Jurnal Teknik Informatika (JUTIF)
Vol. 4, No. 4, August 2023, pp. 875-881

DOI: https://doi.org/10.52436/1.jutif.2023.4.4.1244
p-ISSN: 2723-3863
e-ISSN: 2723-3871

CACHE DATA REPLACEMENT POLICY BASED ON RECENTLY USED ACCESS DATA AND EUCLIDEAN DISTANCE

Mulki Indana Zulfa[*1], Muhammad Syaiful Aliim[1], Ari Fadli[1,2], Waleed Ali[3]

[1,2]Electrical Engineering Department, Engineering Faculty, Universitas Jenderal Soedirman, Indonesia
[3]Computer Science Department, Mathematics and Natural Sciences Faculty, Universitas Gadjah Mada, Indonesia
[4]Information Technology Department, King Abdulaziz University, Saudi Arabia
Email: [1]mulki_indanazulfa@unsoed.ac.id, [2]muhammad.syaiful.aliim@unsoed.ac.id, [3]arifadli@mail.ugm.ac.id, [4]waabdullah@kau.edu.sa

(Article received: July 25, 2023; Revision: August 8, 2023; published: August 21, 2023)

Abstract

Data access management in web-based applications that use relational databases must be well thought out because the data continues to grow every day. The Relational Database Management System (RDBMS) has a relatively slow access speed because the data is stored on disk. This causes problems with decreased database server performance and slow response times. One strategy to overcome this is to implement caching at the application level. This paper proposed SIMGD framework that models Application Level Caching (ALC) to speed up relational data access in web applications. The ALC strategy maps each controller and model that has access to the database into a node-data in the in-Memory Database (IMDB). Not all node-data can be included in IMDB due to limited capacity. Therefore, the SIMGD framework uses the Euclidean distance calculation method for each node-data with its top access data as a cache replacement policy. Node-data with Euclidean distance closer to their top access data have a high priority to be maintained in the caching server. Simulation results show at the 25KB cache configuration, the SIMGD framework excels in achieving hit ratios compared to the LRU algorithm of 6.46% and 6.01%, respectively.

Keywords: *application-level caching, in memory database, hit ratio, relational database, web application.*

1. INTRODUCTION

The presence of internet technology has encouraged the emergence of various supporting technologies such as access media and internet-based devices [1]. In addition, the ease of internet access, both through fixed line networks and mobile networks, has contributed to increasing world internet penetration from year to year. Internet users worldwide reach 3.8 billion users from the entire human population which reaches 7,497 billion. This increase shows that internet users reach 51% of the total world population [2] [3]. Meanwhile, the internet penetration rate in Indonesia reached 73.7% of the total population in early 2022. The total population of Indonesia was recorded at 277.7 million people in January 2022 [4].

Internet infrastructure has an important role in national development. It is an enabling tool for economic growth and increased productivity [5]. Response time on a website is very important because it can increase user satisfaction, retention, and productivity [6]. A study stated that if the website takes more than eight seconds to load the page, consumers are much more annoyed and leave the portal [6].

In general, web pages are divided into two, namely static web pages and dynamic web pages. The static web is only built from a few HTML objects while dynamic web requires access to one or more databases to store and display the desired data. Response time, or in other research referred to as speed, is the time needed to measure how quickly a web page is fully displayed in the browser since it was first clicked [7]. Response time is a problem that must be considered by web application developers to reduce user work time to be faster to improve the experience and comfort of web users [8].

Currently, there are still many website applications that use the Relational Database Management System (RDBMS) as the main storage medium because RDBMS can store structured data or relational data properly [9]. RDBMS has a slow access speed because the data is stored on the hard drive [10]. RDBMS performance will be slower as the number of join query executions increases.

On the other hand, data storage technology based on In-Memory Databases (IMDB) is developing rapidly. This technology is widely used by cloud service providers such as Amazon Web Services, Google Cloud Platform, IBM, and Microsoft Azure [11]. IMDB no longer stores data on the hard disk but in computer memory so it has a

much better access speed [12]. IMDB, also known as NoSQL database [13], can effectively reduce server workload and reduce internet latency [14]. However, IMDB has a weakness, namely, it does not provide data guarantees on Atomicity Consistency, Isolation, and Durability so that IMDB cannot replace RDBMS. Therefore, the idea of combining the two IMDB and RDBMS database technologies can be used to support data transactions that guarantee data availability and consistency but still offer high access speed performance [15].

