AMERICAN ACADEMIC PUBLISHER

INTERNATIONAL JOURNAL OF DATA SCIENCE AND MACHINE LEARNING (ISSN: 2692-5141)

Volume 05, Issue 02, 2025, pages 357-365
Published Date: - 08-12-2025

Enhancing Oracle RAC Performance Through Advanced Caching Models: A Comprehensive Synthesis of Buffer Cache Theory, Cache Fusion Dynamics, and High-Availability Design

John K. Ellison
Global Systems Research Lab, University of Wellington

ABSTRACT

This article presents a comprehensive synthesis of theoretical models, practical techniques, and engineering best practices for designing, tuning, and operating caching and high-availability mechanisms in clustered Oracle Database environments. Drawing from foundational operating-system theory and combinatorial optimization, and integrating contemporary analyses of hierarchical and temporal caching, the work develops an inclusive conceptual framework for understanding buffer cache behavior, Cache Fusion dynamics in Oracle Real Application Clusters (RAC), and strategies to reduce wait events through instance-specific block allocation and cache-tuning. The Methods section details a rigorous, text-based methodology that combines analytical reasoning from queuing and caching theory, utility-optimization formulations, and applied best-practice procedures articulated in vendor documentation. The Results section offers descriptive findings: identification of primary performance bottlenecks, characterization of trade-offs between cache size, data placement, and inter-instance coherence, and demonstration of how hierarchical TTL and LRU approximations inform buffer management policies in shared-disk clustered databases. The Discussion interprets these descriptive results to propose prescriptive interventions, including tailored buffer cache partitioning, selective instance-specific block allocation, and deployment architectures for Autonomous Database on dedicated Exadata infrastructure that reconcile high availability with low-latency access patterns. Practical limitations and implementation caveats are examined, along with future research directions such as adaptive utility-based cache controllers and integration of content-integrity verification techniques into LRU-centric caches. The article aims to be publication-ready and serves as both an advanced tutorial for database architects and a theoretical roadmap for researchers studying caching and high-availability in large-scale clustered database systems.

KEYWORDS

Oracle RAC, buffer cache, Cache Fusion, hierarchical caching, LRU approximation, high availability, instance-specific allocation

INTRODUCTION

The demand for continuously available, high-performance database services has driven considerable innovation in clustering, caching, and coherence protocols. In large enterprises, Oracle Database is frequently deployed in clustered topologies—most prominently Real Application Clusters (RAC)—to provide fault tolerance, horizontal scalability, and resource pooling (Oracle Corporation, 2023). However, clustering introduces a fundamental tension: maintaining a logically unified, consistent view of data across multiple instances often requires coordination that can increase latency and cause application-visible wait events. The buffer cache, a memory-resident store of frequently accessed blocks, plays a vital role in this dynamic: its size, allocation strategy, and interaction with

shared-storage coherence mechanisms determine much of the system's performance envelope (Oracle Corporation, 2021).

A rich body of theoretical work exists for caching systems more broadly—spanning analyses of hierarchical web caches, TTL-based networks, LRU approximations, and performance optimization through utility maximization (Che, Tung, & Wang, 2002; Fofack et al., 2014; Dehghan et al., 2016; Garetto, Leonardi, & Martina, 2016). These efforts provide formal tools and approximations that can guide engineering decisions for database buffer caches, yet the literature historically treats web caches and database buffer caches as distinct domains. This separation overlooks deep analogies: both systems mediate access to slower storage, both exhibit temporal locality and skewed popularity distributions, and both must navigate placement trade-offs under constrained memory resources (Traverso et al., 2013; Fricker, Robert, & Roberts, 2012).

Moreover, classical operating-systems theory contextualizes the concept of resource sharing and the costs of contention and synchronization in multiprocess environments (Coffman & Denning, 1973). When combined with combinatorial optimization perspectives, such as knapsack problem formulations that model allocation under capacity constraints (Kellerer, Pferschy, & Pisinger, 2004), we gain the means to reason about buffer cache partitioning and the prioritization of blocks across instances.

Despite these theoretical foundations, practical gaps persist. Oracle provides extensive high-availability and tuning documentation, yet translating prescriptive recommendations into specific allocation and consistency strategies that minimize cross-instance cache traffic, reduce Cache Fusion wait events, and adapt to temporal locality in workloads remains an open problem for many administrators (Oracle Corporation, 2023; Oracle Corporation, 2021). Oracle Support and contemporary practitioners have offered targeted techniques—such as instance-specific block allocation—to reduce RAC wait events in production settings, but such guidance is often not framed within unified, theoretically grounded models (Oracle Support, 2020; Natti, 2023).

This article addresses that gap by articulating a unified theoretical and practical framework. The objective is twofold: first, to reinterpret advanced caching theory through the lens of Oracle's buffer cache and RAC coherence mechanisms; second, to derive and explain practical, implementable strategies that reduce wait events and improve availability while preserving consistency guarantees. The contribution is primarily synthetic and explanatory—combining formal insights with vendor best practices and field reports to produce detailed, actionable guidance and to lay out avenues for future empirical validation and automation.

The introduction proceeds by defining the core challenges—cache coherence, Cache Fusion mechanics, buffer cache sizing, and the tension between availability and latency—and then surveys relevant prior work from both the theoretical and practical literature that informs the subsequent analysis (Oracle Corporation, 2023; Che, Tung, & Wang, 2002; Garetto, Leonardi, & Martina, 2016). The section ends by articulating the specific research questions guiding this synthesis: how should buffer cache resources be allocated in RAC environments to minimize cross-instance coherence overhead; how can hierarchical and temporal caching models guide cache tuning; and what concrete operational steps can administrators take, especially when deploying Oracle Autonomous Database on dedicated Exadata infrastructure, to harmonize availability and performance?

METHODOLOGY

The methodology adopted here is deliberately text-based and analytic: rather than reporting novel empirical experiments, it rigorously combines established theoretical constructs with practical documentation and field reports to produce prescriptive reasoning and design patterns. This approach is suitable because many of the most impactful insights about caching and coherence are structural—rooted in resource constraints, workload patterns, and protocol dynamics—and therefore can be generalized through careful argumentation grounded in prior work

(Coffman & Denning, 1973; Kellerer, Pferschy, & Pisinger, 2004; Che, Tung, & Wang, 2002).

The methodology has four complementary strands:

Conceptual mapping. The first step establishes correspondences between canonical caching concepts—LRU behavior, TTL-based hierarchies, temporal locality, and utility optimization—and Oracle-specific components: buffer cache, library cache, data block states, and Cache Fusion. This mapping identifies which theoretical results are transferable and which require reinterpretation due to architectural differences (Garetto, Leonardi, & Martina, 2016; Traverso et al., 2013).

Resource-allocation framing. Using combinatorial optimization insights, particularly knapsack problem intuition, the methodology frames buffer cache partitioning and instance-specific block allocations as constrained resource-allocation problems. This framing supports qualitative assessments of trade-offs between concentrating blocks within an instance versus distributing them for redundancy, and ties directly to practical knobs discussed in Oracle documentation (Kellerer, Pferschy, & Pisinger, 2004; Oracle Corporation, 2021).

Performance phenotype derivation. Leveraging temporal-locality models and LRU approximations, the methodology derives qualitative performance phenotypes—sets of workload characteristics that predict whether a particular tuning choice will succeed. For example, workloads with high temporal locality and skewed popularity distributions may benefit more from instance-specific allocation than uniform distribution across instances (Traverso et al., 2013; Fricker, Robert, & Roberts, 2012).

Operational procedure synthesis. Finally, the methodology consolidates practical, vendor-recommended procedures (e.g., buffer cache sizing, RAC parameterization, Exadata deployment patterns) with the theoretical guidance above to produce stepwise, explainable interventions. These are intentionally describable without scripts or code because the focus is on conceptual applicability across versions and environments (Oracle Corporation, 2023; Oracle Corporation, 2021; Oracle Support, 2020).

Throughout, claims are cross-referenced to the provided literature to ensure that major statements are supported by the sources listed. The aim is to produce a comprehensive, reproducible chain of reasoning that converts high-level theoretical principles into precise, defensible operational recommendations for Oracle RAC and Exadata deployments.

RESULTS

The Results section synthesizes descriptive findings arising from the methodology. Because the work is analytic and synthetic rather than empirically experimental, the results are claims, structured observations, and derived trade-offs grounded in cited literature. Each finding is accompanied by an explanation of its theoretical basis and practical implication.

1. Cache Fusion and Inter-Instance Wait Events: Root Causes and Characterization

Cache Fusion, Oracle's mechanism for sharing data blocks directly between instances without touching disk, reduces I/O but introduces inter-instance messaging costs, serialization, and wait events when multiple instances attempt to access or modify the same blocks (Oracle Corporation, 2023). The principal driver of Cache Fusion wait events is contention for ownership and the need to transfer the most recent version of a block between instances, which manifests as network latency and CPU overhead on both sender and receiver. This characterization aligns with the documented RAC wait events and performance advisories (Oracle Support, 2020).

The theoretical interpretation is that Cache Fusion creates a coherence protocol analogous to distributed shared-memory systems studied in operating-systems literature: ownership transfers and invalidations incur

synchronization overheads that grow with contention and the frequency of block state transitions (Coffman & Denning, 1973). Thus, workloads with high write-sharing or frequent cross-instance reads of freshly written blocks will experience higher Cache Fusion wait events than predominantly read-mostly workloads with static hot data.

2. Buffer Cache Size and the Diminishing Returns of Uniform Expansion

Oracle's buffer cache is a finite resource whose aggregate size across instances is often the primary lever for reducing physical I/O (Oracle Corporation, 2021). Increasing total cache capacity reduces miss rates but suffers from diminishing returns: once the working set fits comfortably in memory, additional capacity yields little benefit. The knapsack perspective highlights that capacity allocation is about choosing which blocks to retain—when capacity is scarce, prioritization matters more than raw size (Kellerer, Pferschy, & Pisinger, 2004).

From hierarchical caching results, we note that adding memory at higher levels (instance buffers) is equivalent to increasing the top-level cache in a multilevel cache hierarchy, but unless the placement policy accounts for temporal locality and inter-instance sharing patterns, naive expansion can leave coherence overhead unchanged (Che, Tung, & Wang, 2002; Fofack et al., 2014). Thus, expansion must be accompanied by allocation policies that minimize cross-instance block transfers.

3. Temporal Locality, Popularity Skew, and Partitioning Benefits

Temporal locality—the tendency to reaccess items in short time windows—makes per-instance allocation strategies effective. When specific clients or application servers are predominantly attached to a particular instance, the working set likely has a strong per-instance locality. In such cases, allocating instance-specific blocks (i.e., biasing certain ranges or objects toward a single instance's buffer cache) reduces inter-instance transfers (Traverso et al., 2013). Empirical field reports echo this: instance-specific block allocation has reduced RAC wait events in production applications (Natti, 2023).

However, when temporal locality is global and access patterns are highly skewed across instances—e.g., many instances reading the same hot object—the benefit of partitioning diminishes and may even increase coherence overhead due to replication or repeated transfers among instances.

4. LRU Approximations and the Practicality of Simple Policies

LRU (Least Recently Used) and its approximations are well-studied for caches and have versatile approximations that yield accurate predictions about miss rates under certain workloads (Fricker, Robert, & Roberts, 2012; Martina, Garetto, & Leonardi, 2014). When buffer cache behavior approximates LRU, administrators can use these approximations to forecast the effects of resizing caches and to assess the impact of various partitioning schemes.

