Enhancing Database Management System Performance Using Data Mining Techniques

RAJEEV MAHESHWARI[1], RAJEEV KAUSHIK[2]
[1, 2]*Deptt. of Computer Science & Engineering, R.D Engineering College Ghaziabad Uttar Pradesh India*

Abstract- Cloud computing has gained significant traction due to its cost-effectiveness, scalability, and flexible pay-as-you-go model—trends further accelerated by the proliferation of IoT devices. As a result, many organizations now leverage Database-as-a-Service (DBaaS) for database deployment, enjoying benefits such as high availability, automated scaling, failover support, and reduced administrative overhead. This paper enhances the performance of shared cloud databases under mixed transactional and analytical workloads through two complementary strategies: Business intelligence and decision support systems often execute complex queries with multiple joins and aggregations. To minimize repeated computations, a scalable tree-mining algorithm is employed to identify frequently occurring sub-expressions from historical query plans. These are materialized as views, significantly lowering query execution costs. Web applications like e-commerce and online banking experience region-specific, time-dependent access patterns. To address this, a predictive model based on Parzen window estimation is used to identify time-varying working sets. A novel cache replacement strategy is then proposed, prioritizing blocks based on predicted reuse. Experimental evaluations demonstrate that the proposed methods significantly improve cache hit rates and overall performance compared to existing solutions.

I. INTRODUCTION

Today, data warehouses—also known as decision support systems—play a critical role in business and corporate decision-making. Unlike transactional databases, which store real-time operational data, data warehouses archive historical information for analytical purposes. As new data is continuously added and existing data updated, these warehouses grow significantly in size. They are typically much larger than transactional systems due to their focus on supporting complex, data-intensive queries. Analytical applications often issue ad hoc and resource-heavy queries involving large-scale data access, multiple joins, and extensive aggregations. These OLAP (Online Analytical Processing) queries are far more complex than standard OLTP (Online Transaction Processing) operations. Several adaptive cache replacement algorithms, such as ARC, CAR, and LIRS, improve performance by using historical data and reuse distance rather than simple recency, as in LRU. These algorithms adjust dynamically to access patterns using self-tuning parameters. For example, CAR uses two clocks to distinguish between short-term and long-term utility and adapts using evicted page history. Other techniques, like sequential pattern mining, clustering, and frequency-size-based heuristics, have also been explored to enhance cache and database performance.

In materialized view selection, various strategies have been proposed to improve query response time and reduce maintenance costs. Techniques include clustering similar queries (CBDMVS), association rule mining (ARMMVVM), greedy algorithms, and optimization-based methods like particle swarm and game theory. Some works focused on selecting views based on storage constraints, while others used AND/OR graphs to model query plans. Recent efforts also integrate cost models, data cubes, and heuristic search algorithms to identify optimal or near-optimal view sets. Reviews of these methods highlight gaps and suggest future research directions to refine view selection in cloud and big data environments. Traditional cache management algorithms like Least Recently Used (LRU) are effective for general workloads but have limitations under specific access patterns. LRU prioritizes recency, which can cause inefficiencies, especially in scan-heavy or mixed workloads. It often fails to distinguish between frequently accessed and rarely used data, leading to cache pollution. Approximations such as CLOCK and

DUELING CLOCK attempt to mitigate these issues but still inherit core weaknesses. The Touch Count approach, by inserting blocks mid-list, shows better adaptability in such scenarios.

1.1 Frequency-Based Cache Management

Least Frequently Used (LFU) retains blocks based on access count but struggles with stale data occupying cache space due to outdated usage patterns. To improve LFU's adaptability, several enhanced techniques have been proposed. Frequency-Based Replacement (FBR) integrates reference counting for better locality tracking. The Multi-Queue (MQ) system classifies blocks into different queues based on usage frequency. 2Q and LIRS algorithms further refine this idea, keeping only consistently accessed blocks, thereby reducing short-term frequency bias.

1.2 Recency-Frequency Hybrid Policies

To combine the strengths of LRU and LFU, LRU-K considers the last *K* references for eviction decisions, capturing both frequency and recency. LRFU (Lately Referenced and Frequently Used) uses a Combined Recency-Frequency (CRF) score to estimate future block access likelihood. While effective, LRFU may falter with cyclical access patterns. CLOCK-Pro improves CLOCK by integrating LIRS-style decisions without preset parameters, offering strong performance across diverse workloads.

