the next best traditional cache policy, with a 20% reduction in total hit rate. While the overall result here is undesirable, it demonstrates that we can successfully bias hit ratios towards particular media objects.

Building on this result we explore our render-cost-split cache, in which different volumes of cache space are reserved based on the cache-miss cost of different media objects to render. Those with higher costs are given more space in the cache, where cost was determined by the (expected client requests per track * (experience length/frame rate)). The caches and their hit ratios presented within Fig. 4 illustrate the decline in total hit ratio for each policy, however, an increase in hit ratio for a subset of tracks, within the presented results time takes 19ms per operation and the allocated resources have lowered the number of operations through reuse.

Our hit-ratio results only tell part of the story, however, since cache miss costs vary per-request. Fig. 5 illustrates the performance improvement gained through the partitioning of caches by per-media-object cost; here we see a reduction in total runtime of 2 seconds for our cache divided by bins for different render costs. This is an initial step towards a higher performance solution; a further 5% increment to the initial partition returns a 14 second reduction in total runtime. However, a further 5% step negates the performance gained, forming a search space for total runtime reduction through the adjustment of cache partitions.

B. BBC iPlayer: Real-world Trace

In this section, we present results using a sample from a real-world CDN trace of the BBC iPlayer streaming service, providing a more adversarial workload. In the previous section, while our different media objects have different render costs in the case of a cache miss, the popularity of those media objects is uniform. In practice popularity would be dynamic over time, depending on the behaviour of the current user population.

The unified cache presented within Fig. 6, highlights the need for throughput as FIFO presents the lowest total runtime of 943 seconds, as there are few operations when entering the cache, with S3-FIFO, RC-FIFO, and LRU incurring 6-12 second increases in total runtime, as the operations performed increase; further exacerbated by LFU reporting a 60 second increase in total runtime as hit ratio for LFU is half of all other policy implementations Fig. 7.

We present results for partitions formed through total cost as well as, velocity cost which forms partitions from the percentage of requests made from each device and not the time

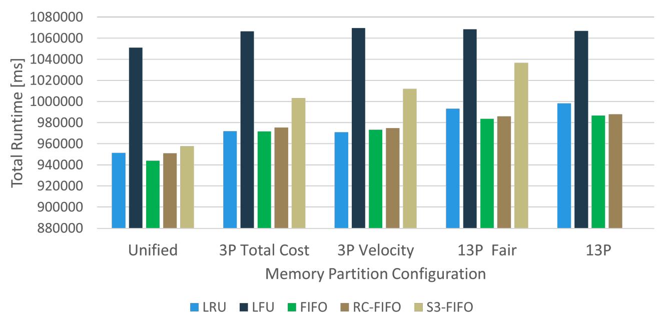

Fig. 6. Total runtime comparison for a unified, velocity/total cost three partitions (3P), and fair/total cost thirteen partitions (13P) cache configuration

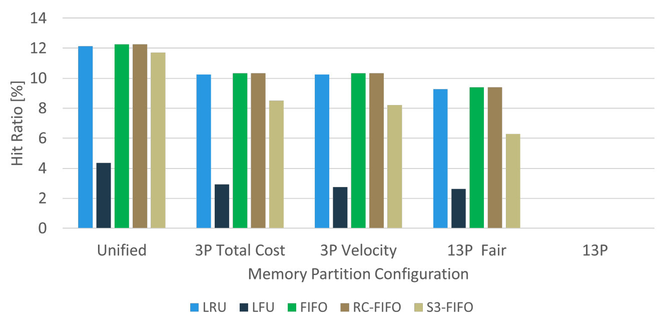

Fig. 7. Hit ratio for unified, velocity/total cost three partitions (3P), and total cost thirteen partitions (13P)

taken to perform a render operation on a cache miss. The first partition, denoted as 3P, encompasses only requests made from a TV, with 65257 requests and a render time / cache-miss-cost of 6ms. The second partition, encompasses Voice(11382), Mobile(11915), Phone(2848), and Tablet(838), for a total of 26,983 requests and a render time of 11ms. Finally, the third partition encompasses requests made from, Unknown (4463), Home Appliance(1), Smart Display(30), Desktop(8841), Set-top box(2756), and Streamer(3503) devices, for a total of 19594 requests with a render time of 19ms. We present the results for 3P within Fig. 6, where a increase in performance time occurs upon partitioning the cache by 20 seconds. We see a small reduction in total runtime for LRU and RC-FIFO of 1-2 seconds when adjusting partitions based on velocity, with all other policies presenting increased total runtime.

