iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** XI **Month of publication:** November 2024

DOI: https://doi.org/10.22214/ijraset.2024.65347

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue XI Nov 2024- Available at www.ijraset.com

Simulation-Based Evaluation of Big Data Caching Mechanisms

Maha Dessokey[1], Sherif M. Saif[2]
Computers and Systems Department, Electronics Research Institute, Cairo, Egypt

Abstract: ***This paper provides a simulation-based evaluation that addresses memory management problems throughout Big Data processing. A significant problem occurs with in-memory computing when there is not enough available memory for processing the whole chunk of data, and hence some data must be selected for deletion to make room for new ones. The selected research strategy is to use different cache selection and replacement algorithms, such as Adaptive Replacement Cache (ARC) and Low Inter-Reference Recency Set (LIRS) algorithms, besides the default one, which is Least Recently Used (LRU). A simulator was built by the authors to assess the use of different caching approaches on Big Data platforms. The evaluation showed that the LIRS and the ARC algorithms gave a better hit ratio for different workloads than the LRU algorithm.***

I. INTRODUCTION

The advent of Big Data has necessitated the development of efficient data processing techniques. In-memory computing, which leverages main memory for data processing, has emerged as a promising approach to accelerate data analytics. However, the limited capacity of main memory necessitates the eviction of less frequently accessed data to accommodate new data. This process, known as cache replacement, significantly impacts the overall performance of in-memory computing systems.

A. Overview

New technologies are needed for real data analytics since some data are continuously gathered, processed, analyzed, and the information is usable as soon as it is generated [1], [2]. There is no one-size-fits-all approach to create a Big Data architecture due to the rapid advancement of technology and the highly competitive market in which it operates.

B. Problem Description

In various Big Data platforms that support in-memory processing, such as Apache Spark and Apache Flink, the choice of which data to cache in memory is entirely up to the developer [3][4][5]. When creating an application consisting of actions with complex dependencies, it might take time to decide whether data should be cached or not. Caching all of the data in memory will cause a severe reduction in performance when there is not enough memory available.

C. Contribution and Paper Organization

In this work, a proposal will be discussed, implemented on a simulator, and then evaluated to solve the stated problem. The proposal is to modify the cache selection and replacement algorithm used by the processing platform.

This section serves as an introduction to the paper. The remaining sections of the paper are organized as follows: section 2 discusses the cache selection and replacement algorithms used in Big Data platforms; section 3 presents the developed simulator to evaluate other algorithms; section 4 presents the results of our evaluation according to the built simulation, and section 5 concludes the research and presents future work.

II. CACHE SELECTION AND REPLACEMENT ALGORITHMS

Cache selection and replacement algorithms play a crucial role in Big Data platforms, as they address two major challenges: data retrieval speed and memory efficiency [3]. In Big Data environments, where large volumes of data must be processed and analyzed rapidly, caching helps reduce the load on storage systems by temporarily holding frequently accessed data in faster, albeit smaller, memory spaces, such as RAM. The efficiency of cache algorithms significantly impacts overall system performance, making these algorithms essential in optimizing data processing, reducing latency, and ensuring system scalability.

A. *Importance of Cache Selection in Big Data Platforms*

Cache selection refers to the process of identifying which data should be placed in the cache based on patterns in data access. In Big Data platforms, where data characteristics and usage patterns can vary widely, an effective cache selection strategy can ensure that only the most valuable data remains in the cache. This prioritization improves data availability and speeds up access times for frequently queried information. For instance, in recommendation systems or real-time analytics, data elements that are accessed repeatedly benefit from being readily available in the cache. Selecting the correct data to cache, such as hot data or frequently accessed user records, can lead to significant performance gains, particularly when data access patterns are predictable patterns. Caching algorithms are especially effective in situations where data access patterns are predictable. In these cases, caching algorithms can prioritize data that is frequently accessed or recently accessed, thereby improving data availability and access speeds in Big Data environments. This can be particularly beneficial in applications with regular or repeated access patterns, as it allows the caching system to keep the most relevant data in memory. Hence the caching system is responsible for managing cached data and deciding when and what to evict when cache storage reaches its limit. This process is essential in Big Data due to the continuous influx of data that quickly fills the cache, requiring efficient replacement strategies to free up space without disrupting performance. Traditional replacement strategies, like Least Recently Used (LRU) and Least Frequently Used (LFU), are often adapted or combined to suit specific Big Data needs, as their standard forms may not account for the highly dynamic and diverse data access patterns in these systems. Advanced algorithms, such as Adaptive Replacement Cache (ARC) and Multi-Queue algorithms, offer tailored solutions for Big Data by balancing between recency and frequency of access, thus improving cache efficiency and extending the utility of limited memory resources [6][7][8][9].

