ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

A General Approach to Scalable Buffer Pool Management

Milind Madhukar Kulkarni, A.V.Mophare

Student, Department of CSE, N.B.N.C.O.E, Solapur, India

Asst. Professor, Department of CSE, N.B.N.C.O.E, Solapur, India

ABSTRACT: In high-end data processing systems, such as databases, the execution con -currency level rises continuously since the introduction of multicore processors. This happens both on premises and in the cloud. For these systems, a buffer pool management of high scalability plays an important role on system overall performance. The scalability of buffer pool management is largely deter-mined by the data replacement algorithm, which is a major component in buffer pool management. It can seriously degrade the scalability if not designed and implemented properly. The root cause is its use of lock-protected data structures that incurs high contention with concurrent accesses. A common practice is to modify the replacement algorithm to reduce the contention on the lock(s), such as approximating the LRU replacement with the clock algorithm or partitioning the data structures and using distributed locks. Unfortunately, the modification usually compromises the algorithm's hit ratio, a major performance goal. It may also involve significant effort to overhaul the original algorithm design and implementation. A general solution to improve the scalability of buffer pool management using any 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, design a system framework, called Buffer Pool that eliminates almost all lock contention without requiring any changes to an existing algorithm. In Buffer Pool, use a dynamic batching technique and a prefetching technique to reduce lock contention and to retain high hit ratio.Our proposed working is on optimal page replacement algorithm for improving performance.

KEYWORDS: Buffer Pool Management, Replacement Algorithm, Lock Contention, Multi-Core, Optimal Page Replacement Algorithm.

I. INTRODUCTION

Data processing systems, such as databases, mostly use high-end servers with many cores for high performance. In the cloud, an Amazon RDS database can have up to 32 virtual CPUs (vCPUs) on such a server, a large number of worker threads run simultaneously to maximize computing power of the system. With the high computing power, the overall system performance is frequently determined by how fast the worker threads can access the data they process. As a common procedure, a data-processing system maintains a buffer pool in the user memory space for its worker threads to cache the data sets that are actively accessed. The system carefully manages the buffer pool using experienced policies to minimize costly disk I/O operations.

Worker threads access the buffer pool synchronously at a very high frequency. With the increasing number of cores (or vCPU count), the buffer pool management must be highly scalable and efficient to effectively manage with the growing processing concurrency. Otherwise, it can become a serious system bottleneck, leading to significant performance reduction. In fact, that type performance problems have been widely observed in various systems, such as PostgreSQL, MySQL, and Oracle. The performance problem is caused by the contention on the locks used in the buffer management, particularly the lock that protects a core data structure used for data replacement technique. The technique makes decisions on which data pages should be cached in memory to absorb effectively requests for on-disk data. To implement the technique, the threads maintain a data structure to track their data-access history. Update the data structure in response to page accesses so that replacement decisions can be made based on that history recorded in the data structure. To guarantee the integrity of the data structure, the updates must be made in a serialized trend. Thus,

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

a lock is necessary to synchronize the updates. Lock contention occur when the lock is held by one worker thread and other threads must wait in the form of busy-waiting and/or context switches. Many replacement algorithms, techniques and their implementations have been proposed. They construct and manage deep page access history to maximize hit ratios and minimize the cost incurred by disk I/O operations. These algorithms, techniques usually take actions upon each I/O access, either a hit or a miss in the buffer, which mainly include a sequence of updates in the data structure to record the access 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.

II. LITERATURE SURVEY

Sr. No	Author/Paper Name	Proposed System	Refer Points
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.	The paper systematically 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 mitigation to this problem.	Study the Gleaner as a solution, which combines resource retention approaches with idling operations and consolidation scheduling.
2	A. Da Zheng and A. S. Szalay, "A parallel page cache: Iops and Caching for multicore systems," in USENIX Hot Storage 2012.	Present 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 no uniform memory architecture (NUMA).	Study NUMA by partitioning the cache by processor and using message passing to avoid remote memory access. The outcome is a system that tracks the scalable performance of direct I/O (no caching) for up to 48 cores, while preserving the hit rates of the Linux page cache.
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.	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 no blocking scheme for buffer fix operations that fix buffer frames for requested pages without locks by combining Nb-GCLOCK and a non-blocking hash table.	Study a non-blocking buffer management scheme based on a lock-free variant of the GCLOCK page replacement algorithm.
4	W. Wang, "Storage management for large scale	A self-tuning algorithm is proposed to automatically tune the page cleaning	Study self-tuning algorithm to automatically

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

	systems," Ph.D. dissertation, Department of Computer Science, University of Saskatchewan, Canada, 2004. [30] L. Huang and T. cker Chiueh,	activity in the buffer cache management algorithm by monitoring the I/O activities of the buffer cache. This algorithm achieves performance comparable to the best manually tuned system.	tune the page cleaning activity in the buffer cache management.
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.	Propose a scheme called DULO (DUal LOcality), which exploits both temporal and spatial locality in buffer cache management. Leveraging the filtering effect of the buffer cache, DULO can influence the I/O request stream by making the requests passed to disk more sequential, significantly increasing the effectiveness of I/O scheduling and perfecting for disk performance improvements.	Study a new and effective memory management scheme, DULO, which can significantly improve I/O performance by exploiting both temporal and spatial locality.

