UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RUAN GOMES DE OLIVEIRA ALVES

A CASE STUDY OF LARC ADMISSION CONTROL ON WEB CACHING

CAMPINA GRANDE - PB

2023

RUAN GOMES DE OLIVEIRA ALVES

A CASE STUDY OF LARC ADMISSION CONTROL ON WEB CACHING

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Thiago Emmanuel Pereira da Cunha Silva

CAMPINA GRANDE - PB

2023

RUAN GOMES DE OLIVEIRA ALVES

A CASE STUDY OF LARC ADMISSION CONTROL ON WEB CACHING

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Thiago Emmanuel Pereira da Cunha Silva

Orientador – UASC/CEEI/UFCG

Franklin de Souza Ramalho

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 28 de Junho de 2023.

CAMPINA GRANDE - PB

RESUMO

O armazenamento em cache é crucial para melhorar o desempenho dos serviços da Web. O principal fator que determina a qualidade do armazenamento em cache é sua capacidade. Se a capacidade do cache for muito pequena, o desempenho do cache (por exemplo, taxa de acertos) será afetado. No entanto, não se pode simplesmente adicionar mais capacidade para melhorar o desempenho. Se a capacidade do cache aumentar além de um determinado ponto, o desempenho do cache não será aprimorado e a capacidade adicionada será desperdiçada. Para uma capacidade definida, os sistemas de cache aplicam uma política de substituição que decide quais itens devem ser removidos quando o cache estiver cheio. A literatura tem muitas políticas de substituição de cache. Neste trabalho, tomamos outra direção. Analisamos como economizar usando a capacidade do cache com base nas políticas de controle de admissão. Em vez de decidir quais itens devem ser removidos do cache, o controle de admissão decide se um item deve entrar no cache. Consideramos o LARC, uma conhecida política de admissão que permite que os itens sejam armazenados em cache apenas em seu segundo acesso. Avaliamos essa política de cache simulando uma carga de requisições de um grande serviço de comércio eletrônico. Os resultados indicam que quando o cache é superdimensionado, por ex. sua capacidade é maior que a ótima, a política de controle de admissão reduz o uso do cache em até 50%, com uma pequena penalidade de desempenho de 5%. Por outro lado, quando o cache é subdimensionado, a política de controle de admissão aumenta o desempenho do cache em até 22% sem nenhum custo no aumento do uso do cache. Acreditamos que esses resultados indicam que há potencial para integrar o controle de admissão ao gerenciamento de cache.

A Case Study of LARC Admission Control on Web Caching

Ruan Gomes
Federal University of Campina Grande
ruan.alves@ccc.ufcg.edu.br

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

ABSTRACT

Caching is crucial for improving the performance of Web services. The key factor that determines the quality of caching is its capacity. If the cache capacity is too small, caching performance (e.g. hit ratio) is impacted. However, one cannot simply add more capacity to improve performance. If cache capacity grows beyond a certain point, caching performance is not improved, and the added capacity is wasted. For a defined capacity, caching systems apply a replacement policy that decides which items must be removed when the cache is full. The literature has a lot of cache replacement policies. In this work, we take another direction. We analyze how to be thrifty using cache capacity based on admission control policies. Instead of deciding which items should be removed from the cache, the admission control decides whether an item should enter the cache. We considered LARC, a well-known admission policy that allows items to be cached only in their second access. We evaluated this cache policy by simulating a request trace of a large e-commerce web service. The results indicate that when the cache is oversized, e.g. its capacity is larger than the optimal, the admission control policy reduces the cache usage up to 50%, with a small 5% performance penalty. On the other hand, when the cache is undersized, the admission control policy increases the cache performance up to 22% with no cost on cache usage increase. We believe these results indicate that there is potential for integrating admission control into cache management.

KEYWORDS

Cache; web; performance; admission control

1 INTRODUCTION

Caching is a performance enhancement for web systems. It involves using a fast storage layer between clients and the target service. A web cache can be viewed as a key-value data structure in which the keys are the `URL`s representing the `http` requests, and the values are the corresponding responses to those requests. When a client requests data, the cache service checks whether it already has a copy of the response. If a copy was found (cache hit), the response could be quickly served without accessing the server directly, reducing response time and server load. If a copy was not found (cache miss), the service retrieves it from the server, stores the request and response in the cache, and returns the response to the client. Once a key-value pair (also known as `item`) is stored in the cache, the subsequent requests for the same item are addressed without requiring direct server access.