B. *Big Data Platforms and Memory Caching*

Big Data platforms incorporate a combination of technologies, tools, and algorithms that facilitate data ingestion, storage, and processing at large scales. These platforms, such as Hadoop, Spark, and cloud-based solutions like AWS and Google BigQuery, are critical for handling structured, semi-structured, and unstructured data efficiently.

Discussing cache selection and replacement algorithms within Big Data platforms, implies that a large-scale infrastructure optimizes data access speeds, reliability, and cost-effectiveness. In these environments, cache algorithms are essential for quickly retrieving data from memory, reducing latency, and supporting real-time or near-real-time analytics tasks. These platforms support workloads that range from batch processing to interactive querying, making caching strategies a crucial aspect of ensuring the timely availability and accessibility of data for high-demand applications.

Examples of Big Data platforms that use cache selection and replacement algorithms, include Hadoop, Spark, and AWS Big Data tools. Each of these platforms addresses the challenges of Big Data through their own architectures and caching mechanisms to handle the storage, retrieval, and processing of large datasets efficiently.

Caching has many advantages such as:

1) *Enhanced Performance and Speed:* Caching significantly reduces data retrieval time, making high-frequency data quickly accessible. In large-scale data systems like Spark or Hadoop, caching intermediate results or commonly accessed datasets minimizes the need for repeated disk reads, which are slower than in-memory access.
2) *Optimized Resource Utilization:* By efficiently managing the cache, platforms can prevent resource bottlenecks and ensure smooth scaling even as data volumes grow. Replacement algorithms that minimize cache misses reduce the need for expensive I/O operations, allowing Big Data systems to handle higher loads with greater efficiency.
3) *Adaptability to Diverse Workloads:* Modern cache algorithms for Big Data are often designed to be adaptable, considering factors such as access frequency, recency, and data size. This adaptability is crucial for handling diverse workloads, where access patterns can shift between batch processing, real-time querying, and machine learning tasks.

Overall, caching algorithms in Big Data platforms enhance data accessibility, improve processing efficiency, and enable scalable system designs, addressing both the volume and velocity of data processing requirements in modern data-intensive applications.

C. *Cache Replacement Algorithms and Memory Caching Techniques*

There are traditional cache replacement algorithms and advanced ones. The following subsections will shed more light on this taxonomy.

1) Traditional Cache Replacement Algorithms

The Least Recently Used (LRU) algorithm is a widely used cache replacement strategy. It evicts the least recently accessed data block from the cache to make room for new data. While LRU is a simple and effective algorithm, it may not be optimal for all work-loads. For instance, workloads with periodic access patterns may suffer from LRU's tendency to evict frequently accessed data.

a) Least Recently Used (LRU)

The LRU policy assumes that the block that has been accessed most recently will most probably be accessed once more shortly and that the block that has been least recently utilized should be evicted and replaced by the cache manager when memory is full [6].
The implementation of the LRU algorithm consists of only one queue. The Most Recent Used (MRU) block is located at the head of the queue, and the Least Recently Used is located at the tail. When a block in the queue is re-accessed, its location is changed to stand at the top of the queue.

