International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-1, November 2019

Hybrid LRU Algorithm for Enterprise Data Hub

A. Murugan, S. Ganesan

Abstract: ***In the computing theory, cache is key concept to process the in memory data from slow storage layer into faster. Key objective is to speed up the end user requests from cache storage. As cache is limited in size, it is essential to build the efficient algorithm to replace the existing unused content from faster memory. Enterprise Data Hub makes a single golden storage to produce any kind of reports at any point of time from the various sources of any system. Data integrity and governance parameters add the credibility to this centralized data. In general, Big Data processing uses the disk based technique to handles the business logic processing. This research paper is to leverage the in memory processing for the business use case of Enterprise Data Hub. This paper provides an algorithm to handle Enterprise Data Hub in efficient design using prioritized Least Recently Used algorithm. This paper depicts about the experimental advantage of execution time optimization and efficient page/cache hit ratio, using hybrid Least Recently Used algorithm with priority mechanism. It helps the industry Enterprise Data Hub for the faster execution model.***

Index Terms—Prioritization, Least Recently Used, Big Data, Enterprise Data Hub, Cache Algorithm, Hybrid LRU, Cache Hit Ratio, etc.

I. INTRODUCTION

Data is foundation of the computer industry. Without data and storage processing, any enterprise system doesn't product the business value by extracting the vital information and knowledge with detailed analyses and performance engineering. In the rapid transformation of Information Technology, enterprise data growth is phenomenal. Enterprise Data Hub is a solution to build and maintain the golden records of any enterprise as shared trustable enterprise data. Most of Big Data solutions are based on disk based data processing and in-memory on need basis. This paper depicts about the efficient way of in-memory caching to handle the data in the most effective logic. Key logic of this paper is to build the hybrid LRU algorithm with parameterized data.

II. LITERATURE REVIEW

A. Enterprise Data Hub

When bank A acquires bank B, obviously, their customer data are tending to duplicate in two distinct stores.

Revised Manuscript Received on November 30, 2019.
***** Correspondence Author
S. Ganesan*, Research Scholar, PG & Research Department of Computer Science, Dr. Ambedkar Govt Arts College, University of Madras, Chennai, INDIA Email: Ganesan.Senthilvel@gmail.com
A. Murugan, Associate Professor & Head, PG & Research Department of Computer Science, Dr. Ambedkar Govt Arts College, University of Madras, Chennai, INDIA Email: amurugan1972@gmail.com

It is highly expected to link all of enterprise critical data to one common store. Enterprise Data Hub (EDH) is the proposed solution which is meta data driven [1] central data repository for any enterprise with open sourced Big Data tools and techniques [4] like HCatalog (Reference Data), Data Wrangler (cleansing), Hadoop Distribution File System, Map Reduce, Hive (distributed process).

The final output of this framework is Master data for any enterprise system.

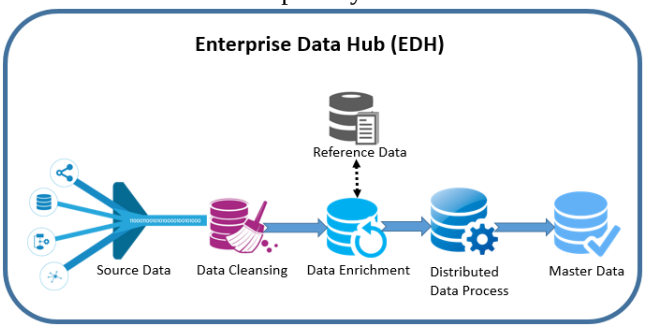

Fig. 1. Architecture of EDH

As Technology needs to enable the business, EDH [7] makes a single shop store to produce any kind of reports at any point of time Data integrity and governance adds the credibility to the store data.

B. Caching Concepts

The word 'Cache' is derived from the concept of 'Storage'. Cache memory is one type of system internal memory, which is used as an interface between Central Processing Unit and Random Access Memory with the main purpose to process the recently used data in a faster way. It is used for a faster access to 'frequently/recently used' data, by avoiding the data transfer done through system motherboard bus.

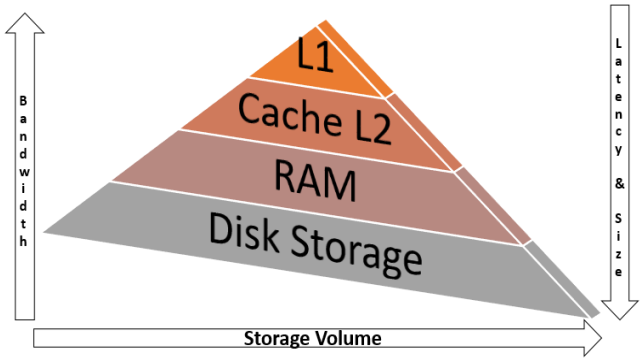

Fig. 2. System Storage Levels & Cache Value

In this paper, the approach is to use replacement policies (cache algorithms), which will choose the element to remove from the cache when we need space for a new element.

Prior to that, it is essential to replace the identified element under replacement policy and followed by removal from the cache. This process is termed to evict the element from the cache. When we get a request to an element, it checks whether the element is stored in the cache. If the element is in the cache, it is termed as a cache hit. Contrarily, cache miss occurs to fetch this element from the slow memory.

It means the optimized frequently used cache data is leveraged for a while without reaching cache miss scenario. Parameterized concept is the key factor of efficient cache hit ration.