The exponential growth in the amount of data and the need to serve data requests quickly in web applications in the big data era cannot be faced by RDBMS [16]. One solution to overcome this problem is to implement web caching. Web caching is proven to be able to reduce bandwidth, latency, and server load [17]. Web caching strategies can take advantage of IMDB which helps RDBMS performance. This study proposes a proof of concept application-level caching for web-based applications that still use RDBMS as their main database.

2. RESEARCH METHOD

Figure 1 shows the research step began with a literature study on application level caching models using method calls and query parsers that are explained more detail in the section 2.3. The next research step is to prepare research datasets and a virtual private server (VPS) as a simulation tool. This study uses the IRcache dataset provided by NLANR which contains data access logs from proxy servers located in the United States.

After all the research tools and materials are ready, the next step is to start designing the proposed SIMGD framework that calculates all cache data node distances to the data center (top access data). The next step is to test the SIMGD framework by calculating the hit ratio and then writing the results.

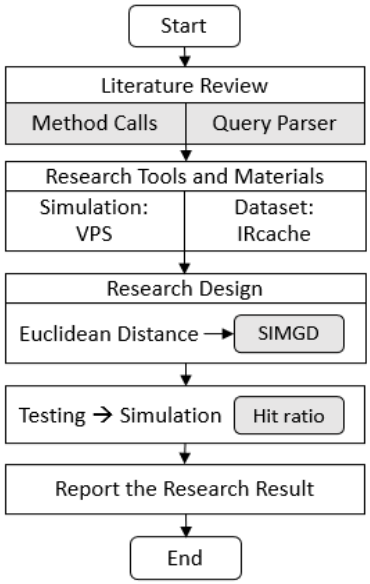

Figure 1. Research methods.

2.1 Dataset

This research was conducted utilizing a simulation using the IRcache dataset which contains a track record of internet access managed by a proxy server located in the United States. The IRcache dataset was first released in 1999 by Alex Rousskov which was then managed by the National Lab of Applied Network Research (NLANR) [18].

Based on a literature study conducted, the IRcache dataset was still accessible in 2013 but after that, this dataset could no longer be freely accessed [19] [20]. However, in the last five years, the IRcache dataset is still being used by Ibrahim et al. in cache replacement studies [21] and Li et al. in content caching optimization research [19]. The IRcache dataset is suitable for use in caching strategy research that measures hit ratio performance because this dataset provides complete, natural, and dynamic access logs for real-world applications. Application-level caching simulation is carried out using a look-aside caching topology.

2.2 Look-aside caching

The look-aside caching topology works by always directing data access requests to the caching server first. If the requested data is not found, the data request is forwarded directly to the relational database used, then stores duplicate data on the caching server. Figure 2 shows the mechanism of look-aside caching.

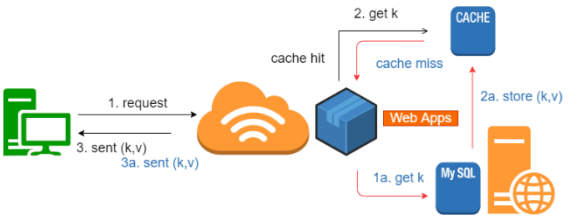

Figure 2. *look-aside caching*.

2.3 The Proposed SIMGD Framework

This study proposes a SIMGD framework that models application level caching in web-based applications. The SIMGD framework adopts the Query Parser [22] and Method Calls [23] methods to map each Controller-Model that accesses a relational database. The query syntax written in the Model will be separated based on syntax: where, join, having, group by, order by, as well as other SQL functions such as sum(), avg(), min(), max() and other functions using the query parser, as shown in Algorithm 1.

This research requires reading access log files from the IRcache dataset and then carrying out the preprocessing process, namely separating each part of the controller, model, and arguments based on a known base url() as shown in Figure 3. The Query Parser [22] and Method Calls [23] produce the expensiveness variable values which are used together with the frequency and timestamp variables

as the basis for calculating the Euclidean distance for each node-data cache.