The second partition configuration, referenced as 13P, forms a configuration aiming to limit system performance as the previously mentioned 13 device types are each allocated resources dependent on their total/velocity cost. Fig. 6, 12P fair highlights the effect of over partitioning the cache while maintaining fairness for all tracks (resource allocation offers minimum resources). We see total runtime increases by 40-44 seconds for LRU, FIFO, RC-FIFO, and S3-FIFO as available resource result in further lowering hit ratio illustrated in Fig. 7. Furthermore, upon removing a fairness requirement to resource allocation, total runtime further increases and hit ratio lowers as tracks with insufficient resources are forced to render all requests.

In general these results highlight the adaptive systems challenge in configuring a split cache to match the deployment environment conditions: the number of bins, and the size of each bin in a cost-aware cache, must be tuned to match both the distribution of render-costs in the request pattern, and the relative popularity of items of those render-costs among the current user population. Achieving this balance may demonstrate the kind of gain in lowering compute time that we see in our synthetic trace results.

V. Discussion

Our results provide an initial exploration into determining cache memory allocation utilising per-track render cost, highlighting a workload with close temporal proximity between requests may offer an even further improvement to total runtime. In addition, runtime is separable from hit ratio as a recorded metric, as a lower hit ratio may still produce a lower overall total runtime as expensive operations are provided sufficient memory to increase a track's hit ratio.

While the partitioning of cache resources determined by render cost provided a reduction in total runtime when presented with our synthetic workload where requests were in close temporal proximity. The search for a configuration which provides an improvement for an adversarial workload, as presented with our BBC CDN trace is challenging. This is due to the CDN trace containing a greater number of elements with multiple accesses across a larger temporal range, limiting the efficiency of recency and frequency approaches. Furthermore, as these sparsely accessed entries limit the number of one-hit wonders, entries appearing only once, we see degrading performance in higher-throughput FIFO implementations. In future we also intend to experiment with a range of other cache size levels (in addition to the 2% tried here) to better understand how different cache sizes affect overall performance.

VI. Conclusion

We have introduced render-cost-driven caching for OBM as a novel application domain and presented the challenges for self-adaptive systems. We have also presented our initial exploration of the design space for render cost caching through a series of initial experiments around cache partitioning. Within future work, we intend to examine the applicability of previous theory from the self-adaptive community to further reduce total runtime through finding the near optimal partition count and size for the deployed environment in near/real-time as information on a cache's state expires in ms.

VII. Acknowledgement

This research was supported by UKRI EPSRC and BBC Prosperity Partnership *AI4ME: Future Personalised Object-Based Media Experiences Delivered at Scale Anywhere*, EP/V038087. Our current implementation of object-based media delivery is reliant on rendering libraries developed by BBC R&D, particularly in contributions by Mark Lomas, Rajiv Ramdhany, James Shephard, Sebastian Ward, Tim Pearce, and Michael Sparks.

References

[1] J. Yang, Y. Yue, and K. V. Rashmi, “A large-scale analysis of hundreds of in-memory key-value cache clusters at twitter,” *ACM Trans. Storage*, vol. 17, no. 3, aug 2021. [Online]. Available: https://doi.org/10.1145/3468521

[2] D. Zhuo, K. Zhang, Z. Li, S. Zhuang, S. Wang, A. Chen, and I. Stoica, “Rearchitecting in-memory object stores for low latency,” *Proc. VLDB Endow.*, vol. 15, no. 3, p. 555–568, nov 2021. [Online]. Available: https://doi.org/10.14778/3494124.3494138

[3] A. V. Nori, J. Gaur, S. Rai, S. Subramoney, and H. Wang, “Criticality aware tiered cache hierarchy: A fundamental relook at multi-level cache hierarchies,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 96–109.

