Improving Buffer Cache Hit-Ratio – Enhance Performance

[1]M. Muni Babu, [2]K. Mamatha
[1] Adhoc Lecturer, Dept. of CSE, JNTUA College of Engineering (Autonomous), Pulivendula,
[2] M.Tech, Dept. of CSE, JNTUA College of Engineering (Autonomous), Pulivendula,

Abstract: The usage of buffer cache improves the system performance and throughput when buffer cache utilized very effectively. For effective use of buffer cache having many techniques in those pattern based methodology works very efficiently and enhance the system performance. Future block reference is identified from the detected pattern. Marginal gain functions are used for managing the partitions. This proposed methodology main aim is improve the buffer cache hit ratio used to enhance the performance.

Keywords:-Buffer Cache, Cache Partition, Replacement policies.

I. Introduction

The operating system attempts to minimize the frequency of disk access by keeping a pool of internal data buffers, called the buffer cache. The buffer cache is used to improve the system performance and minimize the throughput of the system. During system initialization, the operating system allocates space for a number of buffers, configurable according to memory size and system performance constraints. A buffer consists of two parts: a memory array that contains data from the disk and a buffer header that identifies the buffer. The data in a buffer corresponds to the data in a logical disk block on a file system, and the operating system identifies the buffer contents by examining identifier fields in the buffer header. The operating system caches data in the buffer pool according to a "Least Recently Used Replacement algorithm ". It is one of the mostly used replacement algorithm due to its effective utilization of principle of locality. Due to simple management and decrease complexity and simple implementation current GNU/Linux kernel still implements LRU (Least Recently Used) replacement algorithm. But LRU is dependent on the application working set and suffers from degraded performance when weak locality of reference is exhibited by the application working set having a large size compared to the cache size [7-10]. To enhance performance gain and improved cache hit ratio the information in accesses made to the data blocks through the I/O request are used by recent replacement algorithm. The operating system maintains a free list of buffers that preserves the least recently used order [1-10].

A. Buffer cache organization Here level 1 represents the buffers used to read the data values and level 2 registers used at the time of computation as temporary storage and level 3 cache used to store data temporarily like this memory hierarchy works with different levels at the time of computation fig1.

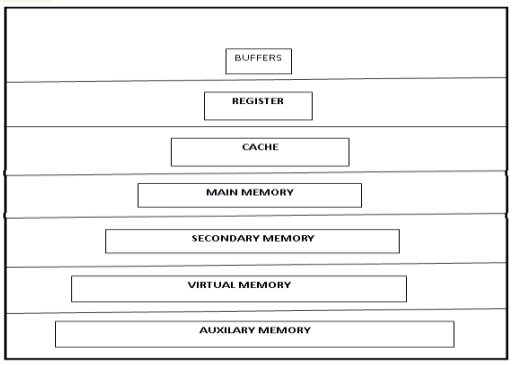

Fig 1: Memory Hierarchy

Memory hierarchy is need for:

1. Increasing of accessing of words.
2. Increasing performance in between memory and processor.
3. Speed access in the time of computation.

The memory hierarchy levels are used in different situations based on input given by user [11-15].

The different fields of buffer cache are as follows:

• **Device Number**: It keeps the records of the devices in the system. Each partition or file system defined during the installation of operating system are identified as Unique devices.

B. Inside the buffer cache: The buffer header contains a device number field and a block number field that specify the file system and lock number of the data on disk and uniquely identify the buffer. We already discussed for efficient management the buffer cache is divided into header and data area it is depicted in "Fig 2".

• **Block Number:** Unique block referenced in the device is determined by the block number.

• **Status Field:** This field is referred by the processor to verify the status of the buffers in the cache.