From the practical standpoint, LRU-like policies are attractive because of their simplicity and predictability. Integrating LRU-based analysis with knowledge of temporal locality enables better decisions: if an instance's workload exhibits strong LRU-friendly behavior, dedicating a proportion of the buffer cache to that instance can reduce misses without hurting overall hit rates significantly.

5. Hierarchical TTL-Based Cache Networks as a Metaphor for Multi-Level Database Cache Architectures

TTL-based hierarchical caching—where cached items have time-to-live values that control their lifespan at different levels—offers a conceptual model for database systems with multiple caching layers (e.g., client caches, instance buffer caches, storage cache). TTL semantics help manage staleness and coherency by bounding the divergence window while minimizing synchronization traffic (Fofack et al., 2014). While database systems require stronger consistency than typical web caches, TTL ideas can be adapted for read-mostly data or for multi-tier architectures where bounded staleness is acceptable.

This metaphor suggests that combining strict coherence (for critical, transactional blocks) with looser TTL-like caching for large read-mostly datasets can yield favorable trade-offs between latency and protocol overhead.

6. Utility Optimization and Adaptive Allocation

Treating cache allocation as a utility optimization problem—where the "value" of keeping a block in cache depends on request rates, impact on response time, and the cost of transferring block ownership—permits principled allocation decisions (Dehghan et al., 2016). Even if exact optimization is computationally infeasible in real time, approximate utility-based rules can be implemented that bias cache admission and eviction based on measured request frequencies and cross-instance access patterns.

This approach aligns with the knapsack intuition: allocate cache slots where the marginal utility per unit of memory is highest. In practical terms, this may translate to preferential retention of large objects that are expensive to re-materialize or of blocks with high cross-instance write contention where avoiding transfers yields big latency reductions.

7. Exadata and Autonomous Database: Architectural Implications

Oracle's Autonomous Database on Dedicated Exadata Infrastructure offers a deployment model that co-designs software and hardware to optimize data locality and I/O offload (Oracle Corporation, n.d.). Exadata's storage servers and smart-scanning capabilities change the calculus for buffer cache allocation: offloading complex scans to storage can reduce the need for large instance buffer caches for analytical workloads, while transactional workloads with small, hot working sets still demand carefully tuned buffer caches at the instance layer.

Therefore, the deployment pattern matters: for transactional OLTP workloads on Exadata, buffer cache tuning and instance-specific allocation remain central; for analytical or mixed workloads, architects should consider how Exadata offload mechanisms change the optimal distribution of memory and compute resources (Oracle Corporation, n.d.).

8. Content-Integrity Verification and Cache Performance Trade-Offs

Introducing content-integrity checks before storing items in caches, as examined in web caching literature, imposes computational overhead and can alter effective hit/miss dynamics (Bianchi et al., 2013). For databases, checksum and redo logs offer integrity guarantees, but adding extra integrity verification at cache admission time (for example, cryptographic signatures) can create additional CPU latency that offsets cache benefits. The performance price of such verification must be weighed against the integrity requirements of the application.

9. Multi-Terabyte, Multi-Gbps Routers and Large-Scale I/O Considerations

Scaling databases to multi-terabyte sizes and supporting multi-gigabit throughput introduces engineering constraints that have analogues in network router designs: hardware must balance forwarding (I/O) capacity with memory for state (cache) management (Rossini et al., 2014). For database clusters, this translates to ensuring the network, storage servers, and CPU subsystems are balanced with the aggregate buffer cache capacity to avoid non-cache bottlenecks influencing perceived cache efficacy.

These descriptive results form the basis for the prescriptive interventions in the Discussion: they specify when instance-specific allocation is likely to help, when LRU-based tuning suffices, and when a hybrid TTL/strict-coherence architecture may be desirable.

DISCUSSION

The preceding synthesis yields multiple interlocking prescriptions and nuanced caveats. This section interprets the

results in depth, weighs counter-arguments, and discusses limitations and future research directions.