2) Advanced Cache Replacement Algorithms

To address the limitations of LRU, more sophisticated algorithms have been developed. The Adaptive Replacement Cache (ARC) algorithm dynamically adapts to varying workload characteristics, combining the strengths of both LRU and Most Recently Used (MRU) strategies. ARC maintains separate lists for recently used and recently unused data blocks, dynamically adjusting the size of these lists based on the observed access patterns.
Another promising algorithm is the Low Inter-Reference Recency Set (LIRS) algorithm. LIRS focuses on identifying data blocks that are likely to be accessed in the near future. By analyzing the reference patterns of data blocks, LIRS can make more informed decisions about which blocks to evict.

a) Adaptive Replacement Cache (ARC)

The ARC algorithm [10] uses two LRU cache buffers to manage both recency and frequency in block accesses. The sizes of the queues are dynamically adjusted. In this algorithm, a new block enters into the first LRU stack. This stack is labeled S1 for recently used blocks. If the block is re-accessed, the block is moved to the second stack labeled S2 for frequently used blocks.

b) Low Inter-Reference Recency Set (LIRS)

The Inter-Reference Recency (IRR) metric is used as the recorded history information of each block in the cache, where the IRR of a block is equal to the number of distinct blocks between two subsequent accesses to the same block in a request sequence. For example, when a request sequence is 3-1-2-4-0-2-3, the IRR of Block 3 equals 4, and the IRR of Block 2 equals 2. According to LIRS policy [11], blocks with high IRRs will be chosen for replacement when needed. It is assumed that if a block's current IRR is high, its subsequent IRR will likely be high too.
Memory caching has a long history and is frequently used in processors, operating systems, databases, file systems, web servers, and storage systems. In Apache Spark, the only used cache selection and replacement algorithm is the Least Recently Used (LRU) [6]. The LRU algorithm performs poorly in some workloads as scanning workloads and cyclic access (loop-like) workloads in which the loop length is greater than the cache size. To solve the LRU issues, numerous buffer cache algorithms have been suggested [8]. Some of these algorithms were studied in [9].

III. PROPOSING AND BUILDING A CACHE SIMULATOR

To evaluate the performance of these algorithms, a comprehensive simulation study was conducted. The simulation environment was designed to mimic real-world Big Data workloads, including various data access patterns and memory constraints. The results of the simulation experiments demonstrated that LIRS and ARC consistently outperform LRU in terms of hit ratio and cache efficiency.

A. Simulation-Based Evaluation

In this paper, a simulator for the cache memory with different eviction techniques was built using Java language. It was proved that both adaptive replacement cache (ARC) [10] and low inter-reference recency set (LIRS) [11] algorithms have better performance for multiple workloads than LRU. The three algorithms are discussed in the following subsections.
In Java, a cache simulator that uses multiple caching policies such as LRU, ARC, and LIRS was built by authors to evaluate different cache selection and replacement algorithms. Objects are only stored via their ids and size.

The cache policies were evaluated using trace-based simulations. The traces that were used were collected from Storage Networking Industry Association (SNIA) [12]. SNIA is a non-profit global organization that is committed to making storage networks a full and reliable solution for the IT industry. The simulator flow chart is shown in Figure 1, the simulator flow is as follows:

1) Set parameters, RamSize, BlockSize, and the selection and replacement algorithm to be used (LIRS, LRU, and ARC).
2) Calculate the number of available blocks= RamSize/BlockSize;
3) Load the trace file "workload," which contains the ID of the requested blocks in sequence.
4) Each of the four classes of "Algorithms" has functions
   a) Get (BlockID, Size): check if the requested block is already in the cache or not. If the block is already in the cache, then Hits increased by one, and the value returned.
   b) Set (BlockID, Size): If the block is not found in the cache, then the miss is increased by one, and the Set function is called. The Set function checks if the cache is not full, then the block is added, else it replaces an existing one according to each replacement policy.
5) At the end, the hit ratio is calculated, which is equal to No. of hits/total No. of requests. The aim of a good replacement policy is to minimize the number of misses and increase the number of hits.

