No Clash on Cache: Observations from a Multi-tenant Ecommerce Platform

Anna Lira
anna.lira@lsd.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil

Ruan Alves
ruan.alves@lsd.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil

Thiago Emmanuel Pereira
temmanuel@computacao.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil

Fabio Morais
fabio@computacao.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil

João Ramalho
joao.ramalho@lsd.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil

Mariana Mendes
mariana.mendes@vtex.com
VTEX
Rio de Janeiro, Rio de Janeiro, Brazil

ABSTRACT

Caching is a classic technique for improving system performance by reducing client-perceived latency and server load. However, cache management still needs to be improved and is even more difficult in multi-tenant systems. To shed light on these problems and discuss possible solutions, we performed a workload characterization of a multi-tenant cache operated by a large ecommerce platform. In this platform, each one of thousands of tenants operates independently. We found that the workload patterns of the tenants could be very different. Also, the characteristics of the tenants change over time. Based on these findings, we highlight strategies to improve the management of multi-tenant cache systems.

CCS CONCEPTS

• **Computer systems organization** → *Cloud computing*; • **Theory of computation** → **Caching and paging algorithms**; • **General and reference** → **Performance**.

KEYWORDS

Web, Cache, Workload, Multi-Tenant, Characterization

ACM Reference Format:
Anna Lira, Ruan Alves, Thiago Emmanuel Pereira, Fabio Morais, João Ramalho, and Mariana Mendes. 2024. No Clash on Cache: Observations from a Multi-tenant Ecommerce Platform. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3629526.3645039

1 INTRODUCTION

Caching is a classic that never dies. From the bottom layers of hardware CPU caches to the upper layers of web caches of cloud applications, caching reduces client-perceived latency and service load.

To get the most out of a cache, one needs to understand the characteristics of the workload submitted to the cache and configure it accordingly. Failing to match parameters such as cache capacity and eviction algorithms to the workload leads to direct impacts on the quality of service (usually observed in miss/hit ratio indicators) or resource waste (i.e., when the cache capacity is over-provisioned and additional capacity does not improve performance).

As an example, when configuring a web cache system, a typical starting point is to estimate the load level. The number of servers used in a large cache service depends on this load-level information since cache systems degrade when overloaded. Despite being useful, one needs more than load levels to define other cache parameters. For example, to determine cache capacity, it is necessary to consider the popularity of cached items. Item popularity matters because many requests sent to a cache might be related to a small number of cached items, thus reducing the need for more cache capacity. Also, caches typically show temporal locality [2]. Consequently, the cache needs to retain only the current working set.

Practitioners are well aware of the importance of considering these advises. However, one factor deviates practice from good practice: multi-tenancy.

To illustrate, consider multi-tenant ecommerce platforms, the case study of this paper. In these platforms, each tenant is an enterprise independent from the others; each tenant has its clients and products. There must be tenants that sell more than others. There must be tenants with large and small product inventories. There must be tenants with seasonal and sporadic selling patterns. However independent, the tenant's ecommerce sites run on shared resources and services (including caches) owned and managed by the platform. Considering that cache services are sensitive to load characteristics, as we described, and tenants are unlikely to have the same workload, what are the challenges a multi-tenant cache operator has to deal with?

To uncover these factors and highlight the challenges of multi-tenant caching, we collected and analyzed a trace of one of the cache services from VTEX, a large-scale ecommmerce platform[1].

[1]https://vtex.com

The observed cache service supports a few thousand tenants for a 10 hours observation period.

We observed that the cache load varies up to three times, reaching hundreds of thousands of requests per minute. Also, the load follows the same overall platform traffic trend (high traffic until late at night and lower traffic at dawn). However, we have a completely different picture when we analyze tenants in isolation. Some tenants exhibit stable load, others show a periodic load pattern, while some show peak periods (even at unexpected hours). Also, the load is highly concentrated among tenants: the 10% most loaded tenants account for almost 80% of the aggregated load.

The item popularity is also concentrated. Around 10% of items received about 50% of all requests. While there is a subset of very popular items, there are many items on the opposite extreme; 69% of the items are requested only one time (also known as one-hit wonders).

We also analyzed the degree of repeated access to the same items for each tenant. A high repetition degree indicates that the load is cache-friendly. This is because, the higher the repetition, the higher the chances of cache hits. Half of the tenants show less than 40% of repetition; a direct implication is that these tenants cannot have more than the 40% hit ratio, regardless of the cache configuration and capacity. Repetition also varies over time. While some tenants sustain low or high repetition during the observed period, others change their behavior as time passes.

The remainder of this paper is organized as follows. Section 2 describes the procedure for collecting data from the ecommerce platform in production. Section 3 describes the context and metrics applied in the workload characterization presented in Section 4. In addition, Section 5 describes the implications of the observed workload characteristics for cache management. Section 6 presents this research's findings and future work.

2 SYSTEM OVERVIEW AND INSTRUMENTATION

This section provides an overview of the Catalog system, the component of the platform from which we collected cache information (Section 2.1). We also describe the instrumentation procedure for data collection on production servers (Section 2.2). Finally, we explain the data used in the analysis and describe how data normalization is performed for anonymization purposes (Section 2.3).