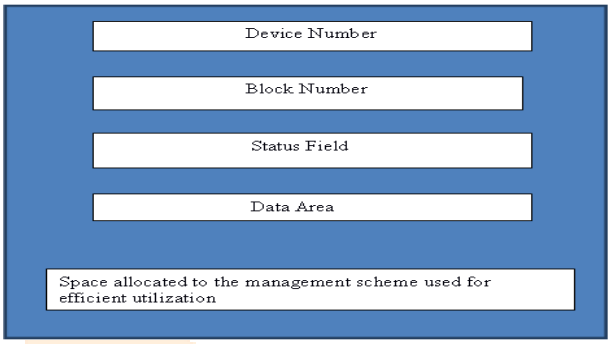

Fig 2: Inside the buffer cache

• **Data Area:** The requested data blocks are fetched from the devices and are placed into the data area of the buffer cache. The size of the data area is same as the block size of the devices.
• **Pointers:** The remaining space is utilized by the data structure or management techniques used for efficient buffer cache utilization to maintain their pointers and information.

Status of a buffer is a combination of the following conditions:
1. The buffer is currently locked.
2. The buffer contains valid data.
3. The kernel must write the buffer contents to disk before reassigning the buffer; this condition is known as "delayed write".
4. The kernel is currently reading or writing the contents of the buffer to disk.
5. A process is currently waiting for the buffer to become free.

Use of a cache between the user process and the file system: To reduce the number of I/O operations through peripheral devices, the kernel maintains a cache of memory dedicated for all block input and output (disk I/O and so forth).

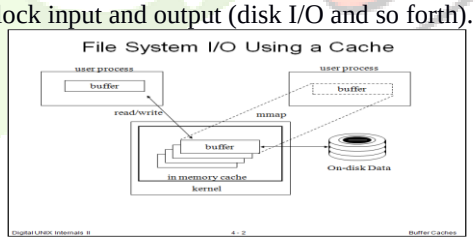

Fig 3: Buffer Cache for File I/O

When accessing file data, the kernel tracks the file and the logical block containing the data. When reading from the disk, the kernel attempts to get the block from the buffer cache. If the block is already in the cache, the kernel does not have to access the disk. If the block is not in the cache, the kernel reads the block from disk and stores it in the cache. When writing to the disk, the kernel checks to see if the data block is in the buffer cache. If the block is in the cache, the kernel updates it, and does not have to access the disk Fig 3.

Advantages and Disadvantages of Buffer Cache:

Advantages:
1. The use of buffers allows uniform disk access, because the kernel does not need to know the reason for the I/O.
2. The system places no data alignment restrictions on user process doing I/O, because the kernel aligns data internally.
3. Use of the buffer cache reduces the amount of disk traffic, thereby increasing overall system throughput and decreasing response time.
4. The buffer algorithm help ensure file system integrity, because they maintain a common, single image of disk blocks contained in the cache.

Disadvantages:

1. The Reduction of disk traffic is also most important for good throughout and response time, but the cache strategy also introduces several disadvantages.
2. Usage of the buffer cache requires an extra data copy when reading and writing from user processes [1-3].

II. EXSISTING METHODS FOR CACHE PARTIOINING

To partition the buffer cache we use different types of methods those are:

1. Static partitioning
2. Dynamic partitioning
3. Marginal gain functions.

Static partitioning: It is a predefined partitioning technique. A static cache partitioning technique divided the cache memory into pre -fetch buffer and cache buffer. The technique was not able to cope with reduces cache hit rates due to reasons as limited size of the buffer and pre-fetch cache, non -redundant pattern of file access, large working sets [9-11].

Dynamic partitioning:
This technique is not predefined. Cost-benefit analysis partitioned the aggregate cache. By using the probabilities of blocks future access for weighing the cost of ejecting or allocation a block to cache partition. To take maximum benefit of the available cache memory dynamic cache management techniques were suggested. One of Such technique is SA-W2R [13-15]. SA-W2R integrated buffer management and constant prefetching allowed the integration of other replacement policies in the modular approach. APACS (An Automatic Prefetching and Caching System) used .The hit ratio of the cache buffer and pre-fetch buffer partitions of the main memory as an indicator for managing the cache partition by taking into account the full context information about the I/O access patterns [11-13].

Marginal Gain Functions: To address the problem of allocating the data blocks among the various cache partition, a widely known concept of resource allocation strategies, marginal gain is used. It is defined by the following equation

MG(n)~~ Hit(n)-hit(n-1)

Where **MG(n):** Determines the expected number of extra buffer hit per unit time that would be obtained by increasing the buffer size from (n-1) to (n). Hit(n) : With n available buffers it is the expected number
of buffer hit per unit time.

III. OVERVIEW OF BUFFER CACHE MANGEMENT TECHNIQUES [5-10]

To improve performance of buffer cache different techniques are available each one has its own pro's and con's.
The techniques are::

1. Reference pattern based technique.
2. Reuse distance based technique.
3. Pre-fetching based technique.

Reference pattern based technique:

Future block reference is identified from the detected patterns. This improves the buffer cache management schemes. Basically pattern based buffer cache management technique is being discussed for enhancing the system performance. The technique determines the victim block to be replaced by identifying the I/O access patterns. The blocks are stored in separate, pattern based partitions of the cache, which supports multiple replacement policies to best utilize the cache under that reference patterns. As the suggested technique aims at improving the buffer cache hit ratio it could be used for enhancing the performance of multimedia application and can be useful for power saving in database server as increased cache hit ratio reduces the memory traffic and thus saves the energy.
Heterogeneous multicore processors |deals with **complexity of multimedia application**s in mobile devices gives Fast **generation addresses** generates Simple addressing patterns frequently **Memory allocation method** deals with **data transfer** using temporary and spatial locality.

Access Patterns are used to reduce overall communication delay. Offer low response time done by large capacity, reasonable cost hard disk and NAND flash memory as 20 devices.
.Effectively optimized and enhances the system performance.

Effective utilization of cache: Filters are used to prevent reuse behavior of blocks (blocks must be efficient). When address of missing cache blocks occurred. At **Block level**, SEQ technique used to handle long sequence page cache with MRU policy. At **File level,** Unified Buffer Management used with reference mode. At **application level**, DEAR technique used which is accurately classified the IO Access pattern. At **Program context level**, program counter based classification and Adaptive multi policy caching to enhance cache performance.

2. Pre Fetching Based Techniques Used to **avoid delays related to disk access,** data blocks are read in advance and kept in main memory. **Predictive pre fetching**: Implemented at **file level** (APACS) by employing dynamic cache partitioning, pipelining and pre-fetch buffer memory. Past behavior of IO request, explored from system calls. **Instructive pre-fetching:** Based on **hints** inserted by compiler or programmer.

3. Reuse distance based Techniques "Time interval difference between two consecutive references".

- **R**elated to memory access instructions
- For better optimization of cache by Replacement Strategies.
- To overcomes drawbacks of LRU by making use of instruction PC.
- Ship (Signature Based Hit Predictor) used to enhance cache performance.

IV. PROPOSED PATTERN BASED METHODOLOGY FOR BUFFER CACHE MANAGEMENT

Predicts **Block Access Pattern** from IO request by analyzing past access behavior done by

1.3-phase Identification process Detection module focused initially, block is accessed by IO requests, DATA STRUCTURE fields used for identification during access Threshold used to avoid conflicts and distinguish between patterns

2. Types of block access pattern .**Sequential Reference:** Blocks referenced only once and never revisited. **Looping Reference:** Blocks referenced at regular intervals. **Others:** None of patterns found. **Thrashing:** allocating > available

3. Data structures: Used for pattern identification and referenced by pattern detection phase done by Pc hash table: maintained 2 counters namely Register (contain information) and Reuse (how many time it used) for recording and analyzing pattern. File hash table: Keeps records of sequential of block referenced and update.

4. **Marginal Gain Value for Identification Pattern** Sequential Reference: only one block is allocated.
Looping Reference: loop length(li) ,period(lp) and buffers(n) **MG (n loops)=1/pi**; if n<=li..
Otherwise 0.
Others**: MG(n)=Hit(n)-Hit(n-1**).

V. PATTERN IDENTIFICATION BY PROPOSED MATHODOLOGY

Used for **Pattern Identification** by exploiting from **file or program context level.** Implemented detection module sets **threshold=3. Initially**-blocks are classified as **others.** After crossing threshold- grouped as **sequential.** If referenced again- **looping.**

others	sequen tial	Looping			
O	S	O	L	O	L
O	S			L	S

Conclusion

As the suggested methodology considers the information gained from the input/output request while making the replacement decision it improves the buffer cache hit ratio and reduces the time elapsed in performing the I/O. System performance is thereby enhanced due to the block pattern identification and pattern based partitioning of the buffer cache. The movement of the block among the partitions and space allocated to each partition are handled dynamically through the low over head marginal gain function.

REFERENCES

[1] Yang, Qiang, and Zhen Zhang. "Model based predictive prefetching." In Database and Expert Systems Applications, Proceedings of 12th International Workshop, IEEE 2001, pp. 291-295.

[2] M. J. Bach, The design of the UNIX operating system. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[3] A. S. Tanenbaum and A. S.Woodhull, Operating Systems Design and Implementation. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[4] Butt, A.R.Ginady, "The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithm", IEEE Transaction on Computers,Vol. 56, no 7, pp 889-908, Jul. 2007.

[5] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L.Min, Y. Cho, and C. S. Kim,"A low-overhead, high-performance unified buffer management scheme that exploits sequential and looping references," in 4th Symposium on Operating System Design and Implementation (OSDI), Oct. 2000, pp.119–134.

[6] J. Choi, S. H. Noh, S. L. Min, and Y. Cho, "An implementation study of a detection-based adaptive block replacement scheme," in Proceedings of the 1999 USENIX Annual Technical Conference, Jun. 1999, pp. 239–252.

[7] C. Gniady, A. R. Butt, and Y. C. Hu, "Program-counter-based pattern classification in buffer caching." in Proceedings of 6th Symposium on Operating System Design and Implementation (OSDI), Dec. 2004, pp. 395–408.

[8] F. Zhou, R. von Behren, and E. Brewer, "AMP: Program context specificbuffer caching," in Proceedings of the USENIX Technical Conference, Apr. 2005.

[9] Prof.P.K. Biswas, "Lecture Series on Digital Computer Organization," Internet: http://nptel.iitm.ac.in, Sep 2009 [Aug 12, 2012].

[10] Reusang, P. Park, S.K. Jeon "Reducing Cache Pollution of Prefetching in small data cache," Proceeding of the International Conference On Complier Design, Austin, Tx, Sep 2001, pp 0530.

[11] J.Choi, S. Cho, "Analytic Prediction of buffer Cache Hit Ratio", IEEE letters, 36(1):10-11, 2000.

[12] Vellanki, V. Chervenak, "A Cost Benefit Scheme for High Performsnce Preditive Caching", Proceeding of ACM/IEEE SC99 Conferenence, 1999.

[13] Jeon. H.S. "Practical Buffer Cache Mangement Scheme Based on Simple Prefeteching",IEEE transaction on Computer Electronics, Vol 52, No 3, pp. 276-285, Aug. 2006.

[14] Lewis, Joshua, Mohammed Alghamdi, M. A. Assaf, Xiaojun Ruan, Zhiyang Ding, and Xiao Qin. "An automatic prefetching and caching system." In Performance Computing and Communications Conference pp. 180-187. IEEE, 2010.

[15] Park, Seung-Ho, Jung-Wook Park, Shin-Dug Kim, and Charles C. Weems. "A pattern adaptive NAND flash memory storage structure." Computers, IEEE Transactions on 61, no. 1, pp:134-138, 2012.