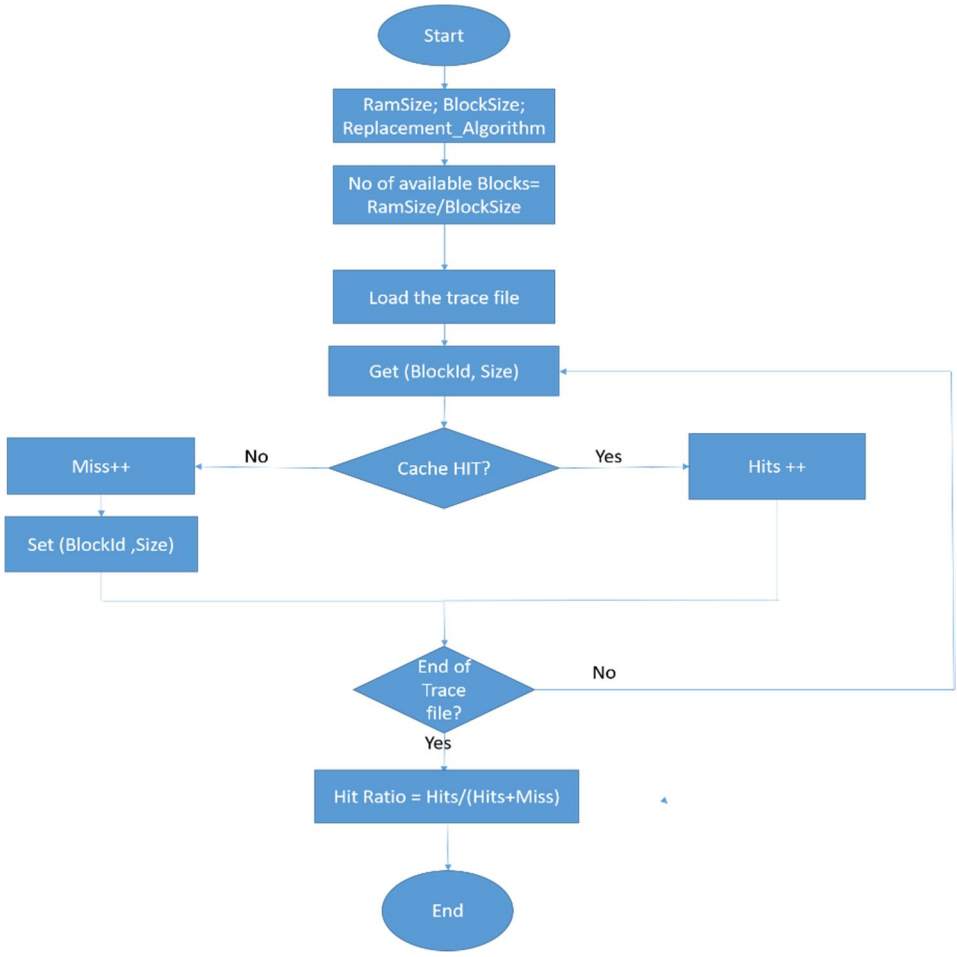


Figure 1: Simulator Flow Chart

The selection and replacement algorithms were evaluated using two block I/O traces which are OLTP and K5sample. The traces were downloaded from SNIA [12] and went through the cleaning phase first to extract only the block id and the block size to be used in our simulator.

OLTP: trace is provided by the authors of the ARC algorithm [10], and it includes references to 3,656,580 requests.

K5sample: The FUJITSU K5 cloud service's parallel storage workloads were obtained and examined by [7]. A sample of this trace with 998 requests was downloaded from SNIA and used in our experiment.

IV. EVALUATION AND SIMULATION RESULTS

Table 1, shows the hit ratio calculated by running the OLTP and K5sample traces as input to the cache simulator and using LRU, ARC, and LIRS replacement algorithms. The results show how ARC and LIRS algorithms are better than the LRU algorithm and that the LIRS algorithm gave the largest Hit ratio in both traces. From these results, we recommend implementing the LIRS algorithm in the Apache Spark platform [13] as a suggested future work.

Table 1: Simulator Hit Ratio results

Trace	LRU	ARC	LIRS
OLTP	75%	75 %	95 %
K5Sample	12.3 %	14 %	58 %

A. Impact on Big Data Processing

The choice of a suitable cache replacement algorithm can significantly impact the performance of Big Data processing systems. By selecting an algorithm that aligns well with the specific workload characteristics, organizations can improve query response times, reduce resource utilization, and enhance overall system performance.