Interpreting Cache Fusion Dynamics and the Role of Placement

Cache Fusion is simultaneously a powerful optimization (avoiding disk I/O by transferring blocks directly between instances) and a potential performance sink when contention is high (Oracle Corporation, 2023). The appropriate response depends on workload shape. For workloads with strong per-instance locality—often the case when application tiers maintain affinity to specific instances—biasing block placement to those instances (instance-specific allocation) reduces the frequency of ownership transfers, thereby lowering Cache Fusion wait events (Natti, 2023). This finding suggests a practical operational rule: when monitoring reveals a small number of hot objects exhibiting high write or read-after-write sharing across instances, administrators should consider steering those objects to a preferred instance.

Counter-argument and nuance: Steering blocks to a single instance creates an availability dependency; if that instance fails, other instances may experience increased latency as they fetch blocks from shared storage or reassign ownership. Oracle RAC's architecture mitigates single-instance failure by allowing another instance to assume responsibilities, but failover costs are nonzero. Thus, instance-specific allocation must be combined with robust failover planning and possibly proactive replication or prewarming on standby instances for critical blocks.

Balancing Cache Size Expansion with Allocation Policies

Expanding aggregate buffer cache remains a common first response to performance problems; yet our analysis underscores that expansion alone can be inefficient. Because cache effectiveness depends on which blocks are retained as much as how many, administrators should adopt policies that consider both capacity and placement. For example, partitioning the buffer cache into global and instance-local regions—allocating a portion for shared hot blocks and another portion for instance-affine data—can capture the benefits of both redundancy and locality (Kellerer, Pferschy, & Pisinger, 2004).

A potential objection is operational complexity: managing partitions adds configuration overhead and may complicate dynamic adaptation to workload shifts. To address this, implement partition policies with conservative defaults and monitoring-based automated resizing where possible. Oracle's documentation provides knobs for SGA and buffer management; these should be used in conjunction with monitoring to adjust partition sizes based on observed hit rates and cross-instance traffic (Oracle Corporation, 2021).

Temporal Locality as a Lever for Simplicity

Temporal locality offers a sweet spot: when workloads have well-defined, short-term hot sets, simple LRU policies paired with modest instance-specific partitioning can achieve most of the potential performance gains with low complexity (Fricker, Robert, & Roberts, 2012). In other words, rather than attempting heavyweight utility optimization, many environments benefit from lightweight heuristics: allocate a small instance-local buffer in addition to a shared pool, rely on LRU behavior for eviction, and monitor for hotspot migration.

However, workloads with long-tail popularity distributions or highly variable temporal locality may require more sophisticated approaches. Here utility-based allocation—admitting blocks based on measured marginal benefit and eviction cost—becomes attractive despite higher computational complexity (Dehghan et al., 2016). Implementers must weigh the measurement overhead against expected gains.

Hierarchical TTL Models and Relaxed Consistency

TTL-based hierarchical caching is widely used in web systems to trade freshness for reduced synchronization overhead (Fofack et al., 2014). For databases, strict transactional semantics often preclude TTL-based relaxation for

critical data. Yet many enterprise workloads include large volumes of read-mostly data (e.g., product catalogs, historical analytics) for which bounded staleness is acceptable. Partitioning data into consistency classes and using TTL-like cache lifetimes for the relaxed class lets systems reap hierarchical caching benefits without violating transactional invariants.

There is a trade-off: if an application's correctness depends on the latest data, TTL is unsuitable. The practical implication is to employ data classification carefully and ensure application-level semantics tolerate bounded staleness before implementing TTL-inspired caching.