2.1 Catalog Service

We collected data from a global business-to-consumer (B2C) and business-to-business (B2B) ecommerce provider. This provider offers tools and services to support ecommerce companies in creating and operating their online stores. This study focused on the Catalog system. The Catalog system is responsible for managing non-ephemeral product data (e.g., titles, descriptions, product identifiers, and enterprise identifiers) for all the tenants in the platform using a cache layer to improve their Quality of Service (QoS).

The Catalog system is composed of three layers: i) the database layer is responsible for the persistence of data and works as an index for the product information; ii) the logic and load balancing layer, responsible for providing access to product information and load balancing of requests; and iii) the cache layer is the service that temporarily stores product information for performance improvement. Figure 1 shows an overview of the system with the three layers and their relationships.

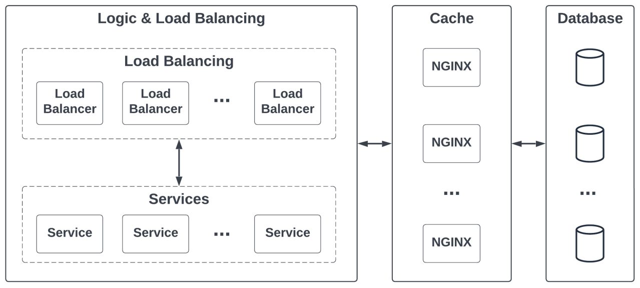

Figure 1: Overview of the architecture of ecommerce platform Catalog system. The cache layer stores database responses. The logic layer requests data from the database, using the cache layer as a proxy. In case of a MISS, requests are forwarded to the databases, and responses are cached. In the case of a HIT, a cached response is directly returned from the cache layer.

The Catalog services access the production information stored on the database through HTTP using a key-value format to structure the data. A key is generated by hashing the URI that uniquely identifies the request made to the database, and the value is the descriptive information for the related product. A cache layer, implemented as a cluster of NGINX [1] servers, is a proxy of the database to the service layer. The cache layer returns product information already cached by previous requests, thus reducing the load on the database system and reducing the latency perceived by the clients.

As the ecommerce system provides Catalog services for different online stores, the cache layer is shared by different tenants. Each request for product information is related to a specific product of a specific tenant in the same cache system. Thus, the cache stores data from products of all tenants' catalogs.

2.2 Data Collection

During our observation period, we activated a more verbose NGINX logging mode to collect the data. The Catalog operates within a high-traffic production environment, where misguided data instrumentation may cause disruption or system performance degradation. Therefore, we collected a 10% random sample of all requests in 5-minute intervals to avoid affecting the reliability of the Catalog. An agent running on the cache service continuously sends the log files to another server, to avoid fulfilling the NGINX servers' disks. The observation period ranged from 21:00 to 7:00 GMT-3, on December 13 and 14, 2022. After that, we stopped the observation and resumed the NGINX logging level to its default value.

To support our analysis, the data we collected includes the information associated with the products requested to the Catalog, cache status, and timing metadata. In summary, we considered the following information for each observed request:

- The request URI. Product and tenant identifications are encoded as parameters in the URI;

- Cache response status, which indicates when the cache answered a request as a MISS or HIT (and other NGINX possible statuses);
- The timestamp the request arrived at the cache layer;
- The identification of the NGINX node that received the requests;
- The request-response wait time.

2.3 About the Data

The collected workload has up to tens of millions of requests. These requests refer, approximately, to another tens of millions of products from approximately a few thousand online stores. Notably, each tenant exhibits unique and specific access patterns, which Section 4 will explore further.

To protect business-sensitive information, we anonymized the dataset. All the data discussed in the following sections was anonymized by applying the min-max normalization function. All the identifiers were anonymized using a hash function. The data normalization allows temporal comparison for all aggregated tenants, for one tenant, as much as comparisons among tenants. For example, Figure 2 presents the normalized number of incoming requests received per minute throughout the data collection period, illustrating the overall variations and trends in request rates.

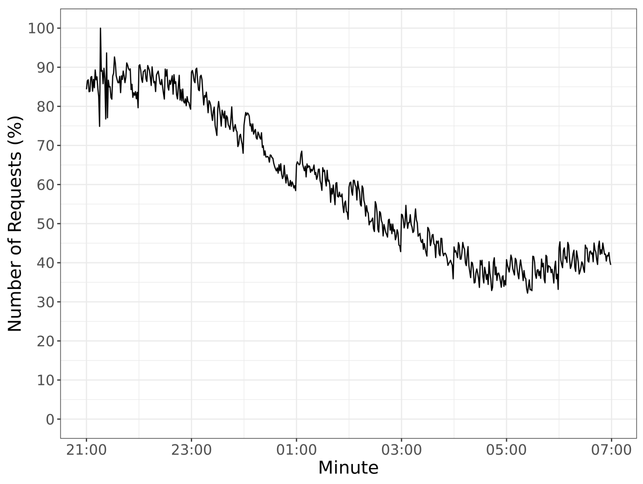

Figure 2: Normalized incoming request rates per minute for all aggregated tenants. It highlights variations in the request rate per minute and trends over the data collection period where peak demand occurs at night, with a typical decline in the dawn, followed by a resurgence of demand in the following hours of the morning.

Furthermore, for this work, we have focused on three key data fields crucial for understanding a multi-tenant cache system. These fields include request timestamp, tenant identifier, and product (or item) identifier.

3 BACKGROUND

In this section, we define the metrics we adopted to our characterization analyzed in Section 4. While these metrics were already described in the literature, some of them were not formally defined.