V. CONCLUSION

In this paper, the cache selection and replacement policy is used to manage the memory and improve the system performance. The only cache selection and replacement policy used in Apache Spark is LRU. Along with the LRU algorithm, two cache policies, ARC and LIRS, were created and tested by creating a cache simulator. Both algorithms, especially the LIRS policy, performed better than LRU. Therefore, we came to the conclusion that additional cache selection and replacement policies must be added to the Apache Spark platform and made available to developers as options.

A. Future Directions

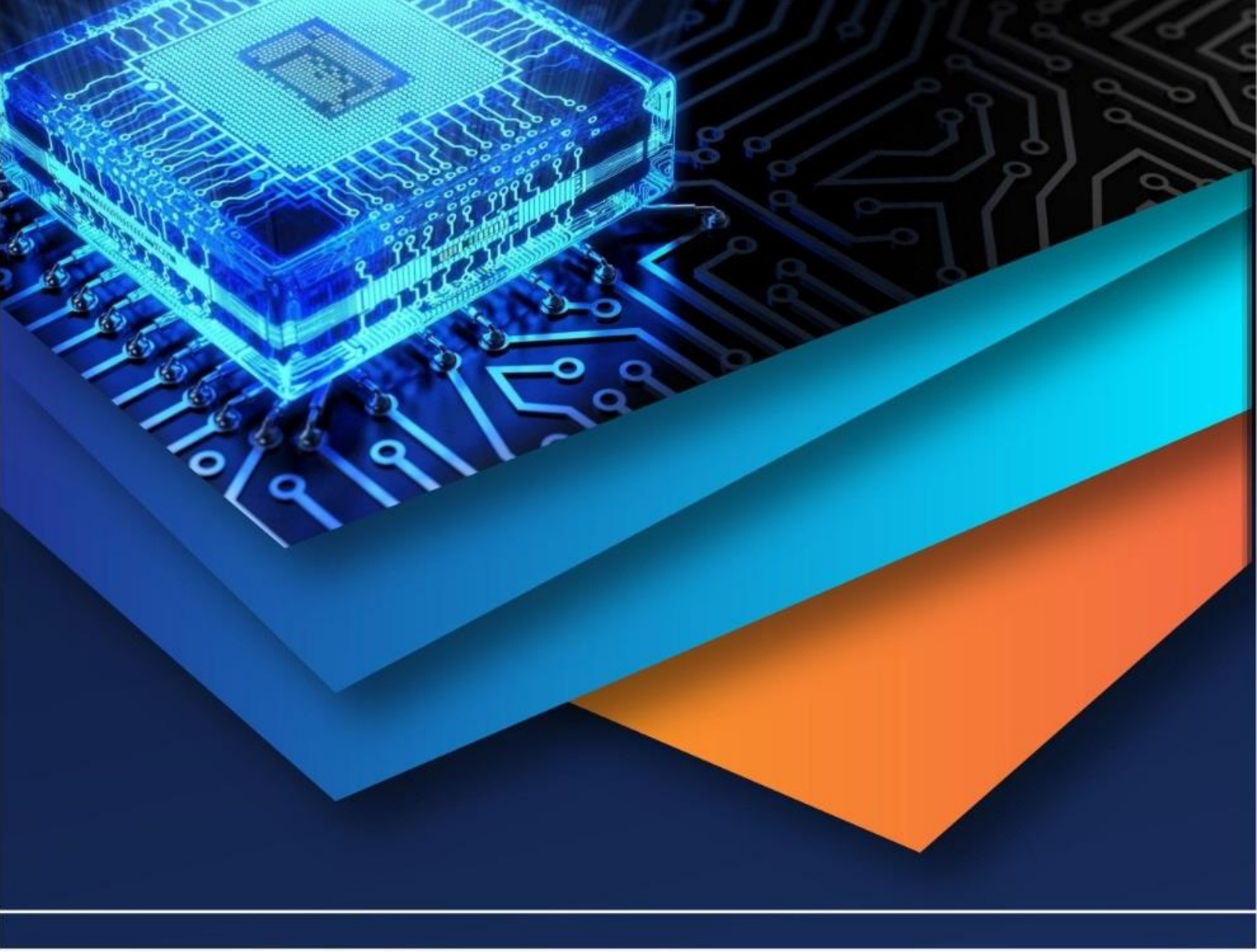
While LIRS and ARC have shown promising results, there is still room for further research and optimization. Future work may explore hybrid algorithms that combine the strengths of multiple strategies, adaptive algorithms that can dynamically adjust their behavior based on changing workload patterns, and hardware-accelerated cache replacement mechanisms. Additionally, investigating the impact of different memory hierarchies and storage technologies on cache replacement algorithms is an important area of research.