Exadata-Specific Considerations

Exadata and Autonomous Database offerings introduce hardware-level features—smart scanning, storage offload, and integrated resource management—that change optimization priorities (Oracle Corporation, n.d.). In particular, Exadata offload can reduce the pressure on instance buffer caches for certain analytical queries. Therefore, architects must consider the workload mix when tuning: favor larger instance caches for write-heavy, low-latency OLTP workloads; rely more on Exadata offload for analytical workloads and reduce instance cache allocation accordingly.

Counterpoint: Exadata offload may introduce different bottlenecks—storage server CPU, network between instances and storage servers—and so buffer cache tuning remains relevant to avoid unnecessary transfers and to exploit Exadata efficiently. The design principle is holistic: treat buffer cache allocation as one lever among several, and coordinate with storage and network capacity planning (Oracle Corporation, n.d.; Rossini et al., 2014).

Content-Integrity Verification: When to Pay the Cost

Adding integrity checks to caching admission and storage provides stronger correctness guarantees but at CPU and latency cost (Bianchi et al., 2013). The decision to enable such checks should be guided by threat models and regulatory requirements. If integrity is paramount and the additional verification latency is acceptable relative to application requirements, integrating verification makes sense. Otherwise, rely on existing database-integrity mechanisms (redo logs, checksums) and keep cache admission cheap and fast.

Limitations of the Synthesis

This work synthesizes theoretical models and manufacturer guidance but does not include new experimental measurements; therefore, its recommendations should be validated in production-like environments before full-scale deployment. The provided analyses assume typical hardware and network characteristics; environments with extreme latencies, heterogeneous storage architectures, or nonstandard access patterns may experience different behaviors.

Moreover, the translation of web-cache theory to database buffers involves approximations: databases require ACID guarantees and often complex locking semantics, which constrain the applicability of some models (e.g., pure TTL semantics). Where practical, adaptation strategies are proposed, but careful validation is still required.

Future Research Directions

Several promising avenues emerge:

Adaptive, utility-driven cache controllers. Implement controllers that estimate the marginal utility of retaining blocks based on request rates, cost-to-transfer, and write probabilities, and then make admission/eviction decisions dynamically. This directly operationalizes the knapsack and utility-optimization intuition (Dehghan et al., 2016).

Workload-aware instance-specific allocation. Develop systems that detect application affinity and automatically

steer data placement to instances, adjusting in real time as affinities change. This would reduce manual tuning and mitigate failover concerns through proactive replication.

Hybrid strict/relaxed coherence models. Formalize hybrid architectures that enforce strict coherence for transactional objects and relaxed TTL-style caching for read-mostly objects, including formal bounds on staleness and their impact on application correctness.

Performance cost of integrity verification. Quantify the trade-off between computational cost of content-integrity verification and the latency benefits of caching in representative database workloads to derive decision rules for enabling integrity checks selectively (Bianchi et al., 2013).

Empirical validation on Autonomous Database and Exadata. Because Exadata changes the baseline assumptions of storage and compute locality, large-scale experiments on Exadata platforms would clarify how much instance buffer cache tuning matters in modern cloud-delivered database services.

CONCLUSION

This article has synthesized theoretical and practical perspectives into an integrated view of caching and high-availability strategies for Oracle clustered databases. The central insight is that cache performance and coherence overheads are determined not solely by raw buffer capacity but critically by placement, temporal locality, and workload-specific sharing patterns. Instance-specific block allocation offers a powerful practical tool when workloads exhibit strong per-instance locality and can materially reduce Cache Fusion wait events (Natti, 2023; Oracle Support, 2020). However, such allocation must be balanced with availability concerns, Exadata offload considerations, and the realities of transactional consistency.

LRU approximations and hierarchical caching models provide accessible analytical tools for administrators, allowing forecasting of hit-rate changes and the evaluation of partitioning strategies (Fricker, Robert, & Roberts, 2012; Garetto, Leonardi, & Martina, 2016). Utility-optimization and knapsack-like reasoning underpin principled allocation strategies that prioritize blocks with the highest marginal benefit per memory unit (Kellerer, Pferschy, & Pisinger, 2004; Dehghan et al., 2016).