To evaluate the overall cache efficiency, we used the Hit Ratio. This metric is the proportion of data requests that were successfully met from the cache (referred to as Hit Status) relative to the total number of requests made. Consider two multisets: R, containing all data requests made during a specific period, and H, containing only those requests fulfilled by the cache within the same period ($H \subseteq R$). The Hit Ratio can be calculated using the following relation:

$$\text{Hit Ratio} = \frac{|H|}{|R|} \tag{1}$$

For example, consider a cache with a capacity of two items and a sequence of requests for items, R = {"A": "MISS", "B": "MISS", "A": "HIT", "A": "HIT", "C": "MISS", "B": "MISS", "A": "MISS"}, and a few hits, represented by H = {"A", "A"}. To calculate the hit ratio, we divide the number of hits by the total number of requests made, which in this case is 2/7 or 0.29. This means that 29% of the requests were successfully satisfied by the cache during the given time period.

To understand the demand, in addition to the number of requests submitted to the cache, we considered the **Footprint**. The Footprint metric measures the amount of data accessed within a specified time window by quantifying the number of distinct requested items in a given period [18]. This metric is essential for analyzing system capacity utilization, as it allows us to see how many items are requested in a given period. It also shares a close relationship with the Working Set Size (WSS) theory, which helps to understand an application's memory requirements. Considering the sequence of requests R and the set F (a subset of R) of distinct items requested, the Footprint is defined as follows:

$$\text{Footprint} = |F| \tag{2}$$

The **Item Repetition Ratio (IRR)** metrics, in its turn, gives an indication on how a workload would take advantage of the cache [9]. The IRR is an upper bound for the cache hit ratio for a determined workload. Given the multiset P of repeated requests for items in the period, the IRR is defined as follows:

$$IRR = \frac{|P|}{|R|} \tag{3}$$

For the previous example, P = {"A", "A", "B", "A"} and the IRR is 4/7 or 57.14%.

We also considered **Temporal Locality** in our characterization. Temporal locality refers to the tendency of the same item to be referenced within short intervals. It differs from concentration, which refers to the aggregate reference counts for items, regardless of the referencing order. Some metrics can be used to measure temporal locality, for example, the LRU stack-depth [12] and Inter-Reference Time [14]. In this work, we will focus on Inter-Reference

Table 1: Normalized statistical measures of the number of requests per minute. These statistics represent a substantial difference between the mean and median to the maximum value, suggesting a significant difference for the peaks.

Mean	Median	Max	Min	Std. Dev
0.41	0.37	1	0	0.20

Time [2]. This metric represents the time between references to the same item. We calculated the mean of the Inter-Reference Time (MIRT). MIRT sets itself apart from LRU stack-depth analysis by focusing on the timing of data access rather than the order within a stack. While LRU stack-depth provides insight into the sequence in which a specific number of requests occurred within a particular interval, MIRT's primary objective is to define and analyze these time intervals.

In addition to the metrics defined above, in the discussion presented in section 5, we adopted LRU as eviction policy. There is a huge literature on eviction policies trade-offs and there are many policies that can be better than LRU [7, 10, 11, 19, 20]. Notwithstanding, we adopted LRU because: 1) it was the policy used by the cache we observed (NGINX); and 2) some analytical methods (as the one used in section 5) are based on LRU.

4 WORKLOAD CHARACTERIZATION

This section discusses the characteristics of the load submitted to the cache of the Catalog system we observed. Section 4.1 focuses on observing the demand for requests the caching system receives. Next, section 4.2 explores the irregular popularity of the items stored in the system, identifying "hot" and infrequently accessed items. Section 4.3 highlights Footprint results, while section 4.4 presents IRR. Finally, section 4.5 evaluates the temporal locality of the data by analyzing the Inter-Reference Time.

4.1 Request Load

The initial analysis centers on the volume of requests entering the system over ten hours, as detailed in Table 1. The table provides normalized statistical metrics derived from the load. The findings reveal a mean of 0.41 and a median of 0.37, indicating that both metrics deviate significantly from the maximum value. This suggests a distribution pattern wherein a substantial proportion of observations exhibit relatively low activity levels compared to the peak request rate.

Regarding the temporal pattern of the request arrival, Figure 2 shows the number of requests per minute over time. The peak of requests occurs in the evening — after working hours — when it is common for demand for web services to be high. While there is a tendency for demand to fall at daybreak, demand for services rises again in the early hours of the morning [17].

The load is very concentrated in a small group of tenants: only 10% of tenants account for approximately 80% of all recorded requests. Figure 3 illustrates the cumulative request count for each tenant, ranked accordingly.

Tenants with a high number of requests can eventually worsen the performance of tenants with a low number of requests. This interference could happen when a burst of requests for new data from tenants evicts other tenants' popular data, so future requests that can reuse this data will result in cache misses. Thus, tenants with a high number of requests can quickly evict items and occupy a big slice of the cache. This highlights the importance of the promotion of fairness and mitigation of interference between tenants in shared caches [3, 6, 13, 16].