III. METHODOLOGY

Background on 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.

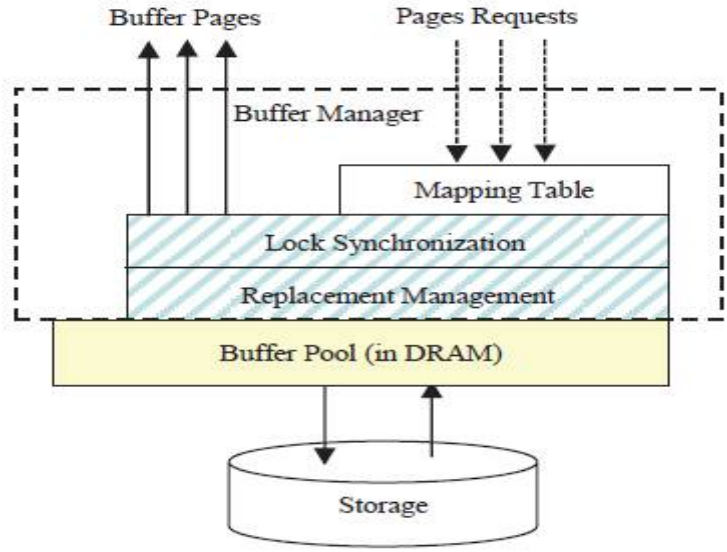


Fig.1. The diagram of a buffer manager.

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.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

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. 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. 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 (1) the replacement management may use one lock for its entire data structure, which is a single point of hot spot, and (2) 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 multicore/multiprocessor system.

IV. PROPOSED METHODOLOGY

This paper 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.

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.

V. PROPOSED SYSTEM ARCHITECTURE

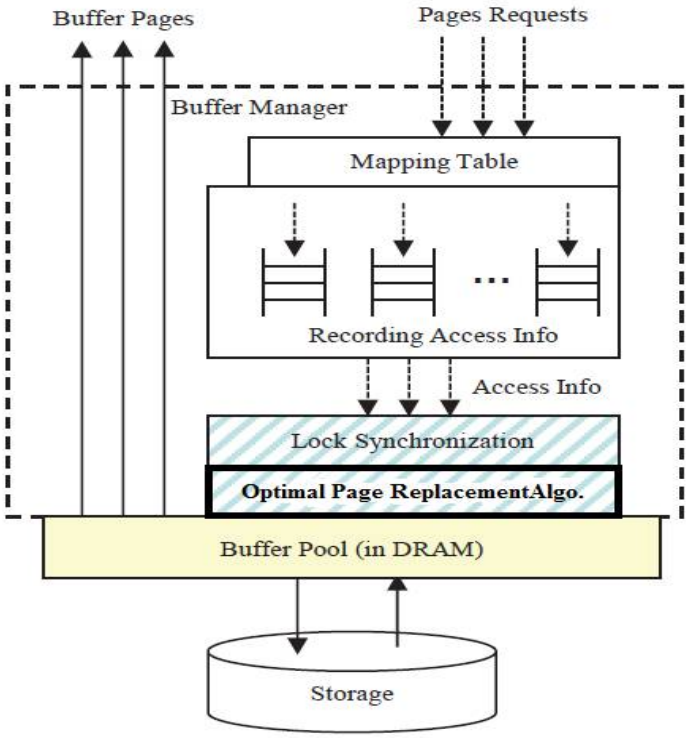


Fig2. Proposed System Architecture

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

A. OPTIMAL PAGE REPLACEMENT ALGORITHM

The best possible page replacement algorithm is easy to describe but impossible to implement. It goes like this. At the moment that a page fault occurs, some set of pages is in memory. One of these pages will be referenced on the very next instruction (the page containing that instruction). Other pages may not be referenced until 10, 100, or perhaps 1000 instructions later. Each page can be labeled with the number of instructions that will be executed before that page is first referenced.The optimal page algorithm simply says that the page with the highest label should be removed. If one page will not be used for 8 million instructions and another page will not be used for 6 million instructions, removing the former pushes the page fault that will fetch it back as far into the future as possible.The optimal page algorithm simply removes the page with the highest number of such instructions implying that it will be needed in the most distant future. This algorithm was introduced long back and is difficult to implement because it requires future knowledge of the program behavior. However it is possible to implement optimal page replacement on the second run by using the page reference information collected on the first run.

B. ADVANTAGES OF OPTIMAL PAGE REPLACEMENT ALGORITHM:-

i) Lowest page fault rate.
ii) Never suffers from Belady's anomaly.
iii) Twice as good as FIFO Page Replacement Algorithm.

C. DISADVANTAGES OF OPTIMAL PAGE REPLACEMENT ALGORITHM:-

i) Difficult to implement.
ii) It needs forecast i.e. Future knowledge.

In this algorithm, pages are replaced which are not used for the longest duration of time in the future.
Let us consider page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 and 4 page slots.
Initially all slots are empty,
so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
When 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will take place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.
Optimal page replacement is perfect.

VI. EXPERIMENTAL SETUP

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. In this section, we evaluate the overall performance of three systems *pgClock*, *pg2Q*, and *pgBat-Pre* on the Power Edge 1900 using 8 cores when we change the buffer size from 32MB to 1024MB, and let the systems issue direct I/O requests to bypass the operating system buffer cache. As the data set sizes of *DBT-1* and *DBT-2* are 6.8GB and 5.6GB respectively, not all the accesses can be satisfied from the buffer.

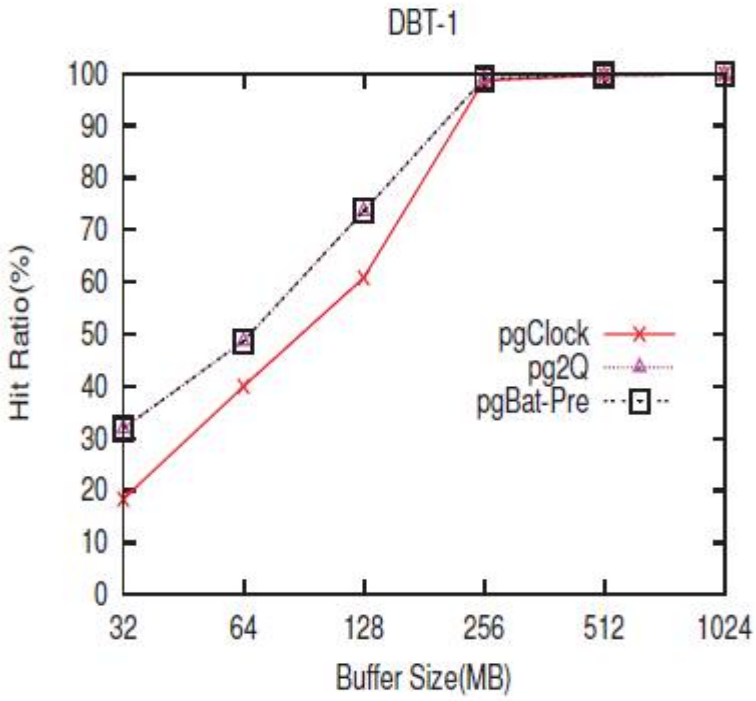
ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

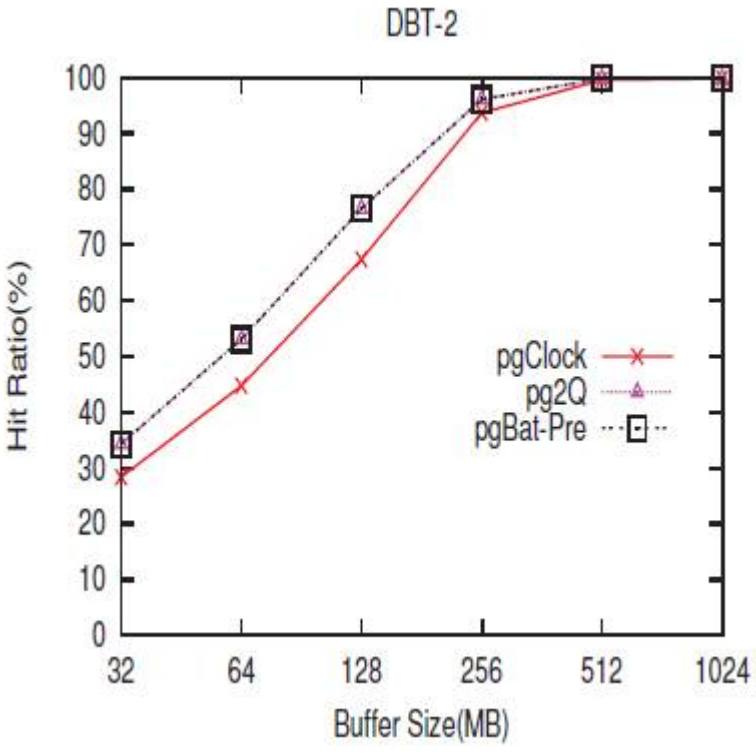
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017





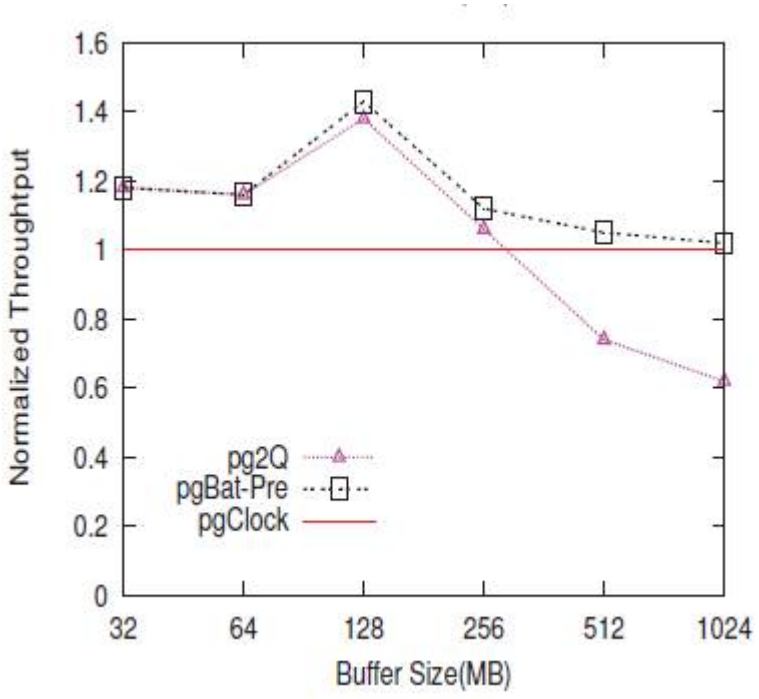
ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

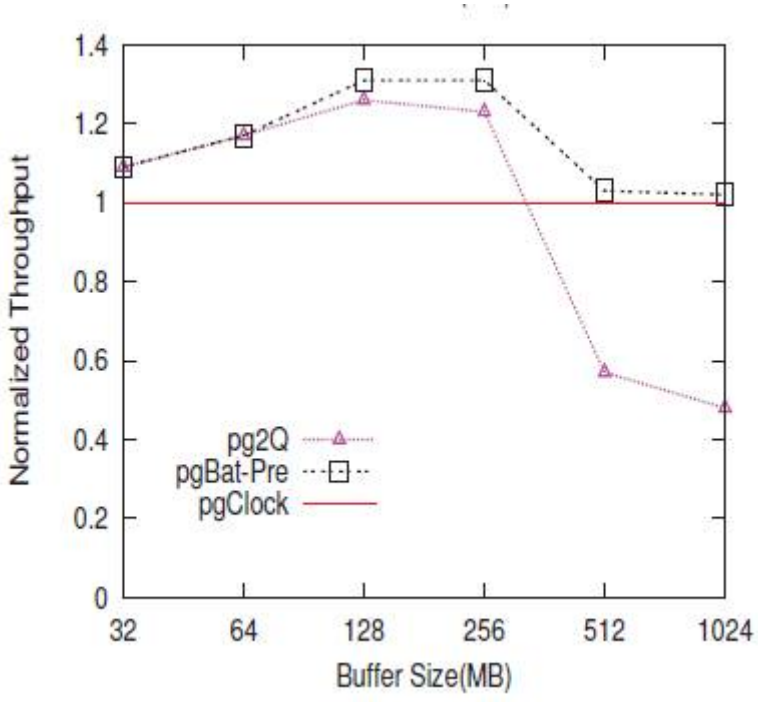
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017





ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

VII. CONCLUSION

In this paper, we address the scalability issue due to lock contention in the implementation of replacement algorithms for the management of buffer cache. We proposed an efficient and scalable framework, optimal page replacement algorithm for page replacement. Without algorithm modification, the performance advantage of the original replacement algorithms will not be compromised, and human effort is also minimized. Existing approach used the technique BP-wrapper, in which the dynamic batching and the prefetching techniques can be used with any replacement algorithms without modification of the algorithms. Without algorithm modification, the performance advantage of the original replacement algorithms will not be compromised, and human effort is also minimized. The only cost of the framework is a small FIFO queue for each transaction-processing thread, which keeps the thread's most recent access information. So our propose work will use the Optimal Page Replacement Algorithm which is the best possible page replacement algorithm.

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.