Operationally, database architects should: (1) monitor Cache Fusion and related wait events to identify cross-instance sharing patterns, (2) classify data by access and consistency requirements to determine where TTL-style relaxation is permissible, (3) apply instance-specific allocation where affinity exists, and (4) balance buffer cache sizing with Exadata offload capabilities. Finally, the field would benefit from research that operationalizes adaptive controllers and validates the synthesized prescriptions under production workloads and modern hardware platforms.

REFERENCES

1. Oracle Corporation. "Oracle Database High Availability Overview and Best Practices" Oracle Help Center, Release 23. https://docs.oracle.com/en/database/oracle/oracledatabase/23/haovw/overview-oracle-database-high-availability-best-practices.html
2. Oracle Corporation. "Oracle Database Performance Tuning Guide:13 Tuning the Database Buffer Cache" https://docs.oracle.com/en/database/oracle/oracledatabase/21/tgdba/tuning-database-buffer-cache.html
3. Oracle Corporation. "Oracle Autonomous Database on Dedicated Exadata Infrastructure: Cloud Deployment Guide." https://docs.oracle.com/en/cloud/paas/autonomousdatabase/dedicated/adbdi/index.html
4. Natti, M. (2023). Reducing Oracle RAC Wait Events by Using Instance-Specific Block Allocation for Production Applications. The Eastasouth Journal of Information System and Computer Science, 1(01), 65–68.

https://doi.org/10.58812/esiscs.v1i01.447

5. E. G. Coffman, Jr. and P. J. Denning, Operating Systems Theory. Prentice Hall Professional Technical Reference, 1973.
6. H. Kellerer, U. Pferschy, and D. Pisinger, Knapsack problems. Springer, 2004.
7. H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: modeling, design and experimental results," Selected Areas in Communications, IEEE Journal on, vol. 20, no. 7, pp. 1305–1314, Sep 2002.
8. N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Performance evaluation of hierarchical TTL-based cache networks," Computer Networks, vol. 65, pp. 212–231, 2014.
9. M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. Tay, "A Utility Optimization Approach to Network Cache Design," in Proc. of IEEE INFOCOM 2016, 2016, to appear, arXiv preprint arXiv:1601.06838.
10. M. Garetto, E. Leonardi, and V. Martina, "A Unified Approach to the Performance Analysis of Caching Systems," ACM Trans. Model. Perform. Eval. Comput. Syst., vol. 1, no. 3, pp. 12:1–12:28, May 2016. http://doi.acm.org/10.1145/2896380
11. S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, "Temporal Locality in Today's Content Caching: Why It Matters and How to Model It," SIGCOMM Comput. Commun. Rev., vol. 43, no. 5, pp. 5–12, Nov. 2013.
12. C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in Proceedings of the 24th International Teletraffic Congress, 2012.
13. V. Martina, M. Garetto, and E. Leonardi, "A Unified Approach to the Performance Analysis of Caching Systems," in Proc. of IEEE INFOCOM 2014. IEEE, 2014, pp. 2040–2048.
14. G. Bianchi, A. Detti, A. Caponi, and N. Blefari Melazzi, "Check before storing: What is the performance price of content integrity verification in LRU caching?" ACM SIGCOMM Computer Communication Review, vol. 43, no. 3, pp. 59–67, 2013.
15. G. Rossini, D. Rossi, M. Garetto, and E. Leonardi, "Multi-Terabyte and multi-Gbps information centric routers," in INFOCOM, 2014 Proceedings IEEE, 2014, pp. 181–189.
16. Oracle Corporation, "Oracle Real Application Clusters Documentation," 2023. https://docs.oracle.com/cd/E11882_01/rac
17. Oracle Support, Reducing Cache Fusion Wait Events in Oracle RAC. Oracle White Papers., 2020.
18. J. Smith, High-Performance Oracle RAC: Strategies for Optimization. Pearson, 2021.