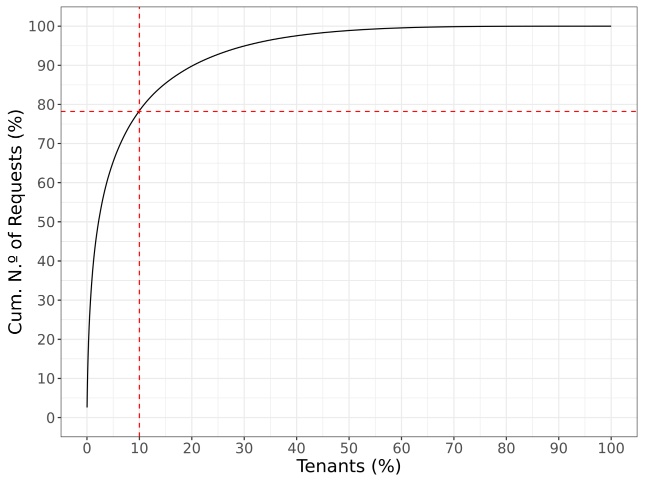

Figure 3: The cumulative sum of received requests, ranked by descending number of requests by a tenant. A small number of tenants are responsible for a large number of requests.

The request load for each tenant also shows temporal variability. This introduces an extra layer of complexity in resource allocation. Figure 4 illustrates the variation in the number of requests over time for a sample of the biggest tenants. Some tenants exhibit peaks in requests, indicating periods of significantly higher demand than their usual levels. This may result from seasonal events, special promotions, or product launches. Conversely, some tenants maintain relatively constant activity over time, with no significant fluctuations in request quantity. This suggests a more stable and predictable traffic profile associated with a regular customer base. Furthermore, some tenants may display intermittent request patterns, alternating between intense activity periods and relatively calm moments. This oscillation may be influenced by peak shopping times or specific marketing actions.

These behaviors may indicate that system management could be more dynamic and sensitive to tenants' needs since the traditional static resource allocation models can lead to provisioning problems. A dynamic resource adjusts resource allocation based on the current workload, trying to ensure that resources are neither wasted nor insufficient. In this scenario, tenants with request peaks may require more substantial cache allocations during these periods of high demand. Conversely, tenants with stable request patterns may benefit from conservative allocations.

In addition to the dynamic allocation of resources, the presence of tenants with high demand at certain times also reinforces the adoption of strategies to promote fairness and avoid inter-tenant interference. This can reduce the effects caused by higher demand in lower-demand customers.

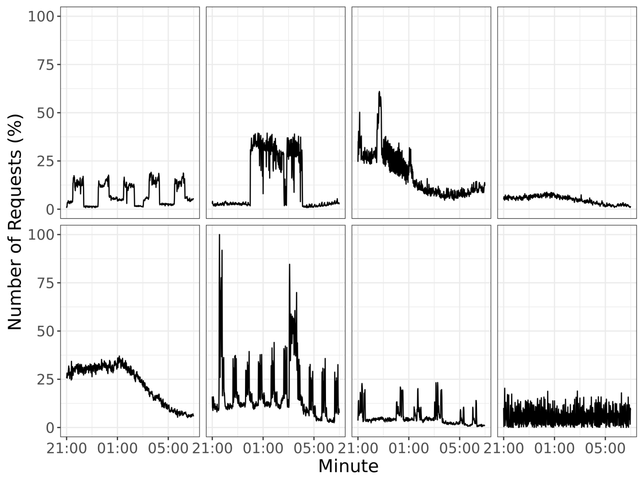

Figure 4: Request number variation over time for a sample of the biggest tenants. Different access patterns emerge, with some tenants experiencing request peaks, others displaying fluctuations, and some exhibiting intermittent patterns.

Table 2: Frequency type distribution. There is a huge fraction of the items that is requests only once.

Item Frequency Type	Percentage
One Request	69%
Two Requests	13%
More Than Two Requests	18%

4.2 Concentration Of Access

Looking for requested items and evaluating the frequency of access, we can see a discrepancy in popularity. Some can be classified as "hot" items with a high access volume. These items are characterized by their great popularity and constant demand. In contrast, other items are rarely accessed and, in some cases, not accessed.

Figure 5 shows the non-uniform pattern in item referencing behavior. Around 50% of all requests are directed at only 10% of the distinct items in the system. This phenomenon suggests the existence of a subset of items that are highly requested, while other items show considerably less repetition. Braun and Claffy describe this concentration phenomenon as a common characteristic of Web traffic [4].

Also, as observed in Table 2, the portion of items that are accessed only once (one-hit wonders) is very large. Items with no repeated access could not take advantage of the cache. However, simply ignoring an item in its first request can be a wrong choice because this item can be re-accessed in the near future, not being a one-hit wonder. Furthermore, the same table shows the percentage of two-hit wonders, which must be understood if is a good choice to store items like them too, since they will be reused only once. Therefore, it is necessary to seek strategies that reduce the number of n-hit wonders according to the particular cache objectives. Many policies seek how to avoid storing unpopular items [5, 7, 8, 10, 20].

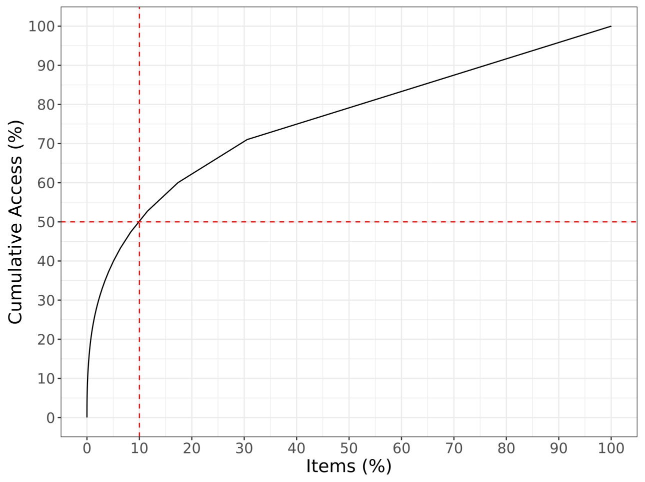

Figure 5: The cumulative sum of accesses per item, ranked by decreasing the number of accesses. The curve experiences rapid growth for a set of items, commonly called "hot" items. A small number of items are accessed more frequently, while the remaining are accessed sporadically. This pattern suggests distinct popularity dynamics among the items.

4.3 Footprint

As mentioned in previous sections, the Footprint value depends on the size of the window range chosen. For our trace, with a time window of 10 hours, the Footprint is half the total of requests made (tens of millions of requests). When we look at the Footprint per tenant, we see a discrepancy in the distribution of Footprint values. Figure 6 illustrates the distribution of Footprints among tenants, highlighting a concentration of Footprint for a small portion of tenants. In summary, the 10% of tenants represent a portion of around 75% of the total Footprint, suggesting that these entities may have distinct usage patterns, potentially utilizing the system more intensively or requiring more resources than the remaining majority. On the other hand, the remaining 90% of tenants contribute only a quarter of the total Footprint.

Additionally, the tenant's Footprint varies over time, influenced by their different workload patterns, such as changes in the number of orders and access. As mentioned, variations in the observed time window can lead to changes in the Footprint value. Figure 7 further illustrates this dynamic, showing the variations in Footprint for the six largest tenants. This emphasizes that a tenant's Footprint can change over time. Observing the Footprint in time windows can help us understand the storage space requirements of each tenant in the system over a given period.

4.4 Item Repetition Ratio

The data, with a time window of 10 hours, indicates variation in IRR values among different tenants, suggesting that some tenants are more adept at utilizing caching benefits, while others may not exhibit behaviors favorable to caching. Figure 8 shows many tenants have low IRR. For example, around 50% of tenants have IRR less than 40%. With low repetition, these tenants could be candidates for

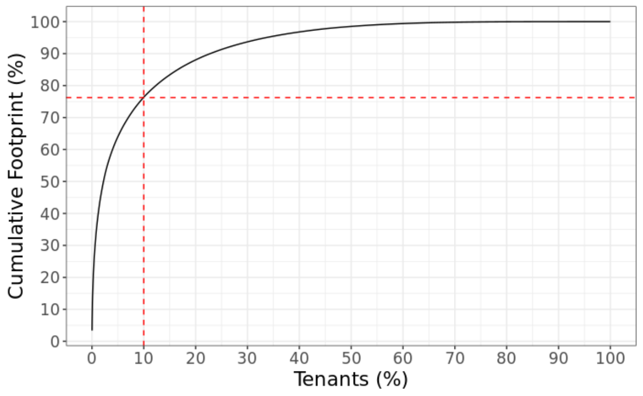

Figure 6: Cumulative distribution of Footprint. This illustrates the concentration of Footprint in a small portion of tenants, around 75% of the total Footprint corresponds to 10% of the tenants, so a small number of tenants occupy a large slice of cached storage.

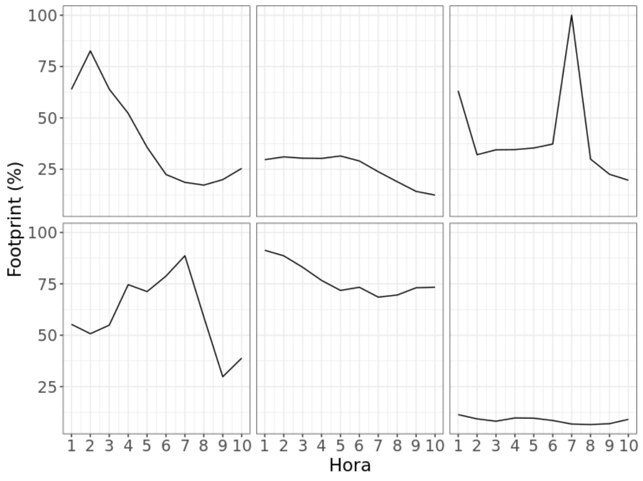

Figure 7: Footprint over time for a selected sample of the biggest tenants by number of requests. This highlights the variability in a tenant's usage of resources throughout different periods of the day. Some tenants exhibit peaks and valleys over time, while others consistently maintain either a high or low usage Footprint.

not having items in the cache since their requests in a time window are for different items, not contributing to the cache hit.

By decreasing the window size for adjacent one-hour windows, we observe some IRR behaviors over time, as shown in Figure 9. Some tenants show high IRR with slight variation, others low IRR with little variation, and others show considerable variation in IRR between windows. In addition to the tenants with a slightly variable IRR, we also observed some that showed a large temporal oscillation. This dynamic behavior emphasizes the importance of evaluating the adequacy of the cache at specific time intervals. This approach makes it possible to discern tenants whose caching behavior may vary over time, thus requiring adaptive cache management strategies for optimal performance. In this case, the system should ideally have a mechanism for calculating IRR periodically that plays a central role in fine-tuning cache capacity by selecting which tenant should or should not be in the cache at a given time.