By advancing the state-of-the-art in cache replacement algorithms, we can unlock the full potential of in-memory computing and drive innovation in Big Data analytics.

REFERENCES

[1] M. S. Mahmud, J. Z. Huang, S. Salloum, T. Z. Emara, and K. Sadatdiynov, “A survey of data partitioning and sampling methods to support big data analysis,” Big Data Mining and Analytics, vol. 3, no. 2, Art. no. 2, 2020.
[2] A. H. Ali, “A survey on vertical and horizontal scaling platforms for big data analytics,” International Journal of Integrated Engineering, vol. 11, no. 6, Art. no. 6, 2019.
[3] G. B. Kamal ElDahshan Eman Selim, Ahmed Ismail Ebada, Mohamed Abouhawwash, Yunyoung Nam, “Handling Big Data in Relational Database Management Systems,” Computers, Materials & Continua, vol. 72, no. 3, pp. 5149–5164, 2022, doi: 10.32604/cmc.2022.028326.
[4] A. M. Mickaelian, “Big Data in Astronomy: Surveys, Catalogs, Databases and Archives,” Communications of the Byurakan Astrophysical Observatory, vol. 67, pp. 159–180, 2020.
[5] S. Vellaipandiyan, “Big Data Framework - Cluster Resource Management with Apache Mesos,” International Journal of Latest Trends in Engineering and Technology (IJLTET), vol. 3, pp. 228–294, Mar. 2014.
[6] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark: Lightning-Fast Big Data Analysis. O’Reilly Media, 2015.
[7] Apache Flink. [Online]. Available: https://flink.apache.org/
[8] N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. A. Rashid, “A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench,” Journal of Big Data, vol. 7, no. 1, Art. no. 1, Dec. 2020, doi: 10.1186/s40537-020-00388-5.
[9] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, “Big Data technologies: A survey,” Journal of King Saud University - Computer and Information Sciences, vol. 30, no. 4, pp. 431–448, 2018, doi: https://doi.org/10.1016/j.jksuci.2017.06.001.

[10] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network," in Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, USA, 2017, pp. 483–498.
[11] M. Dessokey, S. M. Saif, S. Salem, E. Saad, and H. Eldeeb, "Memory Management Approaches in Apache Spark: A Review," in Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2020, Cham, 2021, pp. 394–403.
[12] N. Megiddo and D. S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," in 2nd USENIX Conference on File and Storage Technologies (FAST 03), San Francisco, CA, Mar. 2003. [Online]. Available: https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache
[13] S. Jiang and X. (Frank) Zhang, "LIRS: An Efficient Low Inter-reference Recency Set Replacement to Improve Buffer Cache Performance," in Performance Evaluation Review, Jun. 2002, vol. 30, pp. 31–42. doi: 10.1145/511399.511340.
[14] H. Yoshida, "Storage Networking Industry Association," in Encyclopedia of Database Systems, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 2815–2816. doi: 10.1007/978-0-387-39940-9_1347.
[15] A. K. Garate-Escamilla, A. H. El Hassani, and E. Andres, "Big Data Scalability Based on Spark Machine Learning Libraries," in Proceedings of the 2019 3rd International Conference on Big Data Research, New York, NY, USA, 2019, pp. 166–171. doi: 10.1145/3372454.3372469.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY