JOIV : Int. J. Inform. Visualization, 8(1) - March 2024 31-37

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Application-Level Caching Approach Based on Enhanced Aging Factor and Pearson Correlation Coefficient

Mulki Indana Zulfa [a,*], Sri Maryani [b], Ardiansyah [c], Triyanna Widiyaningtyas [d], Waleed Ali [e]

[a] *Electrical Engineering Department, Jenderal Soedirman University, Kalimanah, Purbalingga, 53371, Indonesia*
[b] *Mathematic Department, Jenderal Soedirman University, Karangwangkal, Purwokerto, 53122, Indonesia*
[c] *Informatic Department, Ahmad Dahlan University, Kragilan, Bantul, 55191, Indonesia*
[d] *Electrical Engineering Department, Malang State University, Lowokwaru, Malang, 65145, Indonesia*
[e] *Information Technology Department, King Abdulaziz University, Rabigh, 25732, Saudi Arabia*

Corresponding author: [*]mulki_indanazulfa@unsoed.ac.id

***Abstract*— Relational database management systems (RDBMS) have long served as the fundamental infrastructure for web applications. Relatively slow access speeds characterize an RDBMS because its data is stored on a disk. This RDBMS weakness can be overcome using an in-memory database (IMDB). Each query result can be stored in the IMDB to accelerate future access. However, due to the limited capacity of the server cache in the IMDB, an appropriate data priority assessment mechanism needs to be developed. This paper presents a similar cache framework that considers four data vectors, namely the data size, timestamp, aging factor, and controller access statistics for each web page, which serve as the foundation elements for determining the replacement policy whenever there is a change in the content of the server cache. The proposed similarCache employs the Pearson correlation coefficient to quantify the similarity levels among the cached data in the server cache. The lowest Pearson correlation coefficients cached data are the first to be evicted from the memory. The proposed similarCache was empirically evaluated based on simulations conducted on four IRcache datasets. The simulation outcomes revealed that the data access patterns, and the configuration of the allocated memory cache significantly influenced the hit ratio performance. In particular, the simulations on the SV dataset with the most minor memory space configuration exhibited a 2.33% and 1% superiority over the SIZE and FIFO algorithms, respectively. Future tasks include building a cache that can adapt to data access patterns by determining the standard deviation. The proposed similarCache should raise the Pearson coefficient for often available data to the same level as most accessed data in exceptional cases.**

***Keywords*— Application-level caching; pearson correlation coefficient; cached data; hit ratio.**

Manuscript received 24 Sep. 2023; revised 18 Nov. 2023; accepted 13 Dec. 2023. Date of publication 31 Mar. 2024.

I. INTRODUCTION

Relational database management systems (RDBMS) have been the fundamental infrastructure for most web applications since computers proliferated [1]. With information considered a primary asset, efficient and rapid data storage, access, and manipulation are of utmost importance [2]. A relational database management system (RDBMS) facilitates the construction of intricate applications through its structured table arrangement and interpretable associations, enabling developers to run intricate data queries [3]. Furthermore, RDBMS provides notable benefits in data integrity, security, and the capability to handle concurrent transactions [1]. When data can be accessed and modified concurrently by several users in real-world online applications, the properties above assume significant importance [2]. Thus, selecting an appropriate Relational Database Management System (RDBMS) and implementing effective database design play crucial roles in determining the triumph of online applications [3].

Many web applications continue to employ RDBMS as their primary storage medium due to its ability to maintain well-structured data [4], [5]. However, these RDBMS architectures are characterized by relatively slow access speeds as data are stored on disk [6]. In contrast, in-memory database (IMDB) technology has experienced rapid growth. It is widely adopted by cloud service providers such as Amazon Web Services, Google Cloud Platform, IBM, and Microsoft Azure [7]–[9]. IMDB stores data in computer memory rather than on disk, resulting in significantly faster access speeds [10], [11].

IMDB, also known as a NoSQL database [12], [13], realizes extensive application as a server cache to alleviate server workloads and reduce internet latency [14], [15]. However, IMDB exhibits a notable limitation in that it does not guarantee suitable ACID properties [4] [16], including desirable atomicity, consistency, isolation, and durability. Consequently, IMDB is not yet positioned to supplant RDBMS as the primary database system [9]. Moreover, implementing IMDB within web applications hosted on shared-hosting platforms replicated and distributed across various geographic locations poses challenges in maintaining privacy and trust [17]. Thus, a coalescence of IMDB and RDBMS concepts can be leveraged to support transactions that ensure ACID compliance while preserving swift data access performance [16]. A pioneering example of such integration is exemplified by Megastore [18]. In the Megastore framework [18], data are partitioned to guarantee the isolation of ACID semantics within each partition, thereby upholding the consistency property.