More than processing power and IO bandwidth, cache capacity is the most important factor for caching performance. However, as it is well known, cache performance could be improved by adding capacity only up to a certain point. Beyond this optimal capacity value, any increment in cache capacity is wasted since the performance (hit ratio) does not improve. Choosing a good capacity value requires careful consideration of many factors, including workload characteristics such as the popularity of cache items and temporal and spatial locality. In addition to the capacity, another important configuration choice is the cache replacement policy. The replacement occurs when the cache is full, and a new item needs to enter the cache; the policy decides which item needs to leave the cache to make room for the new one. The literature is rich on cache replacement policies [3, 5, 6, 8, 9]. These policies are heuristics that try to find and remove the item that is likely to be used in the farthest future. A good replacement policy improves cache performance by retaining in the cache the items more likely to be used in the near future.

In this work, we evaluate admission control policies. These strategies are complementary to the replacement ones. Instead of deciding which items must leave the cache when it is full, an admission control policy decides whether an item should enter the cache. In our evaluation, we considered the LARC admission policy, which allows an item to enter the cache only in the second request. When this heuristic makes the right choice, for example, when it avoids admitting an item accessed only once (also known as "one-hit wonder" [7]), it reduces capacity usage. When the heuristic makes a mistake, it impacts the performance since it turns the second request to an item from a hit to a miss.

To conduct this study, we gathered a 10% sample of requests from a multi-client caching service utilized by a large ecommerce provider, resulting in approximately 62 million observations over approximately 10 hours. To analyze the impact of our admission policy on cache performance, we simulated a cache load using a tool developed by our research group [1].

Through our simulations, we observed the relationship between cache capacity and the efficacy of our admission policy in enhancing cache performance. When cache capacity is insufficient, the LARC policy yielded a remarkable performance improvement of up to 21%. Conversely, for cache capacities that exceeded optimal levels, the application of the policy maintains the cache capacity utilization but leads to a slight degradation of cache performance, approximately 5%. These findings provide valuable insights for future studies and serve as a practical guide for implementing admission policies in cache services. Ultimately, our research aims to optimize capacity

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Trabalho de Conclusão de Curso, Bacharelado em Ciência da Computação, 2023
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

utilization and assist in the determination of optimal configurations for cache services.

The remainder of the paper is organized as follows. Section 2 provides background on cache management policies and discusses their importance for this type of service. Section 3 describes the methodology adopted to conduct this study, including aspects of the request load, the simulator employed for evaluating cache policies, and the admission policy used in this research. The results are discussed in Section 4. Finally, Section 5 describes future work and the importance of the results.

2 BACKGROUND AND RELATED WORK

In this Section, we present the key concepts for conducting this research. We delve into cache management policies, discussing their significance and implications for caching systems.

Cache policies dictate how data is managed within the cache. They determine how incoming requests are handled, including decisions on data replacement and admission. The choice of a cache policy directly impacts the efficiency and performance of the cache, so it is important to understand the motivations for using these policies.

Cache policies rest upon two empirical assumptions. The first assumption is temporal locality, which suggests that recently accessed items are highly likely to be re-accessed in the near future. The second assumption is the skewed popularity of items, implying that certain items are accessed more frequently than others. By incorporating these assumptions into policies, efficient caching strategies can be devised to enhance system efficiency. Two key types of cache policies are eviction and admission policies. They are not mutually exclusive, so it is possible to use an eviction and admission simultaneously.

A replacement policy determines which items are removed from the cache when its capacity limit is reached. The main objective of a replacement policy is to create space for new data while retaining the most relevant data in the cache. In many cases, to choose these policies one must understand the characteristics of the cache usage. These characteristics are presented, for example, in the degree of temporal locality of the access. High temporal locality indicates that recently accessed items are highly likely to be re-accessed in the near future. Another important characteristic is the disparity of the popularity of the items; a small portion of the items could be responsible for a large fraction of the requests to the cache.

Two well-known replacement policies that consider these characteristics are the Least Recently Used (LRU) and Least Frequently Used (LFU). The LFU policy tracks the frequency that the stored items are requested and selects the items with the least frequency to be replaced. In another way, the LRU policy organizes items in order of use and selects the least recently used items to be replaced. LRU is the most commonly used policy in production (including the cache studied in this work) due to its performance and ability to store popular items. However, LRU may exhibit poor performance in scenarios with weak temporal locality, where the workload consists of items accessed only once. In such cases, the cache space can quickly be filled with one-time accessed items, leading to a degradation in service performance because these items are not accessed in the near future, replacing items that would potentially be accessed.

Another replacement policy is the Adaptive Replacement Cache (ARC) [5] policy. This policy utilizes two segments T1 and T2, that share the overall cache storage space and stores the keys and the item's values. Each segment includes a queue known as `ghost` queue, B1 and B2, which stores item keys exclusively. B1 assists the T1 segment in keeping track of the frequency (LFU) that the items are requested, keeping stored in this segment the most frequently requested items. On the other hand, B2 manages the least recently used (LRU) items keys, assisting in storing these items in the T2 segment. The adjustment of segment sizes is controlled by the P variable, tuned by the algorithm that observes the B1 and B2 queues hit ratio. When the segments and the queues are full, items are removed by importance order, according to the purpose of the queue. This adaptive mechanism allows this policy to consider not only the frequency or the least recently used to keep stored items but both of them, enhancing the effectiveness of the caching strategy.

Unlike replacement policies, cache admission policies decide whether incoming data should be cached. An effective admission policy ensures that only valuable and frequently accessed data is stored in the cache, leading to optimized cache utilization and improved overall system performance. Notably, a well-designed admission policy can address the limitation of the LRU algorithm by preventing the storage of items accessed only once, thereby enhancing the cache's effectiveness.

A well-known example of admission policy is the Lazy Adaptive Replacement Cache (LARC) [3], which despite its name, serves as an initial filter for storing items in the cache. This algorithm incorporates two segments with fixed sizes: the main cache storage that stores item keys and values and the `ghost cache` segment, dedicated to storing only items key. In the LARC policy, when an item is requested for the first time, it is not immediately stored in the main cache. Instead, only the item's key is stored in the ghost cache. The item is promoted to the main cache only upon the second request if this item is present in the ghost cache. Both the main cache and the ghost cache have their respective LRU structures. Consequently, when either cache segment becomes full and needs to store a new item, a stored item can be removed to give space to the new item

LARC was developed to improve performance and decrease IO operations in caches that use SSDs. In our work, we adopted a simplified LARC algorithm to simplify the execution of experiments. In our modification, we did not consider space limitation and removal strategies (since there is no space limitation). As there is no space limitation, our results are an upper bound to the original LARC algorithm.

3 METHODOLOGY

This Section overviews the methodology used to conduct this work. Section 3.1 describes the workload used, Section 3.2 describes how we obtained the results, Section 3.3 describes the admission policy used to conduct this work, and Section 3.4 explain the values of the used parameters.

3.1 Workload

The collected load originates from requests made to one of the caching services of a large ecommerce provider. This cache service serves multiple tenants, meaning it is used by more than one service. We sampled 10% of the requests received by the service and then collected them not to impact the performance of the service that runs in production. Despite being a sample, the collected load contains a sufficient number of requests to perform the necessary analyses for this work. Figure 1 shows the workload over time.

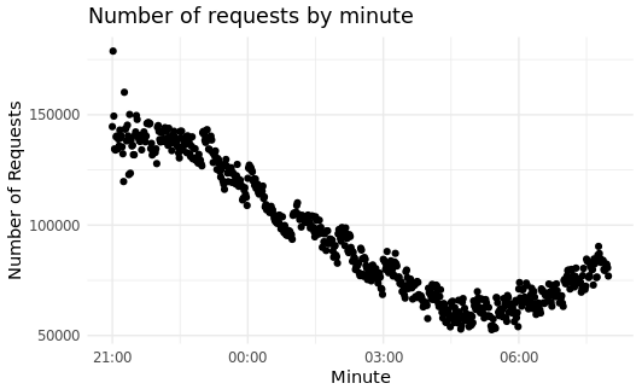

Figure 1: Cache load, in requests per minute, along 10 hours.

The collected trace consists of approximately 62 million requests, gathered over a continuous period of approximately 10 hours, which is an interval that has a significant request quantity to be used in performed analyses. This trace has an average load rate of approximately 1500 requests per second. Due to the extended time interval, the load exhibits variability in the number of requests over this duration, which can be attributed to the access patterns during the collection period. The collected information includes the request timestamp, request item identifier, and the request response status. The collected request statuses are as follows: (1) MISS: The requested item was not found, (2) HIT: The requested item was found and returned to the client; and (3) STALE: The requested item was found and returned to the client but may be expired, necessitating an update.

Table 1 exhibits the distribution of items based on the following frequency types: "one request," "two requests," and "more than two requests" within the load. The table shows the corresponding percentages for each category.

Approximately 25 million different items were requested, with approximately 68% of them being requested only once as it is showed in Table 1, corresponding to around about 17 million items. A significant portion of items can be filtered rather than cached. In turn, items that are requested more than once can be understood as opportunities for performance improvement since they are more popular, therefore they will be reused more if they are stored in the cache. Therefore, it is crucial to thoroughly analyze the load to address the caching of these items, taking into account the potential issues that may arise.

3.2 Cache simulator

To conduct this study, we simulated the load described in Section 3.1. This approach was chosen for its cost-effectiveness and efficiency in obtaining results, thereby streamlining the analysis process. An alternative approach would involve replicating the architecture of the e-commerce provider's service from which the payload was collected. However, this would be prohibitively expensive.

The simulations were executed using a cache workload execution tool developed by our research group [1], which boasts an error rate of less than 1%, which simulates NGINX, a popular cache service. This tool enables the simulation of GET requests. The tool utilizes the LRU (Least Recently Used) replacement policy, which aligns with the policy used by the service from which the load was collected. It allows for the selection of various important parameters for the conducted analyses, including: (1) Cache Capacity: controlling the maximum number of items that can be stored in the cache (integer value), the tool permits the use only the number of items as the capacity value, not storage space.; (2) Admission Policy: managing the admission of items into the cache, with a boolean value representing the use (true) or non-use (false) of the LARC admission policy studied in this work (as described in section 3.3); (3) Time-To-Leave: defining the expiration time of an item, indicating when an item should be updated in the cache and serving as the basis for the STALE request status mentioned earlier; (4) Expiration time: Being the time set for an item not accessed in that interval to be removed from the cache. The tool effectively executes cache loads, resulting in three response types for the aforementioned requests: MISS, HIT, and STALE.

In the simulations, it is important to note that the cache starts empty at the beginning of the simulation. Extracting results from an empty cache can yield unfavorable outcomes, e.g. the initial low storage utilization could significantly affect the cache utilization metric. To address this, we simulate a 'running' cache in a production environment by considering the first 15 million requests of the payload across all simulated cache configuration scenarios to populate the cache storage.

3.3 Admission policy

As mentioned earlier, the admission control mechanism employed in this study is an adaptation of the LARC algorithm used with a web cache workload. The algorithm prevents requested items from being immediately stored in the cache upon their initial request. Instead, it utilizes a storage space to retain the identifiers of requested items, allowing them to be stored in the cache only when a second MISS request occurs. It's worth noting that the storage space consumes minimal memory as it solely stores item keys, which are significantly smaller than their corresponding values.

3.4 Simulations parameters

Table 2 overviews the parameters employed in the simulations. The parameters item expiration time, item time-to-live, and replacement policy reflect the original configuration values obtained from the cache service that the payload used in this work was collected. The capacity values were specifically chosen based on the number of items requested more than once present in the load, turning possible to explore oversized and undersized cache configurations.

Table 1: Percentage Distribution of Workload Item Frequency

Item Frequency Type	Percentage
`One Request`	68.7%
`Two Requests`	13.2%
`More Than Two Requests`	18.1%

Table 2: Selected cache simulation parameters

Parameter	Value
`Cache Capacity`	(from 500K to 2.5M) and (from 3M to 8M)
`Item time-to-leave`	25 minutes
`Item expiration time`	3 hours
`Replacement Policy`	Least Recently Used (LRU)

4 RESULTS

The performance metric employed in this study is the Hit Ratio, which is widely used for evaluating caches. The proportion of hit responses made for cached items. It is important to note that the cache capacity affects the hit ratio. If the capacity is insufficient, the cache may be unable to store many popular items. Conversely, it is possible that the capacity can be reduced if the peak hit ratio can be achieved with smaller capacities.

The higher the hit ratio, the better the performance. The hit ratio is closely tied to the capacity of the cache, which is a significant factor in resource utilization for a caching service. Effective management of the cache capacity is crucial to optimize resource usage. The goal is to minimize resource usage while maximizing the hit ratio. By efficiently allocating and utilizing cache resources, the caching service can improve overall performance and efficiency.

Simulations were run with various cache capacities scenarios, using and not using the proposed admission policy. This allows for comparisons of the use of the admission policy, observing improvements in hit ratio and capacity.

Figure 2 illustrates the enhancements achieved by implementing the admission policy at each of the chosen cache capacities, highlighting the observed improvements in hit ratio and cache usage.

The load consists of a large number of items that are requested only once, totaling approximately 25 million different items in the workload. Out of these, approximately 17 million are one-request items, accounting for approximately 68% of the total items.

Figure 2 indicates the simulated capacities and reveals that not using an admission policy results in a full cache, as the total number of items exceeds all simulated capacities. Conversely, employing the admission policy allows less than 5 million items to enter the cache. Therefore, simulations utilizing the admission policy and having capacities smaller than this value exhaust their storage space, explaining the 0% improvement observed for simulations that have capacities equal to or below 4 million items.

As mentioned, the admission policy permits maximum capacity utilization of fewer than 5 million items. Consequently, Figure 2 also demonstrates that the characteristics of this load contribute

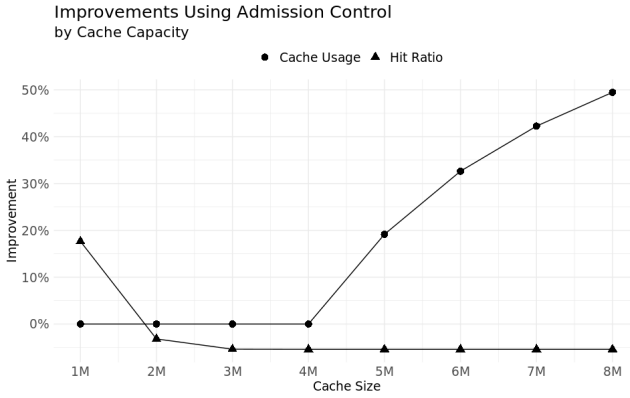

Figure 2: Significant simulation results using LARC admission control show hit rate improvements for capacities less than 3M items and improvements in storage utilization for capacities greater than 4M items

to resource usage improvements in cache with capacities exceeding 4 million items. It's possible to understand the cache usage improvement in simulations with capacities equal to or bigger than 5 million items.

Figure 2 also provides insights into the impact of using the admission policy on the hit ratio. Simulations with capacities greater than 3 million items experience a consistent deterioration of approximately 5% in hit ratio compared to those without the proposed admission policy. This decline is attributed to fewer items being stored in the cache, resulting in more MISS requests that could have been HIT if there had been an opportunity to store the requested items.

Furthermore, simulations with capacities below 3 million items exhibit an improved hit ratio compared to others. To present these findings more clearly, Figure 3 offers a focused depiction of the results presented in Figure 2.

Figure 3 provides insights into the simulations and reinforces that none of the simulations presented show improvements in capacity utilization, as explained earlier. However, it is noteworthy

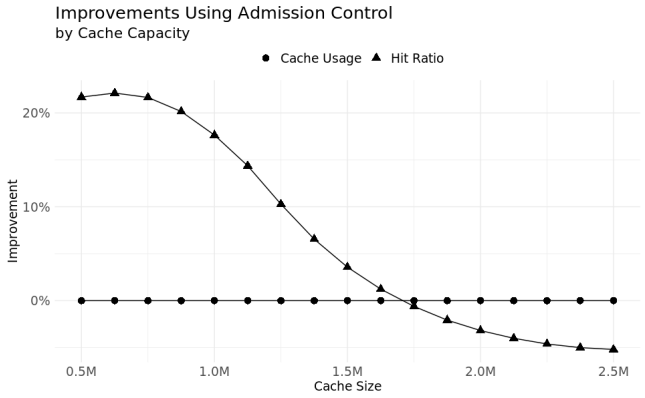

Figure 3: Hit ratio obtained from simulations with cache capacities fewer than 3M using LARC admission control

that simulations with capacities below 1.75 million items exhibit a remarkable enhancement in the hit rate, with the maximum value reaching approximately 22%. This finding indicates a significant improvement in cache service performance when admission policies are implemented for caches with low capacities.

The observed hit ratio improvement in simulations with capacities below 1.75 million items can be attributed to two potential reasons: (1) The primary purpose of the LRU algorithm is to retain items that are more likely to be reused. It achieves this by replacing items that are not frequently used with those that are expected to be reused. (2) Another contributing factor could be temporal locality fault, meaning that there are no requests for the same items within short time intervals. In such cases, admission control prevents infrequently requested items from entering the cache. These characteristics ensure that more popular items remain stored in the cache, resulting in a higher rate of successful HIT requests.

5 CONCLUSION AND FUTURE WORK

This paper has analyzed using the LARC admission policy in a web caching service. Based on the simulation results, it is evident that under-dimensioned cache capacities exhibit a substantial improvement of approximately 22% in the hit ratio. Conversely, caches with larger capacities exceeding 3 million items experience a decline of approximately 5% in the same metric. Furthermore, we observed notable improvements in capacity utilization for simulations with bigger than 4 million capacities.

The results of this study indicate substantial potential for enhancing web caching performance by implementing an admission policy. While the focus of this research was on investigating the LARC policy, it is important to acknowledge other policies in the literature that warrant exploration in future studies, e.g. ML-based [2] and policies that use Bloom Filter strategies [4]. A comparative analysis of these policies in web cache scenarios can assist the developing new policies. To achieve this, it is necessary to implement these policies and evaluate their effectiveness using diverse cache workloads, including both synthetic and obtained from server collections. Comparison of simulation results using these policies will provide valuable insights into the efficacy of different policies and their applicability in real-world settings.

REFERENCES

[1] Gabriela Roberta Alberga do. 2023. Development of a cache simulator for analysis of management policies. *Sistemoteca - Sistema de bibliotecas da UFCG*. http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/29313

[2] Assaf Eisenman, Asaf Cidon, Evgenya Pergament, Or Haimovich, Ryan Stutsman, Mohammad Alizadeh, and Sachin Katti. 2019. 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*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 65–78. https://www.usenix.org/conference/nsdi19/presentation/eisenman

[3] Sai Huang, Qingsong Wei, Dan Feng, Jianxi Chen, and Cheng Chen. 2016. Improving Flash-Based Disk Cache with Lazy Adaptive Replacement. *ACM Trans. Storage* 12, 2 (2016), 8:1–8:24. https://doi.org/10.1145/2737832

[4] Bruce M. Maggs and Ramesh K. Sitaraman. 2015. Algorithmic Nuggets in Content Delivery. *Comput. Commun. Rev.* 45, 3 (2015), 52–66. https://doi.org/10.1145/2805789.2805800

[5] Nimrod Megiddo and Dharmendra S. Modha. 2003. 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*, Jeff Chase (Ed.). USENIX. http://www.usenix.org/events/fast03/tech/megiddo.html

[6] Vivek R. Narasayya, Ishai Menache, Mohit Singh, Feng Li, Manoj Syamala, and Surajit Chaudhuri. 2015. Sharing Buffer Pool Memory in Multi-Tenant Relational Database-as-a-Service. *Proc. VLDB Endow.* 8, 7 (2015), 726–737. https://doi.org/10.14778/2752939.2752942

[7] M. Zubair Shafiq, Amir R. Khakpour, and Alex X. Liu. 2016. Characterizing caching workload of a large commercial Content Delivery Network. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524379

[8] Tzu-Wei Yang, Seth Pollen, Mustafa Uysal, Arif Merchant, Homer Wolfmeister, and Junaid Khalid. 2023. CacheSack: Theory and Experience of Google's Admission Optimization for Datacenter Flash Caches. *ACM Trans. Storage* 19, 2 (2023), 13:1–13:24. https://doi.org/10.1145/3582014

[9] Yuanyuan Zhou, James Philbin, and Kai Li. 2001. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In *Proceedings of the General Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA*, Yoonho Park (Ed.). USENIX, 91–104. http://www.usenix.org/publications/library/proceedings/usenix01/zhou.html