Algorithm 1: Algoritme parseQuery(q)

```
Input: node,q_cost
Output: Vexpv

k = [join, where,group by,having,
distinct, order by, and any SQL function]
q_cost = showstatus(Q)
qnode ← QueryNode(q, qmeta, qhash)

foreach qnode do
  kla ← pregmatch(k,qnode)
  switch(kla)
    case k:
      k_cost = k_cost+1
  end
end

Vexpv = k_cost/q_cost
```

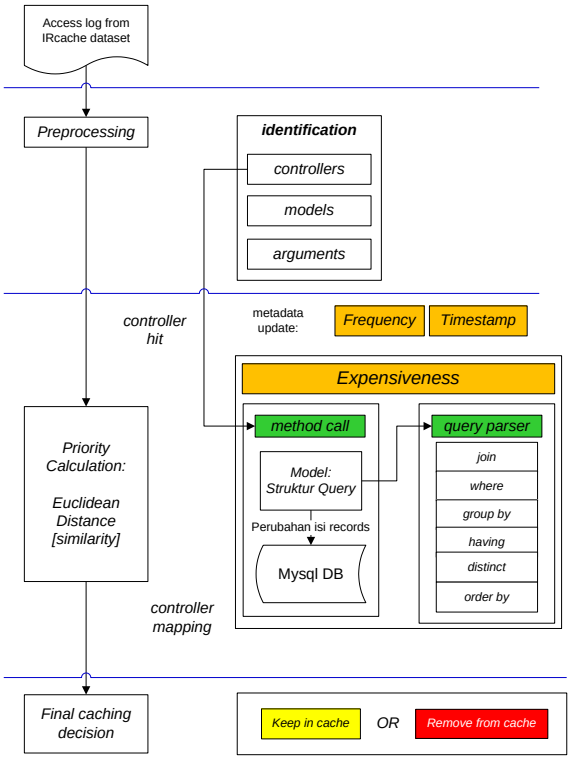

Figure 3. Method call and query parser.

2.4 Euclidean Distance dan Hit Ratio

The Euclidean distance formula basically used the concept of calculating the similarity between two data. The similarity method is used to assess data priority based on the proximity of each data node to the top access data on the same Controller-Model [24]. Controller design and models in the MVC web-based application framework play an important role in the data access route requested by the user. At the beginning of the semester access to controller:Student/StudyPlan can be the busiest but towards the end of the semester controller:StudyResult becomes the most accessed. The farther a node-data (similarity: low) is from the top access data that exists at that time, the faster it is likely to be removed from the caching server.

Top access data may change according to the current trend. Therefore it becomes the basis that the data nodes in the caching server can also change according to the data access conditions. The caching server utility must be able to maximally maintain the data nodes that are most frequently accessed to remain in the caching server so as to increase hit ratio performance. The greater the hit ratio, the faster the response time perceived by the user. Equation (1) is used to calculate the hit ratio by dividing all available data access (N) divided by the number of data services successfully provided from the caching server (r_i).

$$HR = \frac{\sum_{i=1}^{N} r_i}{N} \quad (1)$$

2.5 Node-data Replacement Policy

Not all data nodes can be entered into the caching server because of its very limited capacity. If there is a need to store new node-data on the caching server, then the node-data replacement mechanism must be implemented. The way to do this is to delete one or several data nodes that are already on the caching server until sufficient capacity is available to store the new data nodes.

The proposed SIMGD framework execute node-data replacement mechanism by removing the node-data with the largest euclidian distance from the top access data. Equation (2) is used to calculate the euclidean distance on n-dimensional data nodes with their top access data. The dimensions of the node-data in question are: access count (frequency), recently access (time stamp), and the expensiveness variable from the results of the query parser method. Other application architectures can use Equation (3) to define the expensiveness variable. Equation (3) adopts the key-value formula from the Greedy-Dual Frequency Size (GDSF) algorithm which considers the access frequency F_((g)), cost C_((g)), and size S_((g)) which can represent the value of expensiveness K_((g)).

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \quad (2)$$

$$K_{(g)} = L + F_{(g)} * \frac{C_{(g)}}{S_{(g)}} \quad (3)$$

3. RESULTS AND DISCUSSION

3.1 Prepocessing Result

Table 1 presents two examples of an IRcache raw dataset consisting of 8 properties. Based on the data sample from Table 1, the preprocessing results from the Query Parser and Method Calls are presented in Table 2. Basically, IRcache raw data can

contain the property: "Elapsed", but based on our dataset survey, the value of this property is inconsistent. Therefore, the Query Parser method can be a solution for calculating the value of this property: "Elapsed".

Table 1. The IRcache raw data example.

No	Properties	Example 1	Example 2
1	iddata	4242106418	434356357
2	time	1282592384.330	1282654695.614
3	ipclient	190.151.117.59	161.242.106.167
4	code	TCP_HIT/200	TCP_HIT/200
5	size	465	559
6	method	GET	GET
7	URL	http://1stnatbk.com/images/2236int_r13_c1.gif	http://1stnatbk.com/images/FOM/fom_Pers_savings_f2.gif
8	type	image/gif	image/gif

Table 2. The preprocessing result.

No	Properties	Example-1	Example-2
1	Elapsed	11	44
2	Timestamp	1282592384.330	1282654695.614
3	Controller-Model	/images/	/images/FOM/
4	MVC access	1	2
5	Size	465	559
6	Access count	10	15

3.2 Euclidean Distance to Top Access Data

Each value in Table 2 can be updated according to the real data access conditions. In the example of Table 2, the value of properties: MVC and access count can continue to increase along with improvements to the Controller-Model. In addition to these two examples of data, there are many other data that have different property values. When there is a need to store new data but the caching server is full, the replacement policy mechanism must work.

The proposed SIMGD framework will determine the top access data based on the most recently used data. This needs to be done to determine which data node should be removed first if the caching server is full. The node-data caching storage structure follows the array stack management, where the most recently accessed data will occupy the top position. Therefore, the data node that will be removed first is the data with the furthest distance from the top access data.

Table. 3 Top access data calculation results.

iddata URL	size	fx	mvc	timestamp	istop
494998995	450	2.9981	1	1690699669	yes
454513888	344	1.4817	1	1690699668	no
474727860	571	2.0379	1	1690699668	no
42424182	588	1.6723	1	1690699668	no
41414189	590	1.0961	1	1690699669	no
4242114478	510	1.1139	1	1690699669	no

Table 3 illustrates the results of calculating the Euclidean distance for each data-node with its top data access. Top access data is marked with column istop='yes'. If the caching server has to perform a cache replacement mechanism, then each node-data istop='no' will calculate its euclidean distance to the top access data (istop='yes') which is stored in column fx. The data node with the largest fx value will be removed first from the caching server indicating that the data node has the furthest distance from the top access data.

3.3 Hit Ratio Performance

Caching strategy researchers generally benchmark hit ratio performance with several conventional caching algorithms such as Least Recently Used (LRU), Least Frequently Used LFU, and Greedy-Dual Size (GDS). Each of these caching algorithms has a different mechanism for making caching decisions. The LFU algorithm gives higher priority to data nodes with a high number of accesses. The GDS algorithm prefers small data to be stored on the caching server. Meanwhile, the LRU algorithm will always place recently node-data at the top of the caching server. So the process of deleting data nodes in the LRU algorithm starts from the bottom of the data nodes in the caching server. Based on the literature studies that have been conducted, some of these caching algorithms can excel in certain case studies but can also lose in other cases.

The benchmarking process is also carried out to prepare several caching server capacity configurations, starting from small to large cache capacities. This study uses 8 caching server capacity configurations, with the smallest size being 9 KB while the largest cache size is 150 KB. This cache size can be adjusted to the actual cache storage conditions, even up to hundreds of terabytes. However, due to limited research time, the illustration of cache capacity in this simulation scenario is made quite small.

Based on Figure 4, the hit ratio performance of the proposed SIMGD method is quite competitive with the hit ratio performance of the best LRU conventional caching method. At a cache capacity of 25KB, the proposed SIMGD method managed to outperform the LRU hit ratio performance of 6.46% to 6.01%, respectively. Figure 5 also shows that the proposed SIMGD method excels in many cache capacity configurations, namely 9 KB, 12 KB, 15 KB, 25 KB, 50 KB, 90 KB from the performance hit ratio algorithm based on access count (LFU) and data size (GDS).

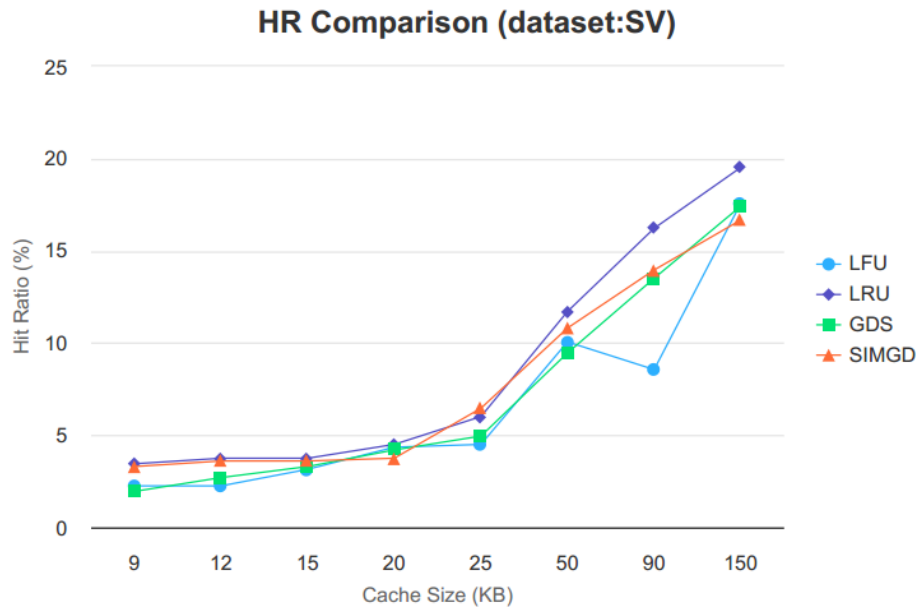

Figure 4. Hit ratio performance on SV dataset.

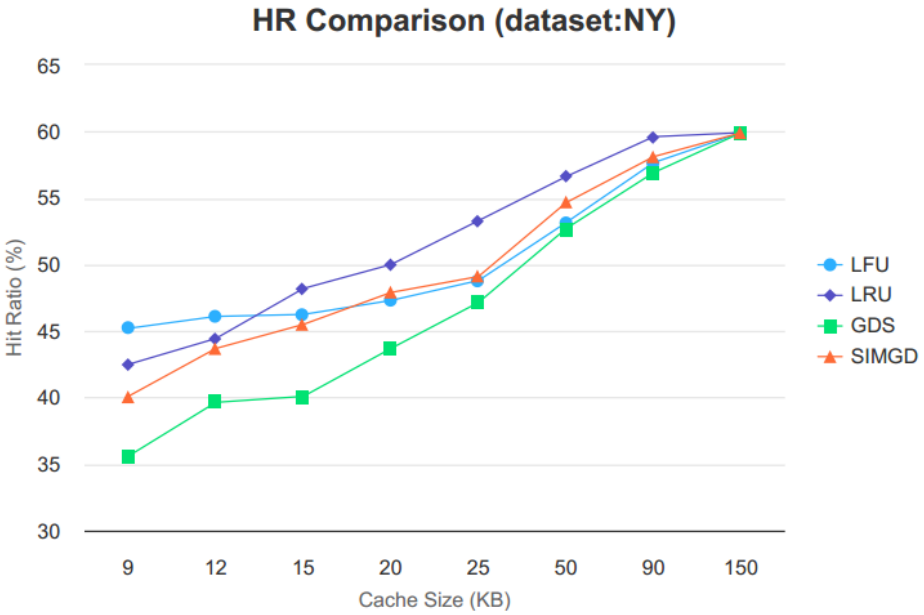

Figure 5. Hit ratio performance on NY dataset.

Figure 5 shows the hit ratio performance of the proposed SIMGD method compared to the LFU, LRU, and GD algorithms. At a small cache size, 9KB, the LFU algorithm is the most superior with a performance hit ratio of 45.2%. But as cache allocation increases, LFU performance decreases. The simulation results show that SIMGD performance starts to increase since the cache capacity is 25 KB. At a cache allocation of 20 KB-150 KB, the proposed SIMGD method starts to be the second best after the LRU hit ratio performance.

In general, the hit ratio performance is strongly influenced by the characteristics of data access and cache allocation provided. The greater the cache capacity, the greater the resulting hit ratio performance. But the decision to add or increase cache capacity is not a good decision considering the price of memory storage is still expensive. Therefore, the manager of the cloud server network infrastructure really hopes for this caching method.

Based on the simulation results in Figures 4 and 5, the IRcache SV and NY datasets have different data access characteristics. This can be seen from the performance range of the hit ratio which is quite far from the two datasets. Simulations on the SV dataset show that the maximum hit ratio performance is only at a value of 19.52% which is achieved by the LRU algorithm. Meanwhile, the maximum hit ratio performance in the NY dataset simulation reached 59.91% which was achieved by the proposed SIMGD method and three other caching algorithms.

The IRcache SV dataset is very similar to the viral phenomenon which only accesses certain data nodes until these data nodes experience a significant increase in the number of accesses. Therefore the performance of the data replacement caching algorithm on the SV dataset is not optimal because almost all the data requested by the user is already in the caching server. The data access pattern on the SV dataset is very beneficial for recently used algorithms such as LRU.

4. DISCUSSION

Web caching research is still a research topic that continues to be developed. This study continues the results of previous research LRU-GENACO [25] which combines the metaheuristic Ant colony algorithm (ACO) and Genetic algorithm (GA) to optimize web caching. However, this method experienced a decrease in hit ratio performance in the NY dataset. Therefore, this study proposes another point of view on the node-replacement mechanism by calculating the distance of each node-data with its top data access.

The data access top acts like a centroid in the clustering concept. Based on the evaluation of our previous research [26], the LRU algorithm is the best in achieving hit ratios. Therefore, the top data access that we choose is based on the data node that was recently accessed (recently used). If there is a need to store new node-data with the caching server condition being full, then the node-data deletion process starts from the node-data with the farthest distance from the top data access. This concept imitates the way clustering works with the assessment that the closer a data is to the data center, the data is said to be more similar (high level of similarity). Data nodes with a high level of similarity with top data access are more feasible to maintain in the caching server so that they are expected to be able to increase hit ratio performance. The higher the hit ratio, the faster the response felt by the user. This is the main goal of web caching strategy.

5. CONCLUSION

The hit ratio performance is highly dependent on the characteristics of the data access pattern and the caching server capacity allocated. The larger the capacity of the caching server, the greater the hit ratio that can be achieved. However, with a very limited caching server capacity, the caching algorithm will become the main focus for maximizing the caching server utility. This study proposes a SIMGD caching method that changes the contents of the caching server based on the proximity of the node-data distance to the top data access at that time. The simulation results show that the hit ratio performance of the SIMGD method is able to compete with the best LRU caching algorithm. In the IRcache SV dataset with a cache size of 25KB, the proposed SIMGD method outperforms the LRU hit ratio by 6.46% and 6.01%, respectively.

DAFTAR PUSTAKA

[1] F. S. Rahayu, L. E. Nugroho, R. Ferdiana, and D. B. Setyohadi, "Research Trend on the Use of IT in Digital Addiction: An Investigation Using a Systematic Literature Review," *Futur. Internet*, vol. 12, no. 10, p. 174, Oct. 2020, doi: 10.3390/fi12100174.

[2] M. D. Ariani, "Peran Kesepian dan Pengungkapan Diri Online Terhadap Kecanduan Internet pada Remaja Akhir," UNISULA, 2018.

[3] F. Sapty Rahayu, L. Kristiani, and S. Fuhrensia Wersemetawar, "Dampak Media Sosial terhadap Perilaku Sosial Remaja di Kabupaten Sleman, Yogyakarta," in *Seminar Nasional Inovasi Teknolog UN PGRI Kediri*, 2019, vol. 2018, pp. 39–46, [Online]. Available: https://proceeding.unpkediri.ac.id/index.php/inotek/article/download/511/423/1241.

[4] M. Situmorang, "Measuring The Effectiveess of Consumer Dispute Resolution on Small Value E-Commerce Transaction," *J. Penelit. Huk. Jure*, vol. 22, no. 4, p. 537, Dec. 2022, doi: 10.30641/dejure.2022.V22.537-550.

[5] R. A. Wahab, "Comparative Analysis of Broadband Internet Development for Digital Economy in China and Indonesia," *J. Penelit. Pos dan Inform.*, vol. 9, no. 1, pp. 63–80, Oct. 2019, doi: 10.17933/jppi.v9i1.274.

[6] Sathiyamoorthi V., Suresh P., Jayapandian N., Kanmani P., Deva Priya M., and S. Janakiraman, "An Intelligent Web Caching System for Improving the Performance of a Web-Based Information Retrieval System," *Int. J. Semant. Web Inf. Syst.*, vol. 16, no. 4, pp. 26–44, Oct. 2020, doi: 10.4018/IJSWIS.2020100102.