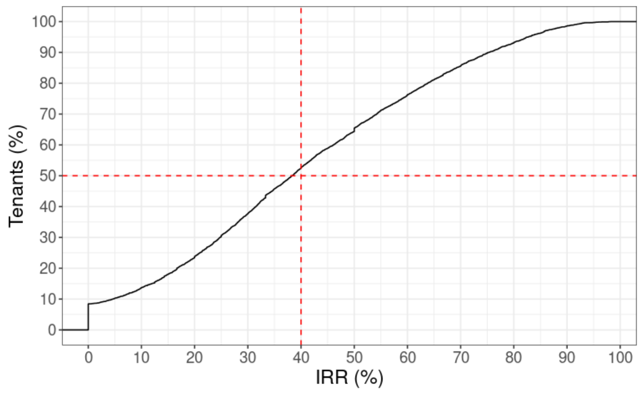

Figure 8: Cumulative distribution of Item Repetition Ratio. Some tenants are better at reusing the cached data, taking advantage of the cache usage. While others may not demonstrate cache-friendly behavior, occupying memory and worsening performance.

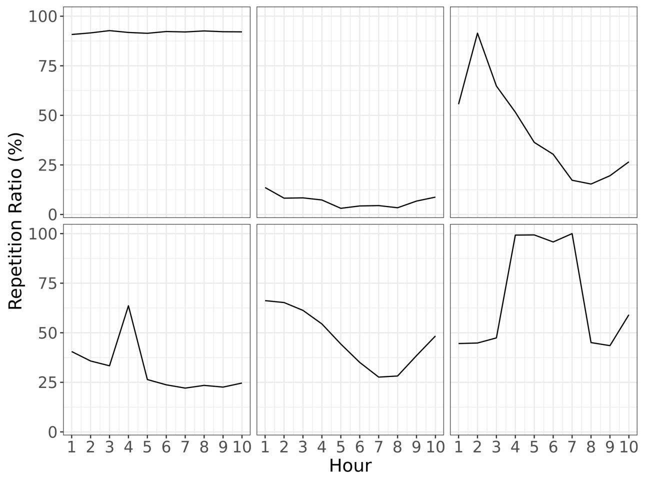

Figure 9: IRR over time for a selected sample of the biggest tenants by number of requests. This illustrates diverse patterns and variations in a tenant's cache-friendliness.

Before entering the cache system, analyzing the tenants' IRR could be a strategy for allowing them in or not, section 5 will describe this approach. For example, if the tenant's IRR is high and has little variation, it is a good candidate for cache entry. On the other hand, if the IRR is low and with little variation, it is a candidate for not entering the cache at any time.

Table 3: MIRT statistical measures for the items.

Mean	Median	Max	Min
0.26	0.19	1	0

4.5 Temporal Locality

The Inter-Reference Time can help us to understand the tendency of reference to the items. Figure 10 exhibits the MIRT of the items in the workload. As explained in Section 3, the MIRT is calculated by finding the mean of the time intervals between references to the same item. It is worth noting that items referenced only once lack Inter-Reference Time values, making it impossible to calculate the mean.

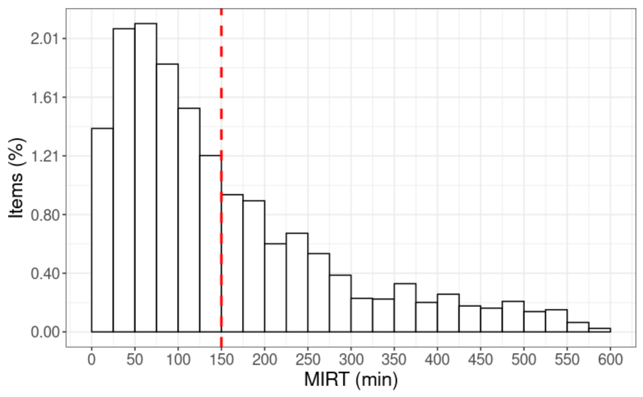

Figure 10: Percentage of items by Mean Inter-Reference Time ignoring items referenced only once. Most have MIRT lower or equal to 150 minutes.

A significant portion of items have a MIRT lower or equal to 150 minutes, specifically 57.20% of them (including items referenced once). Table 3 also reinforces this by showing that the measures of central tendency (mean and median) are low. This is an indication of good cache usage by the items. On the other hand, 42.80% of the items have MIRT longer than 150 minutes. This indicates that many items are idle during specific periods, taking up space in the cache.

Figure 11 shows the dynamic trend in item access patterns for ten adjacent one-hour time windows, where each box corresponds to an analyzed hour. The frequency of access to items changes over different periods, indicating a dynamicity in the access patterns over time. The first few hours have more items with low MIRT, where the number of access is high, and the average number of hits is concentrated mainly between 10 and 15 minutes. After that, the number of accesses decreases and becomes increasingly dispersed, with a reduction in items with low MIRT.

In general, we observe that the pattern of temporal locality changes over time. This shows that tenants' and items' cache requests are variable over time. This further reinforces the need for dynamic management of the cache.