[4] C. Pan, X. Wang, Y. Luo, and Z. Wang, “Penalty- and locality-aware memory allocation in redis using enhanced aet,” *ACM Trans. Storage*, vol. 17, no. 2, may 2021. [Online]. Available: https://doi.org/10.1145/3447573

[5] E. Friedlander and V. Aggarwal, “Generalization of lru cache replacement policy with applications to video streaming,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 3, aug 2019. [Online]. Available: https://doi.org/10.1145/3345022

[6] N. Race, D. Waddington, and W. Shepherd, “A dynamic ram cache for high quality distributed video,” in *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS '00)*, H. Scholten and M. Sinderen , Eds. Springer Verlag, Oct. 2000, pp. 26–39.

[7] A. Cidon, A. Eisenman, M. Alizadeh, and S. Katti, “Cliffhanger: Scaling performance cliffs in web memory caches,” in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'16. USA: USENIX Association, 2016, p. 379–392.

[8] G. Yan and J. Li, “Towards latency awareness for content delivery network caching,” in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 789–804. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/yan-gang

[9] A. Cidon, A. Eisenman, M. Alizadeh, and S. Katti, “Cliffhanger: Scaling performance cliffs in web memory caches,” in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'16. USA: USENIX Association, 2016, p. 379–392.

[10] S. A. Mirheidari, M. Golinelli, K. Onarlioglu, E. Kirda, and B. Crispo, “Web cache deception escalates!” in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 179–196. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/mirheidari

[11] M. Carrascosa and B. Bellalta, “Cloud-gaming: Analysis of google stadia traffic,” *Computer Communications*, vol. 188, pp. 99–116, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422000810

[12] A. Sabnis and R. K. Sitaraman, “Jedi: Model-driven trace generation for cache simulations,” in *Proceedings of the 22nd ACM Internet Measurement Conference*, ser. IMC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 679–693. [Online]. Available: https://doi.org/10.1145/3517745.3561466

[13] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy, “Measurement and analysis of a streaming media workload,” in *3rd USENIX Symposium on Internet Technologies and Systems (USITS 01)*. San Francisco, CA: USENIX Association, Mar. 2001. [Online]. Available: https://www.usenix.org/conference/usits-01/measurement-and-analysis-streaming-media-workload

[14] B. Porter, M. Grieves, R. R. Filho, and D. Leslie, “REX: A development platform and online learning approach for runtime emergent software systems,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 333–348. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/porter

[15] A. Wang, J. Zhang, X. Ma, A. Anwar, L. Rupprecht, D. Skourtis, V. Tarasov, F. Yan, and Y. Cheng, “Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache,” in *Proceedings of the 18th USENIX Conference on File and Storage Technologies*, ser. FAST'20. USA: USENIX Association, 2020, p. 267–282.

[16] M. T. Islam, R. Borovica-Gajic, and S. Karunasekera, “A multi-level caching architecture for stateful stream computation,” in *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 67–78. [Online]. Available: https://doi.org/10.1145/3524860.3539803

[17] D. S. Berger, N. Beckmann, and M. Harchol-Balter, “Practical bounds on optimal caching with variable object sizes,” *SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 1, p. 24–26, jun 2018. [Online]. Available: https://doi.org/10.1145/3292040.3219627

[18] M. Abdi, A. Mosayyebzadeh, M. H. Hajkazemi, E. U. Kaynar, A. Turk, L. Rudolph, O. Krieger, and P. Desnoyers, “A community cache with complete information,” in *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, Feb. 2021, pp. 323–340. [Online]. Available: https://www.usenix.org/conference/fast21/presentation/abdi

[19] Z. Song, D. S. Berger, K. Li, and W. Lloyd, “Learning relaxed belady for content distribution network caching,” in *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'20. USA: USENIX Association, 2020, p. 529–544.

[20] B. Porter, P. Dean, N. Race, M. Lomas, and R. Ramdhany, “Self-adaptive systems challenges in delivering object-based media,” in *2024 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2024.

[21] B. Porter, P. Faulkner Rainford, and R. Rodrigues-Filho, “Self-designing software,” *Commun. ACM*, vol. 68, no. 1, p. 50–59, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3678165

[22] J. Yang, Y. Zhang, Z. Qiu, Y. Yue, and R. Vinayak, “Fifo queues are all you need for cache eviction,” in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 130–149. [Online]. Available: https://doi.org/10.1145/3600006.3613147