[7] A. Gasparyan, "Most Important Metrics for Your Website Performance," *Monitis*. 2019, Accessed: Oct. 26, 2019. [Online]. Available: https://www.monitis.com/blog/most-important-metrics-for-your-website-performance/.

[8] R. Ramakrishnan and A. Kaur, "Performance evaluation of web service response time probability distribution models for business process cycle time simulation," *J. Syst. Softw.*, vol. 161, p. 110480, Mar. 2020, doi: 10.1016/j.jss.2019.110480.

[9] J. M. Medina, C. D. Barranco, and O. Pons, "Indexing techniques to improve the performance of necessity-based fuzzy queries using classical indexing of RDBMS," in *Fuzzy Sets and Systems*, Nov. 2018, vol. 351,

pp. 90–107, doi: 10.1016/j.fss.2017.09.008.

[10] M. Luthfi, M. Data, and W. Yahya, "Perbandingan Performa Reverse Proxy Caching Nginx dan Varnish Pada Web Server Apache," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 2, no. 4, pp. 1457–1463, 2018.

[11] I. Amazon Web Services, "Use Cases and How ElastiCache Can Help," 2019. https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/elasticache-use-cases.html#elasticache-use-cases-data-store (accessed Mar. 22, 2019).

[12] N. Roy-Hubara, A. Sturm, and P. Shoval, "Designing NoSQL databases based on multiple requirement views," *Data Knowl. Eng.*, vol. 145, no. January, p. 102149, May 2023, doi: 10.1016/j.datak.2023.102149.

[13] 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, no. March, pp. 8–10.

[14] M. I. Zulfa, R. Hartanto, and A. E. Permanasari, "Caching strategy for Web application – a systematic literature review," *Int. J. Web Inf. Syst.*, vol. 16, no. 5, pp. 545–569, Oct. 2020, doi: 10.1108/IJWIS-06-2020-0032.

[15] S. Pendse *et al.*, "Oracle Database In-Memory on Active Data Guard: Real-time Analytics on a Standby Database," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, Apr. 2020, vol. 2020-April, pp. 1570–1578, doi: 10.1109/ICDE48307.2020.00139.

[16] G. Wu *et al.*, "Cracking in-memory database index: A case study for Adaptive Radix Tree index," *Inf. Syst.*, vol. 104, p. 101913, Feb. 2022, doi: 10.1016/j.is.2021.101913.

[17] C. Guerrero, I. Lera, and C. Juiz, "Performance improvement of web caching in Web 2.0 via knowledge discovery," *J. Syst. Softw.*, vol. 86, no. 12, pp. 2970–2980, 2013, doi: 10.1016/j.jss.2013.04.060.

[18] NLANR, "The National Laboratory for Applied Network Research (NLANR)," 2001. http://www.nlanr.net/ (accessed Jul. 03, 2020).

[19] X. Li, X. Wang, Z. Sheng, H. Zhou, and V. C. M. Leung, "Resource allocation for cache-enabled cloud-based small cell networks," *Comput. Commun.*, vol. 127, no. April, pp. 20–29, Sep. 2018, doi: 10.1016/j.comcom.2018.05.007.

[20] Z. Zali, E. Aslanian, M. H. Manshaei, M. R. Hashemi, and T. Turletti, "Peer-Assisted Information-Centric Network (PICN): A Backward Compatible Solution," *IEEE Access*, vol. 5, pp. 25005–25020, 2017, doi: 10.1109/ACCESS.2017.2762697.

[21] H. Ibrahim, W. Yasin, N. I. Udzir, and B. Process, "Intelligent cooperative web caching policies for media objects based on J48 decision tree and Naïve Bayes supervised machine learning algorithms in structured peer-to-peer systems," *J. Inf. Commun. Technol.*, vol. 15, no. 2, pp. 85–116, 2016.

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

[23] J. Mertz and I. Nunes, "Automation of application-level caching in a seamless way," *Softw. Pract. Exp.*, vol. 48, no. 6, pp. 1218–1237, Jun. 2018, doi: 10.1002/spe.2571.

[24] D. Zhang, Y. Liu, A. Liu, X. Mao, and Q. Li, "Efficient Path Query Processing Through Cloud-Based Mapping Services," *IEEE Access*, vol. 5, pp. 12963–12973, 2017, doi: 10.1109/ACCESS.2017.2725308.

[25] 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.

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