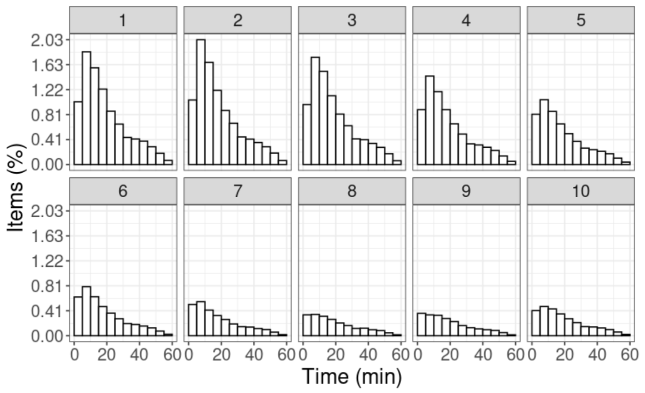

Figure 11: Percentage of items by Mean Inter-Reference Time for each one-hour window of items that are referenced more than once. Showing the dynamic tendency in item access patterns.

5 PERFORMANCE IMPLICATIONS AND ISSUES

In this section, we discuss the possible consequences of the characteristics observed in our multi-tenant trace. In particular, we focus on adapting cache management strategies to be aware of the tenants.

Unpopular items: A Case for Exclusion from Cache. Items with low repetition may not justify their inclusion in the cache. Allocating cache resources to items with sporadic access may lead to sub-optimal resource utilization. In this case, cache policies and systems that consider admission control like Lazy Adaptive CacheReplacement [10], CacheSack [21], and others [5, 7, 8, 11, 20].

One interesting possibility to explore is to consider data about the tenants as hints to the admission control algorithm. For example, would be useful to have more aggressive admission control policies for tenants that have a low item popularity profile?

Item Repetition Ratio as a Filter for Tenants. As section 4 indicated, some tenants are not cache-friendly (because they have low IRR). In addition to the **item** admission control, one can consider a **tenant** admission control (using IRR as an indicator). To provide a short overview of this idea, we evaluated the impact of removing some tenants from our multi-tenant trace. We used a trace-drive simulation that gives the overall hit ratio based on a given trace capacity. Considering an optimal capacity to attend all the tenants found on the trace as the baseline, we simulated scenarios removing the tenants that have IRR below three different thresholds (20%, 30%, and 50%). As Figure 12 shows, when tenants with 20% or fewer IRR values were excluded, the optimal cache capacity was reduced to 90% of the original capacity, resulting in an optimal hit ratio of 61%. Similarly, imposing a stricter criterion of 30% IRR led to an optimal capacity of 78% of the original, leading to a higher hit rate of 64%. Excluding tenants with IRR values of 50% or less further improved the efficiency of the cache, producing an optimal capacity of 49% of the original capacity and a commendable hit rate of 71%. Obviously, other trade-offs need to be analyzed, including the corresponding impact on the latency and server load caused by the removal of tenants. In any case, the impact on resource usage could be important, as indicated by this analysis.

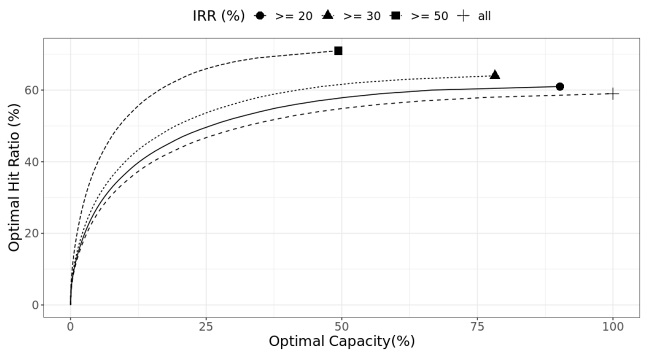

Figure 12: Optimal hit ratio based on optimal capacity for each one of workloads. Each line is one type of workload (i.e. the line IRR >= 20 means that only tenants with IRR above or equal to 20% are on the workload). This suggests that some tenants with low IRR can be removed to reduce the capacity while maintaining the same level of performance or increasing it.

Cache partition. As mentioned in our characterization, a few big tenants concentrate the most on cache usage. We also show that the load of the tenants varies over time. It is not clear the impact a spike in the load of a big tenant can cause on the QoS of the other tenants (in particular, if the capacity of the cache is not too large). One possible to avoid the interference caused by tenants sharing the same cache is to create and enforce partitions on the shared cache to host the tenants [3, 6, 13, 15, 16].

6 CONCLUSIONS

This paper has provided an analysis of the workload characteristics in a multi-tenant cache environment using data from a prominent web cache service in a large-scale ecommerce platform. The findings revealed a significant imbalance in request distribution among tenants, with approximately 80% of requests directed towards 10% of them. Moreover, we observed distinct access behavior patterns, ranging from steady loads to periodic spikes in request volumes.

Furthermore, the study highlighted that many requests concentrated on a small number of items (hot items). On the other hand, a majority part of the items experienced sporadic access. Additionally, it was evident that some tenants demonstrated low cache-friendliness, resulting in a cache hit ratio decrease due to infrequent item retrievals.

The cache-friendliness is mainly related to tenants' request patterns and product inventory. Significant request patterns were identified in tenants' workloads: Some tenants are more searched than others, and this implies a more significant number of requests to the system; can present seasonal request patterns, others sporadic patterns; can present peaks and/or valleys in request patterns, or can present stable patterns over time. In addition, there are tenants with an extensive number of products, which can mean that they need more storage space than others. In addition, we found that access patterns vary over time, emphasizing the dynamic nature of cache requests. This underlines the importance of dynamic cache management to adapt to changing access patterns and optimize the use of resources.

