ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

# A General Approach to Scalable Buffer Pool Management

Milind Madhukar Kulkarni[1], Prof.A.V.Mophare[2],

M.E. Computer Science and Engineering, Department of Computer Science and Engineering, N.B.N.C.O.E, Solapur Pune National Highway, Kegaon, Solapur, Maharashtra, India.

Assistant Professor, Department of Computer Science and Engineering, N.B.N.C.O.E, Solapur Pune National Highway, Kegaon, Solapur, Maharashtra, India.

**ABSTRACT:** The data management system usually trust the maintenance scheme based on its temporary and permanent memory management abilities. In this scenario, buffers are playing a vital role to keep the memory clean and which optimizes the memory with high end of efficiency. The main objective of this proposed approach is to eliminate the overhead and efficiently optimize the scalability of buffer pool management. This approach follows the powerful optimal page replacement algorithm to improvise the scalability of buffer pool management as well as improve the performance of the system. This eliminates the overall operation cost of fetching and handling the queries of database as well as which builds a checking platform to provide a stage to identify the performance of optimal page replacement algorithm as well as compare the features of the proposed approach with other existing methodologies. For all the present implemented system guarantees the approach produces good accuracy and produces greater performance by conducting several experiments, which will results and produces the data for analyzing performance of proposed approach.

**KEYWORDS:** Buffer Pool Management, Optimal Page Replacement Algorithm, Multi-Core Environment, Data Management, Scalability, Data Replacement.

## I. INTRODUCTION

Data-Processing frameworks, for example, databases, generally utilize top of the line servers with numerous cores for elite. In the cloud, an Amazon RDS database can have up to 32 virtual CPUs [vCPUs] on such a server, a substantial number of worker threads run all the while to amplify computing efficiency of the system. With the high computing power, the general framework execution is every now and again dictated by how quick the worker threads can get to the information they process. As a typical strategy, an Data-Processing framework keeps up a buffer pool in the client memory space for its specialist threads to reserve the informational indexes that are effectively gotten too. The framework painstakingly deals with the buffer pool utilizing experienced approaches to limit expensive plate I/O tasks. worker threads get to the buffer pool synchronously at a high recurrence. With the expanding number of cores, the cradle pool administration must be very versatile and proficient to successfully deal with the developing preparing simultaneousness.

Else, it can turn into a genuine framework bottleneck, prompting critical execution decrease. Indeed, that compose execution issues have been generally seen in different frameworks, for example, PostgreSQL, MySQL, and Oracle. The execution issue is caused by the dispute on the secures utilized as a part of the cradle administration, especially the bolt that ensures a core information structure utilized for information substitution strategy. The procedure settles on choices on which information pages ought to be reserved in memory to retain adequately asks for on-circle information. To execute the procedure, the threads keep up an information structure to track their information get to history.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

Refresh the information structure in light of page gets to with the goal that substitution choices can be made in light of that history recorded in the information structure. To ensure the uprightness of the information structure, the updates must be made in a serialized drift. In this way, a bolt is important to synchronize the updates. Bolt conflict happen when the bolt is held by one worker thread and different threads must hold up as caught up with pausing or potentially setting switches. Numerous substitution calculations, systems and their executions have been proposed. They build and oversee profound page get to history to augment hit proportions and limit the cost brought about by plate I/O tasks. These calculations, methods as a rule take activities upon every I/O get to, either a hit or a miss in the buffer, which basically incorporate a succession of updates in the information structure to record the entrance history.

Though the action are usually designed to be simple and efficient to minimize overhead, in a production system where a large number of threads access on-disk data frequently and concurrently, the high frequency and concurrency may lead to serious lock contention. Performance degradation occur by lock contention can be significant in large-scale systems. This subject has been a major research problem for years.