1.3 Adaptive Cache Management

Adaptive algorithms leverage eviction history to respond dynamically to changing patterns. ARC (Adaptive Replacement Cache) and its variants, such as CAR and CART, merge recency and frequency using a self-adjusting parameter. ARC manages two main lists—short-term (T1) and long-term (T2)—and their respective ghost lists (B1, B2) to continuously rebalance priorities. CAR introduces dual CLOCK structures to track both short- and long-term utility. These strategies outperform static methods by automatically adjusting to workload behavior, addressing key LRU shortcomings.

II. RESEARCH METHODOLOGY

In data warehouses supporting business intelligence systems, materialized views are widely used to enhance the performance of complex decision support queries. These queries often involve expensive operations such as joins across large tables and heavy aggregations (e.g., SUM, AVG, COUNT, VARIANCE, STDDEV, MAX, MIN). As these queries are typically executed over massive historical datasets to detect trends, the computational cost can be significant.

Creating materialized views helps reduce this burden by precomputing and storing intermediate results. However, when dealing with workloads containing millions of queries, generating and maintaining materialized views becomes costly. Therefore, it is essential to select a minimal set of materialized views that can maximize performance benefits for the most demanding queries.

1. Component-Level Materialization:

Unlike conventional methods that generate materialized views for entire queries, the proposed approach creates views based on frequently recurring query components and subqueries.

2. Plan-Based Analysis:

Instead of analyzing raw query text, we extract frequent subcomponents by studying execution plans from past query workloads, offering a more accurate representation of query behavior.

3. Efficient Tree Mining Algorithm:

We introduce a specialized tree mining algorithm designed to operate on execution plan trees. This method incorporates advanced pruning techniques to reduce search space and manage large-scale query workloads effectively.

Through extensive experiments using standard benchmarks, real-world, and synthetic datasets, we demonstrate that our approach identifies a richer set of candidate subqueries compared to state-of-the-art and traditional techniques, leading to better materialization strategies.

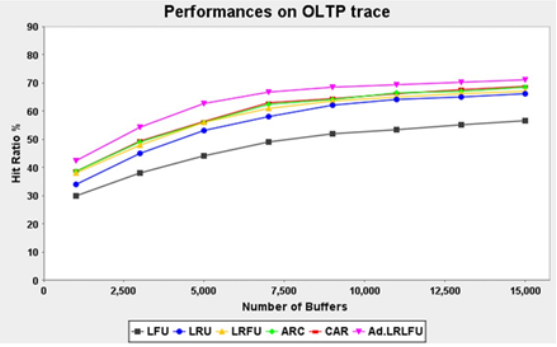

Fig. 1: Performance comparison on OLTP trace

III. RESULTS ANALYSIS

The Adaptive LRLFU algorithm consistently delivers the highest hit ratios across diverse workloads, particularly excelling in scenarios with periodic or irregular access patterns. Although its advantage is less pronounced in random workloads, it still outperforms leading algorithms such as ARC, CAR, LIRS, and CART. It demonstrates strong resilience to unpredictable access patterns and benefits significantly from larger buffer caches.

In workloads dominated by a fixed working set, Predictive LRLFU offers similar performance, but Adaptive LRLFU generally remains more effective. As cache size grows beyond the working set, performance differences between algorithms diminish; however, Adaptive LRLFU continues to utilize memory more efficiently, especially in limited cache environments.

CONCLUSION

This work introduces two innovative buffer management algorithms—Predictive LRLFU (PR-LRLFU) and Adaptive LRLFU (AD-LRLFU)—which leverage access pattern periodicity to enhance cache replacement. Evaluated on both synthetic and real-world workloads, they consistently surpass traditional methods such as LRU, LIRS, ARC, and CAR, especially in scenarios with small cache sizes or recurring access trends. Their predictive and adaptive features enable robust performance across diverse workloads, making them ideal for shared cloud databases.

Additionally, the PR-ACF algorithm is proposed for flash-based systems. It reduces write operations by selectively evicting clean pages using a probability-driven model. By dividing the cache into HOT and COLD zones and targeting only cold, clean pages for eviction, PR-ACF achieves superior hit ratios, fewer write operations, and faster runtimes compared to existing flash caching techniques when tested on Flash-DBSim.

REFERENCES

[1] Chou, Hong-Tai, and David J. DeWitt. ”An evaluation of buffer management strategies for relational database systems.” Algorithmica 1.1-4 (1986): 311-336.

[2] Robinson, John T., and Murthy V. Devarakonda. Data cache management using frequency-based replacement. Vol. 18. No. 1. ACM, 1990.

[3] O’neil, Elizabeth J., Patrick E. O’neil, and Gerhard Weikum. ”The LRU-K page replacement algorithm for database disk buffering.” ACM SIGMOD Record 22.2 (1993): 297-306.

[4] Cherkasova, Ludmila. Improving WWW proxies performance with greedy-dual- size-frequency caching policy. Hewlett-Packard Laboratories, 1998.

[5] Kim, Jong Min, et al. ”A low-overhead high-performance unified buffer manage- ment scheme that exploits sequential and looping references.” Proceedings of the 4th conference on Symposium on Operating System Design & Implementation- Volume 4. USENIX Association, 2000.

[6] Lee, Donghee, et al. ”LRFU: A spectrum of policies that subsumes the least re- cently used and least frequently used policies.” IEEE transactions on Computers 50.12 (2001): 1352-1361.

[7] Jiang, Song, and Xiaodong Zhang. ”LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance.” ACM SIGMET- RICS Performance Evaluation Review 30.1 (2002): 31-42.

[8] Megiddo, Nimrod, and Dharmendra S. Modha.

”ARC: A Self-Tuning, Low Over- head Replacement Cache.” FAST. Vol. 3. No. 2003. 2003.163

[9] Bansal, Sorav, and Dharmendra S. Modha. ”CAR: Clock with Adaptive Replace- ment.” FAST. Vol. 4. 2004.

[10] Jiang, Song, and Xiaodong Zhang. ”Making LRU friendly to weak locality work- loads: A novel replacement algorithm to improve buffer cache performance.” IEEE Transactions on Computers 54.8 (2005): 939-952.

[11] Jiang, Song, Feng Chen, and Xiaodong Zhang. ”CLOCK-Pro: An Effective Im- provement of the CLOCK Replacement.” USENIX Annual Technical Conference, General Track. 2005.

[12] He, Zhen, Richard Lai, and Alonso Marquez. ”On using cache conscious cluster- ing for improving OODBMS performance.” Information and Software Technology 48.11 (2006): 1073-1082.

[13] Chiang, I. Robert, Paulo B. Goes, and Zhongju Zhang. ”Periodic cache replace- ment policy for dynamic content at application server.” Decision Support Systems 43.2 (2007): 336-348.

[14] Wan, Shenggang, et al. ”An adaptive cache management using dual LRU stacks to improve buffer cache performance.” Performance, Computing and Communi- cations Conference, 2008. IPCCC 2008. IEEE International. IEEE, 2008.

[15] Saurabh Chauhan, Dharamveer Singh, Atul Kumar Singh (2022) “Artificial Intelligence In The Military: An Overview Of The Capabilities, Applications, And Challenges”, Journal of Survey in Fisheries Sciences, Vol 9 (2) pp 984-991. https://doi.org/10.53555/sfs.v9i2.2911

[16] Kiran, Dharamveer Singh, Nitin Goyal, (2023) “Analysis Of How Digital Marketing Affect By Voice Search”, Journal of Survey in Fisheries Sciences, Vol. 30 (2) 407-412. https://doi.org/10.53555/sfs.v10i3.2890

[17] Yukti Tyagi, Dharamveer Singh, Ramander Singh, Sudhir Dawra (2024) “Analysis Of The Most Recent Trojans On The Android Operating System”, Educational Administration: Theory and Practice, Vol. 30(2) 1320-1327. https://doi.org/10.53555/kuey.v30i2.6846

[18] Shivanee Singh, Dharamveer Singh, Ravindra Chauhan (2023) “Manufacturing Industry: A Sustainability Perspective On Cloud And Edge Computing”, Journal of Survey in Fisheries Sciences, pp 1592-1598. https://doi.org/10.53555/sfs.v10i2.2889