As the next steps in this industry collaboration, we plan to investigate how can we take into consideration tenants' profiles into cache management policies (admission control, eviction, and partition). Also, it is necessary to consider the mechanisms to adapt to changes in the behavior of the tenants. Although some of these issues are discussed in the literature, in particular for hyperscalers [21], there are still open questions on how to support regular web caches.

7 ACKNOWLEDGMENTS

This work has been funded by MCTIC/CNPq-FAPESQ/PB (EDITAL Nº 010/2021) and by VTEX BRASIL (EMBRAPII PCEE1911.0140).

REFERENCES

[1] Nginx documentation. https://nginx.org/en/docs/. Accessed: 2023-11-05.

[2] Arlitt, M. F., and Williamson, C. L. Internet web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw. 5*, 5 (1997), 631–645.

[3] Berger, D. S., Berg, B., Zhu, T., Sen, S., and Harchol-Balter, M. Robinhood: Tail latency aware caching - dynamic reallocation from cache-rich to cache-poor. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018* (2018), A. C. Arpaci-Dusseau and G. Voelker, Eds., USENIX Association, pp. 195–212.

[4] Braun, H.-W., and Claffy, K. C. Web traffic characterization: an assessment of the impact of caching documents from ncsa's web server. *Computer Networks and ISDN systems 28*, 1-2 (1995), 37–51.

[5] Chai, Y., Du, Z., Qin, X., and Bader, D. A. WEC: improving durability of SSD cache drives by caching write-efficient data. *IEEE Trans. Computers 64*, 11 (2015), 3304–3316.

[6] Cidon, A., Rushton, D., Rumble, S. M., and Stutsman, R. Memshare: a dynamic multi-tenant key-value cache. In *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017* (2017), D. D. Silva and B. Ford, Eds., USENIX Association, pp. 321–334.

[7] Einziger, G., Friedman, R., and Manes, B. Tinylfu: A highly efficient cache admission policy. *ACM Trans. Storage 13*, 4 (2017), 35:1–35:31.

[8] Eisenman, A., Cidon, A., Pergament, E., Haimovich, O., Stutsman, R., Alizadeh, M., and Katti, S. Flashield: a hybrid key-value cache that controls flash write amplification. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019* (2019), J. R. Lorch and M. Yu, Eds., USENIX Association, pp. 65–78.

[9] Gu, R., Li, S., Dai, H., Wang, H., Luo, Y., Fan, B., Basat, R. B., Wang, K., Song, Z., Chen, S., Wang, B., Huang, Y., and Chen, G. Adaptive online cache capacity optimization via lightweight working set size estimation at scale. In *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023* (2023), J. Lawall and D. Williams, Eds., USENIX Association, pp. 467–484.

[10] Huang, S., Wei, Q., Feng, D., Chen, J., and Chen, C. Improving flash-based disk cache with lazy adaptive replacement. *ACM Trans. Storage 12*, 2 (2016), 8:1–8:24.

[11] Megiddo, N., and Modha, D. S. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the FAST '03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA* (2003), J. Chase, Ed., USENIX.

[12] Ponciano, L., Andrade, N., and Brasileiro, F. V. Bittorrent traffic from a caching perspective. *J. Braz. Comput. Soc. 19*, 4 (2013), 475–491.

[13] Pu, Q., Li, H., Zaharia, M., Ghodsi, A., and Stoica, I. Fairride: Near-optimal, fair cache sharing. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016* (2016), K. J. Argyraki and R. Isaacs, Eds., USENIX Association, pp. 393–406.

[14] Sediyono, A. Dynamic average of inter-reference time as a metric of web cache replacement policy. In *Proceeding of International Conference on Rural Information and Communication Technology* (2009).

[15] Seyri, A., Pan, A., and Vamanan, B. Dynamically sharing memory between memcached tenants using tingo. In *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2019, Companion Volume, Orlando, FL, USA, December 9-12, 2019* (2019), ACM, pp. 40–42.

[16] Suh, G. E., Rudolph, L., and Devadas, S. Dynamic partitioning of shared cache memory. *J. Supercomput. 28*, 1 (2004), 7–26.

[17] Vallamsetty, U., Kant, K., and Mohapatra, P. Characterization of e-commerce traffic. *Electronic Commerce Research 3*, 1 (2003), 167–192.

[18] Xiang, X., Ding, C., Luo, H., and Bao, B. HOTL: a higher order theory of locality. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013, Houston, TX, USA, March 16-20, 2013* (2013), V. Sarkar and R. Bodík, Eds., ACM, pp. 343–356.

[19] Yang, J., Qiu, Z., Zhang, Y., Yue, Y., and Rashmi, K. V. FIFO can be better than LRU: the power of lazy promotion and quick demotion. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems, HOTOS 2023, Providence, RI, USA, June 22-24, 2023* (2023), M. Schwarzkopf, A. Baumann, and N. Crooks, Eds., ACM, pp. 70–79.

[20] Yang, J., Zhang, Y., Qiu, Z., Yue, Y., and Vinayak, R. FIFO queues are all you need for cache eviction. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023* (2023), J. Flinn, M. I. Seltzer, P. Druschel, A. Kaufmann, and J. Mace, Eds., ACM, pp. 130–149.

[21] Yang, T., Pollen, S., Uysal, M., Merchant, A., Wolfmeister, H., and Khalid, J. Cachesack: Theory and experience of google's admission optimization for datacenter flash caches. *ACM Trans. Storage 19*, 2 (2023), 13:1–13:24.