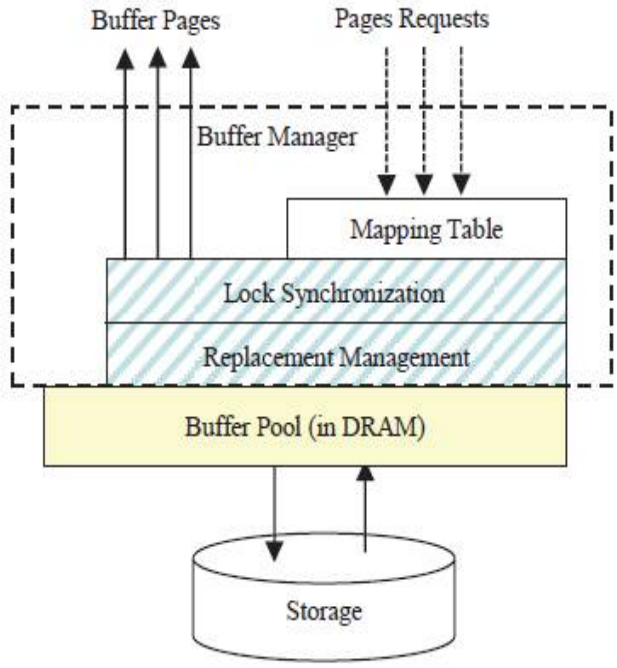


**Fig.1 Buffer Manager Model**

**A. Buffer Management**

In a data processing system, a buffer stores a collection of buffer pages of fixed sizes and is shared by worker threads. Data pages read from hard disks are cached in the buffer for possible reuses. A buffer manager uses some data structures such as linked lists and mapping tables (e.g. hash tables or trees) to organize the metadata of the buffer pages, including identifiers of the cached data pages, status, and pointers to form linked lists and mapping tables. The buffer manager is a central component frequently used by all worker threads upon each page request. Simultaneous updates on its data structures have to be carried out in a controlled fashion to maintain its data structure integrity. Usually lock synchronization is used for this purpose.

The use of locks to synchronize mapping table searching and updating does not limit system scalability. The mapping table usually uses distributed locking or hierarchical locking. At the same time, since the mapping table is not changed upon hits, concurrent accesses are allowed upon hits, and exclusive accesses are only required upon misses,

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

which are usually rare, compared to hits. Both factors above help maintaining high scalability. For example, in a hash table, the metadata of buffer pages are distributed into a large number of small hash buckets, each of which is protected by a local read/write lock.

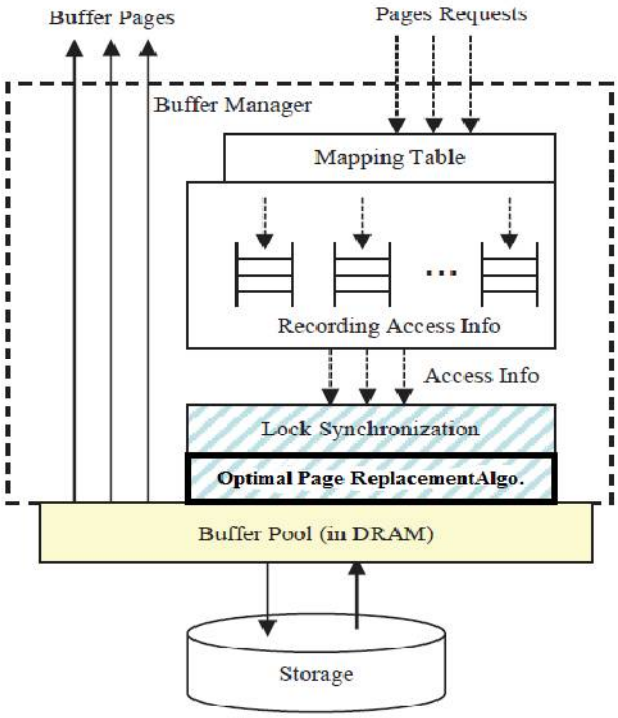


**Fig.2Proposed System Architecture**

Even when multiple threads need to access the same bucket, they can search the bucket concurrently. Only when searches fail (i.e., misses), the exclusive accesses to a bucket are required. We focus on the lock contention in the replacement management because (a) the replacement management may use one lock for its entire data structure, which is a single point of hot spot and (b) most replacement algorithms require an update of their data structures upon every page access. Therefore, a thread has to acquire the lock for every page request to exclusively conduct the replacement management operations. The highly contented lock may dramatically degrade system performance on a multi-core/multiprocessor system.

**B. Proposed System Summary**

This proposed approach provides a general solution to improve the scalability of buffer pool management using optimal replacement algorithms for the data processing systems on physical on-premises machines and virtual machines in the cloud. Instead of making a difficult trade-off between the high hit ratio of a replacement algorithm and the low lock contention of its approximation, we design a system framework, called optimal page replacement algorithm that eliminates almost all lock contention without requiring any changes to an existing algorithm.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

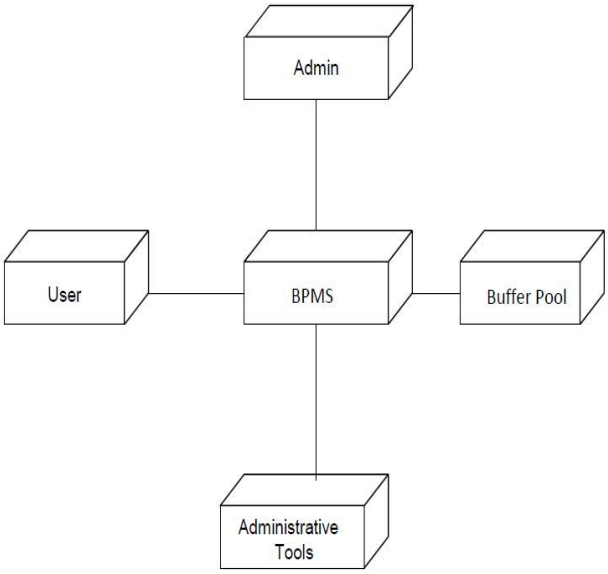


**Fig.3Implementation Scenario of the Proposed System**

The use of locks to synchronize mapping table searching and updating does not limit system scalability. The mapping table usually uses distributed locking or hierarchical locking.

At the same time, since the mapping table is not changed upon hits, concurrent accesses are allowed upon hits, and exclusive accesses are only required upon misses, which are usually rare, compared to hits. The proposed system guarantees the scalability and performance.The efforts addressing performance degradation due to lock contention have been made actively in the system and database community. In general, lock contention can be reduced by applying the following different approaches.

***Distributed Locks***

Reducing lock granularity is a commonly used method for decreasing lock contention. Replacing a globally shared lock with multiple fine-grained locks can remove the single point of contention. LRU replacement algorithm and set-associative page cache use multiple lists to manage their buffers. Each list is protected by a separate lock. When a new page enters the buffer LRU and set-associative page cache choose a list by hashing, which guarantees that a page enters the same list every time it is loaded from the disk. The distributed lock approaches, including the one used in LRU and set-associative page cache, have serious Drawbacks.

First, they cannot be used to implement replacement algorithms that need to detect access sequences, because pages in the same sequence may be distributed into multiple lists.

Second, though pages can be evenly distributed into multiple lists, accesses to buffer pages may not. Lists with hot pages, such as top-level index pages or pages in a small table for a parallel join, may still suffer from lock contention.

Third, to reduce contention the buffer has to be partitioned into hundreds, even thousands, of lists. Each list has a much smaller size than the buffer size. With the small lists, those pages that need a special protection from eviction, such as dirty pages and index pages, may be evicted prematurely from the buffer. In contrast, our framework is able to implement all replacement algorithms without partitioning the buffer.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

***Reducing Lock Holding Time***

The larger the lock holding time, the more serious the contention would be. Reducing lock holding time is another effective approach to reduce lock contention. The TSTE (two stage transaction execution) strategy used separates disk I/O and lock acquisition into two mutually exclusive stages and ensures that all the data pages a transaction needs are already in local memory before they are locked. In this method, TSTE reduces the delay caused by lock contention in the disk-resident transaction processing system to the same level as that experienced by Memory-resident transaction systems. This framework uses fetch to reduce lock holding time by reducing hardware cache misses when running the replacement algorithm and by executing a replacement algorithm for multiple accesses.

***Wait-Free Algorithms and Transactional Memory***

As lock synchronization can cause issues like performance degradation and priority inversion, wait-free synchronization [32] addresses these issues by guaranteeing that a transaction completes the operation of accessing the shared resource in a limited number of steps regardless of the execution progress of other transactions. However, programs adopting this technique are difficult to design, and it is possible that an algorithm does not have its wait-free implementation. Moreover, wait-free synchronization requires atomic primitives supported by hardware. Transactional memory is another approach to address the issue of lock contention. In a transactional memory, a transaction is considered as a series of operations on the shared resources. The atomicity of its execution is guaranteed by either hardware or software that implements transactional memory. It improves system scalability through enabling optimistic concurrency control. While hardware transactional memory has not been available, there are various software transactional memory (STM) implementations.

## II. LITERATURE SURVEY

In the year of 2014 the authors "X. Ding, P. B. Gibbons, M. A. Kozuch, and J. Shan" proposed a paper titled "Gleaner: mitigating the blocked-waiter wakeup problem for virtualized multicore applications [1]" in that they described such as: the number of cores in a multicore node increases in accordance with Moore's law, the question arises as to what are the costs of virtualized environments when scaling applications to take advantage of larger core counts. While a widely-known cost due to preempted spinlock holders has been extensively studied, this paper studies another cost, which has received little attention. The cost is caused by the intervention from the VMM during synchronization-induced idling in the application, guest OS, or supporting libraries--we call this the blocked-waiter wakeup (BWW) problem. This systemsystematically analyzes the cause of the BWW problem and studies its performance issues, including increased execution times, reduced system throughput, and performance unpredictability. To deal with these issues, the paper proposes a solution, Gleaner, which integrates idling operations and imbalanced scheduling as a mitigation to this problem. We show how Gleaner can be implemented without intrusive modification to the guest OS. Extensive experiments show that Gleaner can effectively reduce the virtualization cost incurred by blocking synchronization and improve the performance of individual applications by 16x and system throughput by 3x.

In the year of 2012 the authors "A. Da Zheng and A. S. Szalay" proposed a paper titled "A parallel page cache: Iops and caching for multicore systems [2]" in that they described such as: presenting a set-associative page cache for scalable parallelism of IOPS in multicore systems. The design eliminates lock contention and hardware cache misses by partitioning the global cache into many independent page sets, each requiring a small amount of metadata that fits in few processor cache lines. We extend this design with message passing among processors in a non-uniform memory architecture (NUMA). We evaluate the set-associative cache on 12-core processors and a 48- core NUMA to show that it realizes the scalable IOPS of direct I/O (no caching) and matches the cache hits rates of Linux's page cache. Set-associative caching maintains IOPS at scale in contrast to Linux for which IOPS crash beyond eight parallel threads.

In the year of 2010 the authors "M. Yui, J. Miyazaki, S. Uemura, and H. Yamana" proposed a paper titled "Nb-GCLOCK: A non-blocking buffer management based on the generalized CLOCK" in that they described such as: propose a non-blocking buffer management scheme based on a lock-free variant of the GCLOCK page replacement algorithm. Concurrent access to the buffer management module is a major factor that prevents database scalability to processors. Therefore, we propose a non-blocking scheme for bufferfix operations that fix buffer frames for requested

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

pages without locks by combining Nb-GCLOCK and a non-blocking hash table. Our experimental results revealed that our scheme can obtain nearly linear scalability to processors up to 64 processors, although the existing locking-based schemes do not scale beyond 16 processors.

## III. EXPERIMENTAL RESULTS

A general solution to improve the scalability of buffer pool management using optimal replacement algorithms for the data processing systems on physical on-premises machines and virtual machines in the cloud. Instead of making a difficult trade-off between the high hit ratio of a replacement algorithm and the low lock contention of its approximation, we design a system framework, called optimal page replacement algorithm that eliminates almost all lock contention without requiring any changes to an existing algorithm.

The use of locks to synchronize mapping table searching and updating does not limit system scalability. The mapping table usually uses distributed locking or hierarchical locking. At the same time, since the mapping table is not changed upon hits, concurrent accesses are allowed upon hits, and exclusive accesses are only required upon misses, which are usually rare, compared to hits. Here We have evaluated the scalability of the systems by setting the buffer size equal to the data sizes of the workloads, and have shown that lock contention can be reduced significantly by combining the batching and pre fetching techniques. However, buffer sizes are usually much smaller than data sizes in real systems. Thus, the ability of the systems to reduce costly I/O operations by improving hit ratios is also critical to the overall performance.

***First-In, First-Out (FIFO)Page Replacement Algorithm***: Another low-overhead paging algorithm is the FIFO (First-In, First-Out) algorithm. For ex - consider a supermarket that has enough shelves to display exactly k different products. One day, some company introduces a new convenience food—instant, freeze-dried, organic yogurt that can be recon- situated in a microwave oven. It is an immediate success, so our finite super -market has to get rid of one old product in order to stock it.

***Least Recently Used*** is a good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called ***LRU (Least Recently Used)*** paging. Although LRU is theoretically realizable, it is not cheap. To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front is a very time consuming operation, even in hardware (assuming that such hardware could be built).

One possibility is to find the product that the supermarket has been stocking the longest (i.e., something it began selling 120 years ago) and get rid of it on the grounds that no one is interested any more. In effect, the supermarket maintains a linked list of all the products it currently sells in the order they were introduced. The new one goes on the back of the list; the one at the front of the list is dropped. We evaluate the overall performance of the system on corei3 processor we changes the buffer size, workload with algorithm and without algorithm and compare these result for gaining the higher Throughput & scalability.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

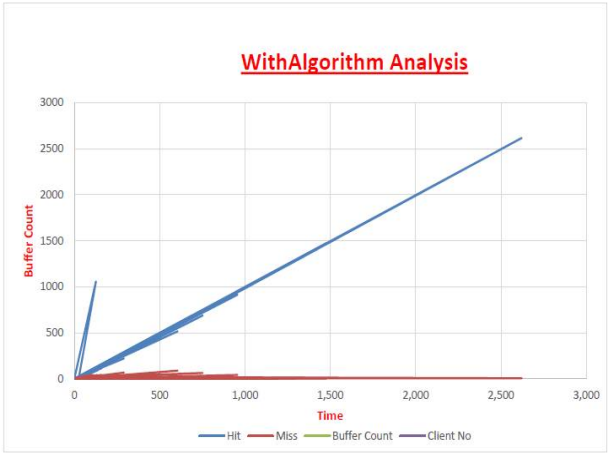


**Fig.4Graphical Analysis of Time Vs. Buffer Count with the Analysis of Algorithm**

For this analysis we have taken total 1100 client ridings with buffer count, hit, miss of each and every client and plot this hit and miss ratio using optimal page replacement algorithm.

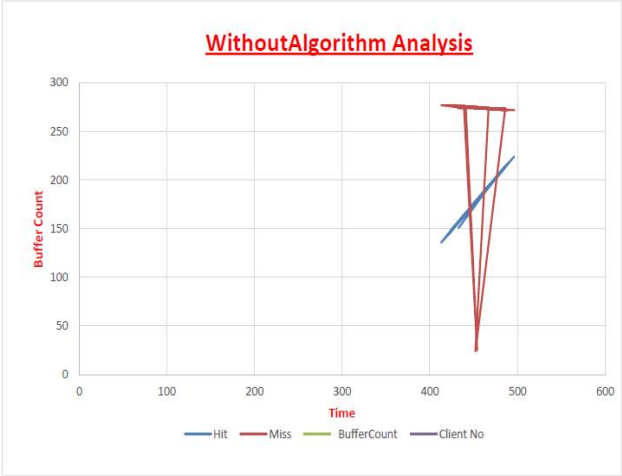


**Fig.5Graphical Analysis of Time Vs. Buffer Count without the Analysis of Algorithm**

For this analysis we have taken total 1100 client ridings with buffer count, hit, miss of each and every client and plot this hit and miss ratio without optimal page replacement algorithm.

The following figure illustrates the Run time estimation without algorithm process.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 7, Issue 5, May 2018

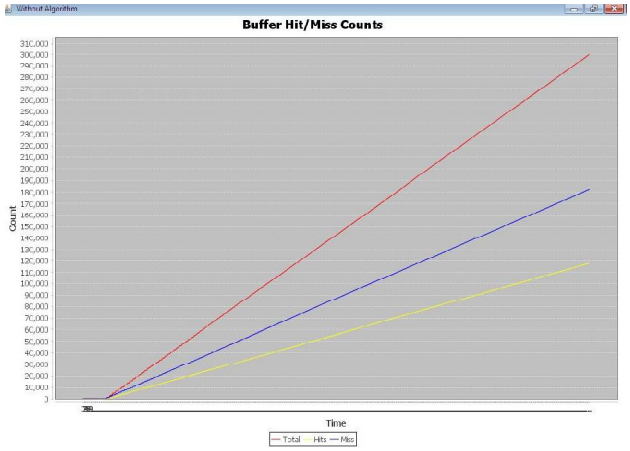


**Fig.6Runtime Analysis without the consideration of Algorithm**

The following figure illustrates the Run time estimation with algorithm process.

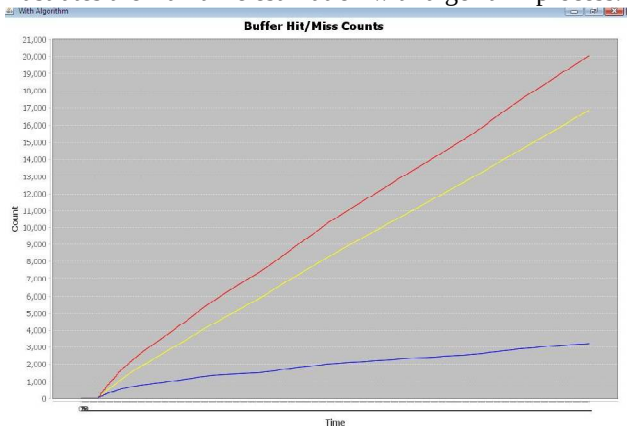


**Fig.7Runtime Analysis with Algorithm Consideration**

## IV. CONCLUSION AND FUTURE SCOPE

Here we indicate the adaptability/scalability issue because of secure dispute in the execution of replacement algorithms for the administration of buffer cache. We proposed a proficient and adaptable system, optimal page replacement algorithm for page substitution. Without processing change, the execution preferred standpoint of the first replacement algorithms won't be imperiled, and human exertion is likewise limited.

The accompanying things can be enhanced in future over the present work.

(a) The optimal page replacement algorithm gives the exact outcome (Accuracy level=80%) as the proprietor of approach is proposed.

(b) Optimal page replacement algorithm is bit moderate in handling various Client Request as contrasted and other page replacement algorithm.

(c) It likewise gives higher throughput and adaptable contrasted and other calculation.

(d) This approach is tried on single framework not conveyed framework so all Test Results has a place with limited for single framework.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Visit: www.ijirset.com*

**Vol. 7, Issue 5, May 2018**

## REFERENCES

[1] X. Ding, P. B. Gibbons, M. A. Kozuch, and J. Shan, "Gleaner: mitigating the blocked-waiter wakeup problem for virtualized multicore applications," in USENIX ATC 2014, 2014, pp. 73–84.
[2] A. Da Zheng and A. S. Szalay, "A parallel page cache: Iops and caching for multicore systems," in USENIX Hot Storage 2012.
[3] M. Yui, J. Miyazaki, S. Uemura, and H. Yamana, "Nb-GCLOCK: A non-blocking buffer management based on the generalized CLOCK," in ICDE 2010, 2010, pp. 745–756.
[4] W. Wang, "Storage management for large scale systems," Ph.D. dissertation, Department of Computer Science, University of Saskatchewan, Canada, 2004. [30] L. Huang and T. cker Chiueh,
[5] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, "DULO: an effective buffer cache management scheme to exploit both temporal and spatial locality," in FAST 2005, pp. 8–8.
[6] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," ACM Trans.Comput. Syst., vol. 9, no. 1, pp. 21–65, 1991.
[7] X. Zhang, R. Casta ˜ neda, and E. W. Chan, "Spin-lock synchronization on the butterfly and KSR1," IEEE Parallel Distrib. Technol., vol. 2, no. 1, pp. 51–63, 1994.
[8] F. J. Corbato, "A paging experiment with the multics system," in In Honor of Philip M. Morse, Feshbach and Ingard, Eds. Cambridge, Mass: MIT Press, 1969, p. 217.
[9] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: An effective improvement of the CLOCK replacement," in USENIX ATC 2005.
[10] S. Bansal and D. S. Modha, "CAR: Clock with adaptive replacement," in FAST 2004, pp. 187–200.
[11] G. Glass and P. Cao, "Adaptive page replacement based on memory reference behavior," in SIGMETRICS 1997, pp. 115–126.
[12] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, "DULO: an effective buffer cache Management scheme to exploit both temporal and spatial locality," in FAST 2005, pp. 8–8.
[13] "DB2 for z/OS: DB2 database design," 2004. [Online]. Available:http://www.ibm.com/developerworks/db2/library/techarticle/dm0408whitlark/index.html
[14] A. Da Zheng and A. S. Szalay, "A parallel page cache: Iops and caching for multicore Systems," in USENIX HotStorage 2012.
[15] M. Blasgen, J. Gray, M. Mitoma, and T. Price, "The convoy phenomenon," SIGOPS Oper. Syst. Rev., vol. 13, no. 2, pp. 20–25, 1979.
[16] S. Jiang and X. Zhang, "LIRS: an efficient low inter-reference recency set replacement Policy to improve buffer cache performance," in SIGMETRICS 2002, pp. 31–42.
[17] X. Ding, P. B. Gibbons, M. A. Kozuch, and J. Shan, "Gleaner:mitigating the blocked-waiter wakeup problem for virtualized multicore applications," in USENIX ATC 2014, 2014, pp. 73–84.
[18] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management Replacement algorithm," in VLDB 1994, pp.439–450.
[19] W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza, and N. Mac-Naughton, "The oracle . Universal server buffer manager," in VLDB 1997, M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, Eds., pp. 590–594.
[20] X. Ding, S. Jiang, and X. Zhang, "BP-Wrapper: a system framework making any Replacement algorithms (almost) lock contention free," in ICDE 2009, pp. 369–380.
[21] Y. Zhou, J. Philbin, and K. Li, "The multi-queue replacement algorithm for second level buffer caches," in USENIX ATC 2001, pp. 91–104.
[22] The Open Source Development Laboratory, "OSDL – database test suite." [Online]. Available: http://old.linux-foundation.org/lab activities/kernel testing/osdl database test suite/
[23] Transaction Processing Performance Council, "TPC-W." [Online]. Available: http://www.tpc.org/tpcw
[24] Transaction Processing Performance Council., "TPC-C." [Online]. Available: http://www.tpc.org/tpcc
[25] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in MoBS 2009, June.