Determining data priorities thus becomes a crucial consideration due to the minimal memory capacity. To address this issue, some research has utilized machine learning methods to create replacement policies or memory content replacement methods [19]. However, using machine learning techniques in this context entails computationally intensive training processes [20], [21]. Therefore, in recent years, several researchers have introduced the concept of application-level caching (ALC), which is more flexible and can be implemented in real-world web applications [22]. The schematic caching framework [23] proposes a query parser to break down query results into unique key values to be stored in IMDB. The Hyperbolic caching framework [24] proposes access frequency and access time variables to assess the priority of cached data before it is stored in the cache server. In addition, an APLCache caching framework [20] proposes reactive and proactive caching mechanisms by studying user access behavior based on access frequency, memory consumption, and staticity. However, several caching frameworks fail to address changeable decisions that lack robustness due to the necessity of aligning caching logic with web development architecture.

This research proposes a caching framework by considering the access controller in the MVC architecture (Models-Views-Controllers) combined with several other variables such as access count, data size, timestamp, and aging factor to make the caching decision more comprehensive. Furthermore, all caching decision variables are formulated using the Pearson correlation coefficient to calculate the similarity of each cached data item with respect to its top-accessed counterparts. When a request for memory cache replacement arises, the data item with the lowest correlation level is removed from memory first.

II. Materials and Methods

A. Related Works

Application-level caching is a software development methodology that leverages memory as a popular data storage medium, enabling repeated access without the need to retrieve data from the Relational Database Management System (RDBMS) [25]. Typically, ALC approaches involve manual implementation by application development teams, including the development of code functions that direct data storage into memory [20]. This approach is time-consuming and requires extensive source code modifications in response to any changes in business processes that affect cached data priorities. ALC approaches must therefore consider factors such as hit ratio performance, data access characteristics, and data change frequencies [26].

ALC research has focused on three methods: weighting, machine learning, and optimization. Ma et al. [27] proposed the caching framework WSCRP, which calculates the weight of cached data by incorporating variables such as data size (S_i), network cost (csv_i), request time start-end (Δt_i), aging factor (L), and access frequency (F_i).Generally, the corresponding weight is as described in Eq. (1). Ma et al. [28] also introduced the caching framework WGDSF, which considers the weight of document types (WDT) and their access frequency (WTF). This is then combined with the greedy dual size frequency (GDSF) caching algorithm on the aging factor variable (L) and network cost (SC), as seen in Eq. (2).

Akbari et al. [19] proposed the FPRA caching framework based on machine learning, utilizing the Fuzzy C-Means (FCM) algorithm. Each cached data item in memory is grouped into respective clusters based on three parameters: access recency (PR), access frequency (PF), and reference rate $P\delta_{tk}(j)$. When there is a need to store new data in memory, FPRA removes the member of the cluster with the smallest cumulative reference time (Δt_n) first, as indicated by Eq. (3). Zhang et al. [29] suggested the use of data cache clustering with variables R (interval time), F (frequency), and S (size) employing the K-Means algorithm. Variables R and S are sorted based on their smallest values, while variable F is sorted based on its largest value. Cached data with the smallest cumulative RFS value is removed first when there is a need to store new data, followed by Eq. (4):

$$WC = \frac{S_i}{csv_i} * \frac{L_i}{F_i * \Delta t_i} \tag{1}$$

$$H(j) = L + SC(j) * WDT(j) * WTF(j) \tag{2}$$

$$PR_j = \left(\sum_{n=0}^{(k-1)} \Delta t_n\right) - (k-1) \tag{3}$$

$$RFS_{x,y} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_n - x_n)^2} \tag{4}$$

$$p_i = \frac{n_i}{t_i} \tag{5}$$

In addition to these two primary caching methods, caching weighting and caching mining, Blankstein et al. [30] introduced the hyperbolic caching (HC) framework, which calculates cached data priority (p_i) using access frequency (n_i) and access time (t_i) since data entered the cache, as illustrated in Eq. (5). The fundamental concept behind HC is to separate the cached data priority calculation function from its structure. This approach differs from previous proposals where the cached data priority calculation method influenced its position in memory. Another method within the ALC realm was suggested by Mertz et al., which achieved

simplicity by embedding caching logic code to consistently provide data services directly from memory using method calls [20] [26].

Mertz et al. [20] proposed a reactive caching mechanism using method calls while the application is running and then built its caching model based on the nested cacheability pattern [31] [22]. This nested caching pattern approach is prone to suboptimal solutions. Their research was then improved with a metaheuristic optimization method using Ant Colony and Genetic Algorithm algorithms so that it is not easily trapped in local optimum solutions [32]. The data cache weighting methods proposed by [31] and [22] are also challenged at determining the optimal solution because they rely on greedy methods completely. In addition to these studies, a machine learning-based caching system using FCM has also been proposed, but this research will be difficult to implement in real-world DBMS web applications because it adds computationally free databases, web servers, and dataset training. Although these studies have weaknesses, the concept of using aging factors in [28] and [32] can be well adopted to solve the problem of cache pollution. Meanwhile, the concept of calculating the proximity distance between two data caches using Euclidean distance [19] as a caching decision consideration was eventually adopted into Pearson correlation since this method more strongly refers to the substance of the linear relationship between two data caches. Furthermore, method calls proved to be reliable for implementation in real-world web applications [22].

Based on the literature study, the proposed similarCache framework fills a research gap in application-level caching for real-world web applications. This research is motivated by the convenience and ease offered by systems with method calls, while recognizing that each cached data item possesses numerous properties that can be leveraged to propose a robust and comprehensive cache replacement policy. Therefore, this paper presents a caching framework denoted as similarCache, which adopts method calls to map each data access to its respective controller. The relationship between data and controllers stands as a key element that necessitates continuous updates. Subsequently, the coefficient of correlation for each cached data item is computed concerning the most frequently accessed data items at that moment. Cached data items with the lowest correlation coefficients are evicted from memory first. The operational mechanism of the proposed similarCache framework is described below.

B. Look-Aside Caching

The proposed similarCache framework employs a topological look-aside caching approach to ensure that every data request is promptly searched within the memory. This technique is implemented because the primary objective of ALC is to enhance user-side response times. Internet users tend to bypass and seek alternative websites if they encounter slow response times [33] [34]. Response time is a critical metric for developers of web-based applications and cloud network infrastructure managers because it can significantly impact user satisfaction and comfort [35] [36]. The operational mechanism of the look-aside caching architecture, as illustrated in Fig. 1, is described as follows.

Based on Fig. 1, the look-aside caching adopted by the similarCache proposal prioritizes data responses sourced from the server cache. The caching system receives the signal from the client and sends it to the cache and database controllers. The cache controller looks for the requested data. If HIT or data are found, these data are immediately given back to the client. However, if the requested data is not found, then the data request signal received by the database controller is immediately sent to the RDBMS. The RDBMS provides the requested data to the database controller and forwards it to the controller cache. The cache controller stores the data as new in cache memory and then forwards it directly to the client.

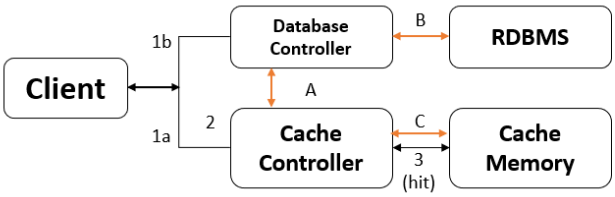

Fig. 1 Look-aside caching mechanism [37]

C. Method Calls

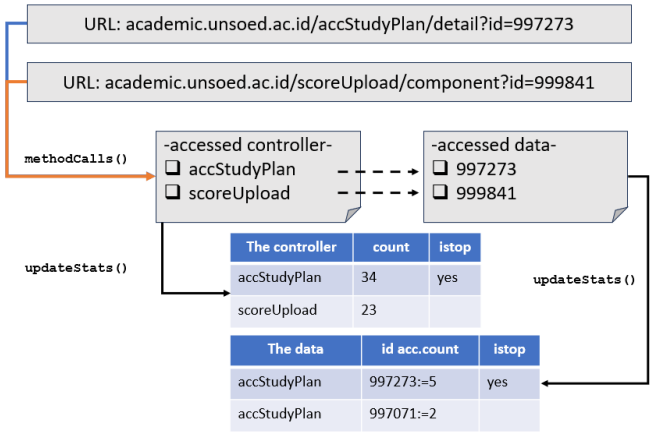

Fig. 2 Method calls mechanism

Fig. 2 provides an illustration of how method calls function within a web application, effectively mapping each URL access within the application server. In many cases, web applications built using the MVC (Models-Views-Controllers) framework feature multiple controllers responsible for interacting with the database. However, there are instances when only a select few controllers are in high demand. For instance, we consider an academic information system at a university. At the start of the semester, the StudyPlan Controller sees its highest access frequency, but toward the end of the semester, the StudyResult Controller becomes the most heavily accessed. Access patterns for these controllers remain relatively consistent, underscoring the importance of method calls in breaking down access to each data ID. As a result, method calls vitally contribute to pinpointing the currently popularly accessed controllers, thus influencing the prioritization of data loaded into them.

D. Correlation Coefficient

The proposal of the similarCache framework, which calculates the correlation coefficient for each cached data item with the top-accessed data, draws inspiration from the caching cluster model [19]. Essentially, caching system technology

must be capable of swiftly determining whether a piece of data should be placed in memory. Consequently, implementing a caching policy based on a machine learning approach can be challenging to realize in practice. However, a different scenario arises when machine learning technology is utilized to analyze user data access patterns, which subsequently influence the selection of items offered by a marketplace, for example.

Thus, the similarCache framework proposal adopts the concept of similarity calculation of the caching cluster model [19] as a method for determining the priority of data to be placed in memory. Whenever similarCache needs space to accommodate new data in memory, it must perform two fundamental tasks: (1) establish the top-accessed data and (2) calculate the correlation coefficient for each data item in memory concerning the top-accessed data. This concept for top-accessed data resembles the notion of centroids that are used for clustering algorithms in machine learning. Based on our prior research following [38], we have determined that using the least recently used value and aging value of the Greedy-Dual Size Frequency (GDSF) algorithm offers respective advantages, including the capability to implement different access patterns. Therefore, we assign our top-accessed data values based on the least recently used data, the aging factor $K_{(i)}$, access frequency $F_{(i)}$, network cost $C_{(i)}$ and data size $S_{(i)}$, as shown in Eq. (6). Ultimately, data with the lowest correlation coefficient will be removed to make room for new data to occupy memory. Eq. (7) represents the Pearson correlation coefficient formula between two cached data $\rho(x_i, y_i)$ used by the similarCache framework to make caching decisions with its top-accessed (x). The next section will provide a more detailed explanation of the data vectors that similarCache employs for computing this correlation coefficient:

$$K_{(i)} = L + F_{(i)} * \frac{C_{(i)}}{S_{(i)}} \tag{6}$$

$$\rho(x,y) = \frac{\sum_{i=1}^{n}(x_i - x) \times (y_i - y)}{\sqrt{\sum_{i=1}^{n}(\Delta x_i)^2} \times \sqrt{\sum_{i=1}^{n}(\Delta y_i)^2}} \tag{7}$$

E. The Proposed SimilarCache

TABLE 1.
WEBSITE DATA PROPERTIES.

No	Properties	Example 1
1	iddata	4242106418
2	URL	http://1stnatbk.com/images/2236int_r13_c1.gif
3	controller	/images/
4	data	2236int_r13_c1.gif
5	timestamp	1282592384.330 (baseline: top-accessed data)
7	size	465 KB

$$norm_{(contr)} = \frac{1}{n}\sum_{i=1}^{n}\frac{ct_{(i)} - ct_{min}}{ct_{max} - ct_{min}} \tag{8}$$

Data vectors in the web application domain typically possess several properties, as indicated in Table 1. SimilarCache records every access statistic for these data items. Subsequently, these data can be revisited if similarCache needs to execute a replacement policy, as illustrated in Fig. 2. Notably, each time a replacement policy is executed, the top-accessed data are reset by similarCache. All other property values associated with these top-accessed data serve as the baseline for calculating the correlation coefficients for all cached data items within memory.

The similarCache framework proposal designates recently used data as the top-accessed data, as based on our previous research findings, the recently used data property demonstrates good caching performance [38]. However, it is essential to note that the value of the recently used data property may not be entirely reliable because its performance in specific data access patterns is no better than that of cost-based and aging factor-based algorithms [32]. Therefore, the similarCache framework proposal also incorporates an aging factor (Eq. 6) In the final calculation of data similarity using the Pearson correlation coefficient.

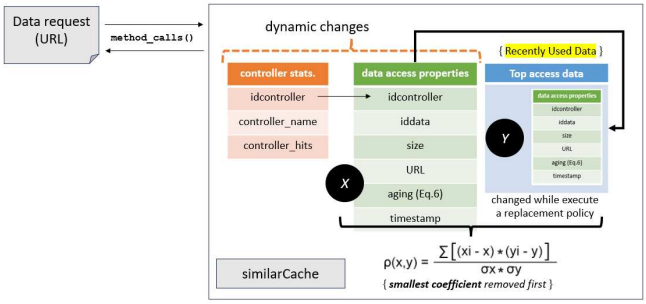

Fig. 3 The Pearson correlation coefficient in the proposed similar cache

Figure 3 demonstrates the functioning of the similarCache system, which uses all cached data attributes mentioned in Table 1 and the aging factor value derived from Equation 6. According to Equation 7, the variable x represents the property values of the most frequently requested data, whereas the variable y represents the property values of all other cached data items. The correlation coefficients for all cached data items in memory are determined about the most frequently accessed data at that specific moment. Consequently, the data item stored in the cache with the lowest correlation coefficient is prioritized for removal from the memory and replaced with different data. An innovative idea implemented in this study is the inclusion of the controller access patterns while calculating the final similarity. This concept is implemented based on our acknowledgment that controller access statistics in web applications may display fluctuations. Consequently, we apply the standard min-max approach to normalize each controller access $(ct_{(i)})$ using Equation 8. The outcome of this normalizing procedure produces controller access statistics that range from 0 to 1. Once all values in Table 1 have been normalized, the proposed similarCache architecture will compute the cached data for all data that is currently being accessed the most. This technique is implemented whenever there is a request to allocate storage for a new data cache. The cache server will remove the data cache that has the lowest Pearson Coefficient value and replace it with the new data cache.

The proposed similarCache framework aims to overcome the technical constraints in application-level caching research for real-world online applications by implementing proven strategies that enhance hit ratio performance. The proposed

similarCache has multiple benefits, as seen by the description of the proposed technique in Figure 2. One advantage of similarCache is its capacity to minimize cache pollution by selecting the least recently used material as a reference for caching before inserting it into the server cache. The LRU algorithm has been demonstrated to be highly effective in achieving a high hit ratio [38]. Another benefit is that the caching choice is more thorough as it considers several factors, including the access controller, access count, data size, date, and aging factor.

The design of the caching system must be comprehensively considered so that it does not cause network bottlenecks and reduce the performance of the database and web server. Generally, caching systems are developed uniquely by researchers according to the case study, whether the goal is to increase the hit ratio or reduce the bandwidth usage by maximizing the byte hit ratio [39]. This decision represents a trade-off that must be chosen in designing a caching system [40]. Not all goals can be simultaneously realized, i.e., the features of one caching algorithm cannot be entirely superior to those of other caching algorithms [41] [42].

III. Results and Discussion

A. Result

Based on the simulations conducted on four IRcache datasets, the hit ratio (HR) performance of the proposed similarCache framework proves to perform well when compared to the commonly used SIZE and FIFO algorithms for implementing replacement policies. These four IRcache datasets exhibit varying access patterns, resulting in distinct maximum hit ratio performances. The maximum hit ratio performance is observed when utilizing the largest memory size configuration, analogous to the scenario where all data can be perfectly accommodated within the caching server's memory. However, practical instances are faced with data access growth consistently outpacing and exceeding the memory capacity that can be allocated. Thus, the proposition of an appropriate replacement policy method can significantly maximize the utility of the constructed caching system.

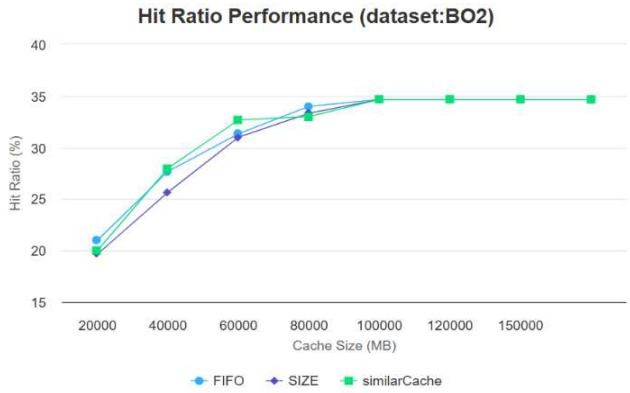

Fig. 4 HR Performance Using the BO2 Dataset

Fig. 4 illustrates the hit ratio performance using the BO2 dataset, as characterized by moderately cacheable requests compared to other datasets. Based on the maximum memory size configuration, all three replacement policy methods achieve an optimal hit ratio performance of 34.67%. The hit ratio performance of the proposed similarCache framework with the smallest memory configuration reaches 19.67%, only 1.33% behind the FIFO algorithm. However, when the memory configuration is increased twofold, the similarCache framework outperforms the others with hit ratio performances of 32.67%, 31%, and 31.33%, respectively. In the final configuration, all three cache replacement methods achieve optimal hit ratio performance.

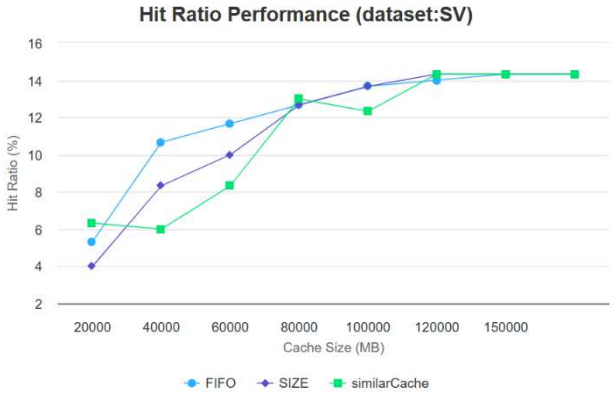

Fig. 5 HR Performance Using the SV Dataset

Fig. 5 illustrates the hit ratio performance on the SV dataset. Based on our statistical information, the SV dataset features the highest number of unique requests, resulting in the lowest hit ratio performance among the three datasets. The highest hit ratio performance of 14.33% can only be achieved with the largest cache configuration. This result is markedly different from the simulations on the BO2 dataset, where hit ratio performance was achieved in the last three memory configurations. The proposed similarCache framework exhibits the best hit ratio performance of 6.33%, surprisingly in the smallest memory configuration. In this configuration, similarCache outperforms the SIZE and FIFO algorithms by 2.33% and 1%, respectively.

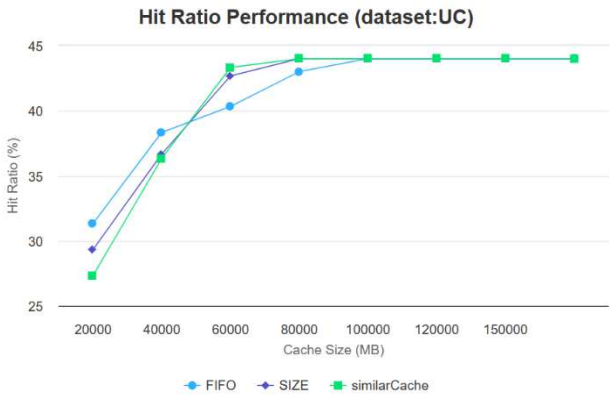

Fig. 6 HR Performance Using the UC Dataset

Fig. 6 and Fig. 7 showcase the simulation outcomes of similarCache on the UC and NY datasets, respectively. The NY dataset stands out due to its significantly lower count of unique requests compared to the other three datasets. This unique attribute enables the system to achieve the highest hit ratio performance because it can accommodate a vast amount of data in a memory cache. Another noteworthy distinction is that in the last four configurations, all replacement policy

methods manage to attain optimal hit ratio performance. In these scenarios, application server administrators reap substantial benefits, as they are spared the need for significant investments in large memory capacities. However, the access patterns observed in the NY dataset represent an anomaly, resembling a viral access phenomenon that occurs briefly during specific times and then quickly dissipates.

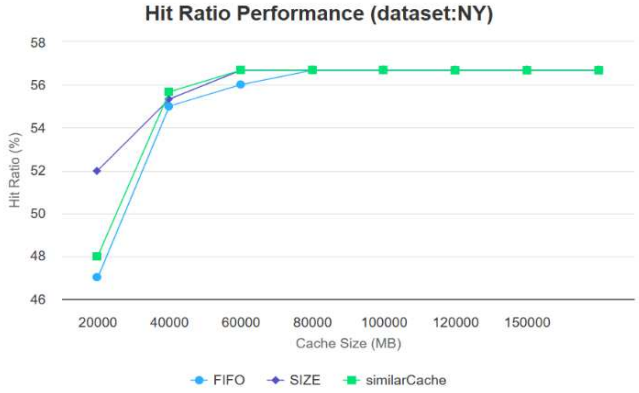

Fig. 7 HR Performance Using the NY Dataset

B. Discussion

Based on the simulation results from the four IRcache datasets, the proposed similarCache framework presents a distinct approach to handling each access to the application database. In the current era of artificial intelligence and large datasets, we recognize the growing need for information systems to adopt more comprehensive considerations before making decisions. Web-based applications that continue to rely on RDBMS benefit from a dependable intermediary to provide swift responses to frequently recurring access requests for the same popular data. As a result, research in application-level caching at the application level remains an open and evolving field.

Our extensive literature review reveals a multitude of studies in application-level caching, each offering unique perspectives and valuable insights.

Blankstein et al. proposed the hyperbolic caching framework, which employs a simple calculation concept $p_i = \frac{n_i}{t_i}$ for caching decisions. However, this approach has a notable limitation—it does not consider data size, potentially diminishing the prospects of enhancing hit ratio performance for small-sized data. Another framework, APLcache, was introduced by Mertz et al., featuring the development of both proactive and reactive caching components, which adds significant complexity to the caching decision-making process. The observed hit ratio performance improvement, approximately 2.78%, appears disproportionate when weighed against the intricacies of the caching system it proposes. Even our earlier research on the LRU-GENACO framework demonstrated algorithmic complexity that did not yield commensurate hit ratio performance improvements, merely achieving a 1% increase.

We acknowledge that there is inevitably a trade-off in the objectives of the caching system being constructed. The choice may involve improving the hit ratio at the expense of increased computational load or prioritizing energy efficiency in the server environment, which could lead to a decrease in hit ratio performance or response time due to constrained resources. However, the performance of the similarCache framework concept is less beneficial in cases where there is a significant increase in access to specific cached material, such as during a viral phenomenon. In general, cache servers prioritize caching material that is frequently accessed, whereas similarCache aims to proactively address this issue to avoid cache pollution.

The subsequent similarCache will be designed to adapt to changing data access patterns. The three datasets utilized in the preceding simulation exhibit common data access patterns. However, due to the need for improved performance at smaller cache sizes, the access patterns on the SV dataset vary considerably for the proposed similarCache. We began to develop the idea that, in the event of an access abnormality, a proposed similarCache can efficiently ascertain the distribution of data access patterns by employing standard deviation calculations. The suggested similarCache should have the capability to increase the Pearson coefficient for data that has a large number of visits, particularly for top-accessed data, in the case of an anomaly.

IV. Conclusion

This study presents a web-based application-level caching solution that employs method calls to link controller accesses with database query results. This paper has the main contribution of the similarCache framework employs the Pearson correlation coefficient to prioritize the replacement of data in the server cache. The performance of the system is evaluated by analyzing the hit ratio using IRcache datasets. The simulation findings indicate that the data access patterns, and cache size significantly influence the hit ratio performance. Although the store capacity is restricted, similarCache enhances the efficiency of cache memory consumption. Simulations on the SV dataset demonstrate that the proposed similarCache outperforms the commonly used SIZE and FIFO techniques by 2.33% and 1%, respectively. The future work involves developing a comparable cache that can adapt to changes in data access patterns by calculating the present standard deviation. If an abnormality occurs, the proposed similarCache should have the capability to elevate the Pearson coefficient for data that is frequently accessed to the same level as data that is accessed the most.

Acknowledgment

This research was funded by the Indonesian Ministry of Education, Culture, Research and Technology through the DRTPM research with number 2529/UN23/PT.01.02/2023.

References

[1] T. Connolly and C. Begg, *Database Systems : A Practical Approach to Design Implementation, and Management*. Pearson, 2014.

[2] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*. McGraw-Hill, 2010.

[3] C. J. Date, *An Introduction to Database Systems*. Pearson, 2004.

[4] M. Indrawan-Santiago, "Database Research: Are We at a Crossroad? Reflection on NoSQL," 2012 15th International Conference on Network-Based Information Systems, Sep. 2012, doi: 10.1109/nbis.2012.95.

[5] G. Karnitis and G. Arnicans, "Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation," 2015 7th International Conference on Computational Intelligence, Communication Systems and Networks, Jun. 2015, doi: 10.1109/cicsyn.2015.30.

[6] M. I. Zulfa, R. Hartanto, and A. E. Permanasari, "Caching strategy for Web application – a systematic literature review," International Journal of Web Information Systems, vol. 16, no. 5, pp. 545–569, Oct. 2020, doi: 10.1108/ijwis-06-2020-0032.

[7] W. Vogels, "Scaling Amazon ElastiCache for Redis with Online Cluster Resizing."

[8] I. Amazon Web Services, "Use Cases and How ElastiCache Can Help."

[9] K. Kaur, R. Rani, C. Sci, and E. Deptt, "Modeling and Querying Data in NoSQL Databases," in *IEEE International Conference on Big Data*, 2013.

[10] H. K. Lee, B. S. An, and E. J. Kim, "Adaptive Prefetching Scheme Using Web Log Mining in Cluster-Based Web Systems," 2009 IEEE International Conference on Web Services, Jul. 2009, doi: 10.1109/icws.2009.127.

[11] M. Kusuma, Widyawan, and R. Ferdiana, "Performance comparison of caching strategy on wordpress multisite," 2017 3rd International Conference on Science and Technology - Computer (ICST), Jul. 2017, doi: 10.1109/icstc.2017.8011874.

[12] W. Puangsaijai and Sutheera Puntheeranurak, "A Comparative Study of Relational Database and Key-Value Database for Big Data Applications," in *International Electrical Engineering Congress*, 2017, pp. 8–10.

[13] D. J. Carlson, *Ebook Redis in Action*. Manning Publications, 2013.

[14] S. Bouchenak, A. Cox, S. Dropsho, S. Mittal, and W. Zwaenepoel, "Caching Dynamic Web Content: Designing and Analysing an Aspect-Oriented Solution," Middleware 2006, pp. 1–21, 2006, doi: 10.1007/11925071_1.

[15] Y. K. Alae El Alami, Mohamed Bahaj, "Supply of a Key Value Database Redis In-Memory by Data from a Relational Database," in *IEEE Mediterranean Electrotechnical Conference*, IEEE, 2018, pp. 46–51.

[16] A. E. Lotfy, A. I. Saleh, H. A. El-Ghareeb, and H. A. Ali, "A middle layer solution to support ACID properties for NoSQL databases," Journal of King Saud University - Computer and Information Sciences, vol. 28, no. 1, pp. 133–145, Jan. 2016, doi: 10.1016/j.jksuci.2015.05.003.

[17] J. Shamsi, M. A. Khojaye, and M. A. Qasmi, "Data-Intensive Cloud Computing: Requirements, Expectations, Challenges, and Solutions," Journal of Grid Computing, vol. 11, no. 2, pp. 281–310, Apr. 2013, doi: 10.1007/s10723-013-9255-6.

[18] J. Baker *et al.*, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services," in *Proceedings of the Conference on Innovative Data system Research (CIDR)*, 2011.

[19] D. Akbari Bengar, A. Ebrahimnejad, H. Motameni, and M. Golsorkhtabaramiri, "A page replacement algorithm based on a fuzzy approach to improve cache memory performance," Soft Computing, vol. 24, no. 2, pp. 955–963, Dec. 2019, doi: 10.1007/s00500-019-04624-w.

[20] J. Mertz and I. Nunes, "Automation of application-level caching in a seamless way," Software: Practice and Experience, vol. 48, no. 6, pp. 1218–1237, Feb. 2018, doi: 10.1002/spe.2571.

[21] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "Intelligent Web proxy caching approaches based on machine learning techniques," Decision Support Systems, vol. 53, no. 3, pp. 565–579, Jun. 2012, doi: 10.1016/j.dss.2012.04.011.

[22] R. Meloca and I. Nunes, "A comparative study of application-level caching recommendations at the method level," Empirical Software Engineering, vol. 27, no. 4, Apr. 2022, doi: 10.1007/s10664-021-10089-z.

[23] V. Holmqvist and J. Nilsfors, "Cachematic – Automatic Invalidation in Application-Level Caching Systems," in *International Conference on Performance Engineering*, 2019, pp. 167–178.

[24] A. Blankstein, S. Sen, M. J. Freedman, A. Blankstein, S. Sen, and M. J. Freedman, "Hyperbolic Caching : Flexible Caching for Web Applications Hyperbolic Caching : Flexible Caching for Web Applications," in *Proceedings of the 2017 USENIX Annual Technical Conference*, 2017.

[25] J. Mertz and I. Nunes, "Understanding Application-Level Caching in Web Applications," ACM Computing Surveys, vol. 50, no. 6, pp. 1–34, Nov. 2017, doi: 10.1145/3145813.

[26] J. Mertz, I. Nunes, L. Della Toffola, M. Selakovic, and M. Pradel, "Satisfying Increasing Performance Requirements With Caching at the Application Level," IEEE Software, vol. 38, no. 3, pp. 87–95, May 2021, doi: 10.1109/ms.2020.3033508.

[27] T. Ma, Y. Hao, W. Shen, Y. Tian, and M. Al-Rodhaan, "An Improved Web Cache Replacement Algorithm Based on Weighting and Cost," IEEE Access, vol. 6, pp. 27010–27017, 2018, doi: 10.1109/access.2018.2829142.

[28] T. Ma, J. Qu, W. Shen, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "Weighted Greedy Dual Size Frequency Based Caching Replacement Algorithm," IEEE Access, vol. 6, pp. 7214–7223, 2018, doi: 10.1109/access.2018.2790381.

[29] J. Zhang, "Replacement Strategy of Web Cache Based on Data Mining," 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Nov. 2015, doi: 10.1109/3pgcic.2015.75.

[30] A. Blankstein, S. Sen, M. J. Freedman, A. Blankstein, S. Sen, and M. J. Freedman, "Hyperbolic Caching: Flexible Caching for Web Applications," in *Proceedings of the 2017 USENIX Annual Technical Conference*, USENIX Annual Technical Conference, 2017. doi: 10.5555/3154690.3154738.

[31] J. Mertz and I. Nunes, "A Qualitative Study of Application-Level Caching," IEEE Transactions on Software Engineering, vol. 43, no. 9, pp. 798–816, Sep. 2017, doi: 10.1109/tse.2016.2633992.

[32] M. I. Zulfa, R. Hartanto, A. E. Permanasari, and W. Ali, "LRU-GENACO: A Hybrid Cached Data Optimization Based on the Least Used Method Improved Using Ant Colony and Genetic Algorithms," Electronics, vol. 11, no. 19, p. 2978, Sep. 2022, doi: 10.3390/electronics11192978.

[33] J. Thomas, "Are ASEAN's internet speeds world class?," The Asean Post.

[34] A. Saverimoutou, B. Mathieu, and S. Vaton, "Influence of Internet Protocols and CDN on Web Browsing," 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Jun. 2019, doi: 10.1109/ntms.2019.8763827.

[35] D. Ayuba, A. Ismail, and M. I. Hamzah, "Evaluation of Page Response Time between Partial and Full Rendering in a Web-based Catalog System," Procedia Technology, vol. 11, pp. 807–814, 2013, doi: 10.1016/j.protcy.2013.12.262.

[36] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/jiot.2016.2579198.

[37] S. Motaman, S. Ghosh, and N. Rathi, "Cache Bypassing and Checkpointing to Circumvent Data Security Attacks on STTRAM," IEEE Transactions on Emerging Topics in Computing, vol. 7, no. 2, pp. 262–270, Apr. 2019, doi: 10.1109/tetc.2017.2653813.

[38] M. I. Zulfa, A. Fadli, A. E. Permanasari, and W. A. Ahmed, "Performance comparison of cache replacement algorithms onvarious internet traffic," JURNAL INFOTEL, vol. 15, no. 1, pp. 1–7, Feb. 2023, doi: 10.20895/infotel.v15i1.872.

[39] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "A Survey of Web Caching and Prefetching," *Int. J. Adv. Soft Comput. Appl.*, vol. 3, no. 1, pp. 1–27, 2011.

[40] X. Li, X. Wang, Z. Sheng, H. Zhou, and V. C. M. Leung, "Resource allocation for cache-enabled cloud-based small cell networks," Computer Communications, vol. 127, pp. 20–29, Sep. 2018, doi: 10.1016/j.comcom.2018.05.007.

[41] T. Chen, "Obtaining the optimal cache document replacement policy for the caching system of an EC website," European Journal of Operational Research, vol. 181, no. 2, pp. 828–841, Sep. 2007, doi: 10.1016/j.ejor.2006.05.034.

[42] S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," ACM Computing Surveys, vol. 35, no. 4, pp. 374–398, Dec. 2003, doi: 10.1145/954339.954341.