C. Various Caching Algorithms

Computer Industry evolves with multiple Cache replacement algorithms [8], based on optimal and practical implementation. This paper highlights Top-5 key cache-replacement industry algorithms.

1. Optimal Cache algorithm (OPT)
2. Least Recently Used (LRU)
3. Least Frequently Used (LFU)
4. Adaptive Replacement Cache (ARC)
5. Low Inter-reference Recency Set (LIRS)

As name stands OPT, it was the first recognized optimal caching logic, invented by Belady in 1966. Core logic is to replace the existing element, which does not have the probability of getting cache request for the longest period in the processing cycle.

LRU was an improvement version of the earlier OPT by leveraging the recently used data in the near future. It was designed in 1965 to replace the existing elements in the cache memory, based on the least recently used concept.

LFU was designed in 1971 to replace the existing elements, based on degree of the workloads frequency. In this model, it is essential to keep track of the usage frequency for each element in the cache storage.

ARC is the combination of LRU & LFU concepts. It has two parameters namely recency and frequency, out of which the efficient method is adapted for the given workload.

Post millennium, LIRS was developed in 2002 with the concept of the reuse distance logic by avoiding the recency factor. Reuse distance is nothing but the distance between the last request and the second last request for a given element in the cache storage.

D. Proposed Solution

This paper provides an efficient algorithm to handle enterprise data hub using prioritized LRU (Least Recently Used) algorithm. Big data processing is proven to perform faster using in-memory based approach, instead of disk based processing.

Implementation depicts about the experimental advantage of execution time optimization and efficient page/cache hit ratio, using hybrid LRU algorithm with priority mechanism.

III. CACHE ALGORITHM USED

A. Least Recently Used

In Big Data processing, it is necessary to retain the least recently used data for further processing. It is the reason to baseline LRU in Enterprise Data Hub system. In 1965, Least Recently Used algorithm (LRU) cache replacement algorithm was designed to reuse the recently used in the near future. LRU inspired by its predecessor Optimal Cache (OPT) algorithm. Key differential factor of LRU is about online processing against OPT's offline mode.

B. Priority Queue

Computing theory depicts the impact of using different priority queues in the system performance. In general, priority queues can handle any type of keys like integer, float, etc.

Historical experiment results are evident to demonstrate the performance improvement using cache-efficient priority queues. The core concept is by having significant overhead in the constant factor and so it performs the best. Priority Queue performance [4] metric is

Table- I: Priority Queue Models

#	Priority Queue Model	Insert/Delete(min)
1	Sequence Heap (Cache aware)	$O(\log_4 N)$
2	Bottom up Binary Heap (worst case)	$O(\log_2 N)$
3	Aligned 4-ary Heap (worst case)	$O\left(\frac{1}{B}\log_k \frac{N}{l} + \frac{1}{k} + \frac{\log}{l}\right.$

N:queue size, B: block size, M: cache size, l: θ(M), k: θ(M/B)

Priority algorithm is highly efficient to compute the shortest paths in a graph with non-negative edge-weights. Key execution concept of the algorithm, is the efficiency of the heap using priority queue. In this paper, an experimental study depicts on how the heap affects performance.

C. Hybrid LRU Proposal

Area of improvement of LRU, is to reuse the elements, which are more frequently and prioritized usage than others. The objective of this paper is to build the hybrid LRU with the combination of priority queue and LRU concepts.

IV. TECHNICAL IMPLEMENTATION

A. System Environment

The experiments run on a 3-node cluster with four cores, 16GB of RAM running Ubuntu 14.04 operating systems. All cache algorithms were implemented using .NET Core Compiler of version 3. In terms of Enterprise Data Hub eco system, Cloudera Distributed version 5.2 has been setup. This experiment environment is minimum viable platform to prove the algorithmic concept and its value. We measure the performance in term of execution time as well as cache hit ratio between LRU and hybrid cache models.

B. Least Recently Used

In terms of algorithmic step, the logical flow goes as below.

Pseudo code of LRU algorithm is represented as:

```
Let Capacity = memory size to hold the number of pages
Let Current_Set = Pages in the current memory set
function PageAlgorithm_LRU()
{
   If (Memory access of the given page is success) Then
   {
      Retrieve the content to process
   }
   Else
   {
      Call PagesTraversal() function
   }
}

function PagesTraversal ()
{
   Assign indexes as empty map of pageIndex;
   Assign pageFaults as zero;
   If (Current_Set < Capacity) Then
   {
      Do
      {
         Insert Current_Page into Current_Set;
         Increment Current_Page;
         Incremen Current_Set;
      }
      While (Current_Set == Capacity);
      Insert Current_Page index into indexes map;
      Increment pageFaults;
   }
   Else
   {
      If (Current_Page in Current_Set)
      {
         No action;
      }
      Else
      {
         Do
         {
            LRU_Page = Page from Current_Set;
         } While (Current_Set == Capacity);
         Assign Current_Page = LRU_Page;
         Insert Current_Page index into indexes map;
         Increment pageFaults;
      }
   }
   Return page faults.
}
```

The above algorithm is illustrated here with a sample data set to prove. In our sample, we have the page references namely 7 0 1 2 0 3 0 4 2 3 0 3 2. Size of page slots are defined as 4 and so all 4 page slots are empty initially. Naturally, the first 4 pages named 7 0 1 2, are allocated to the available empty slots with 4 page faults. During the next cycle, the following page 0 is supposed to be loaded into the memory. Fortunately, page 0 is already available and so 0 page fault. On processing the upcoming page 3, it will remove the first in the list – page 7 to load page 3. It is termed as least recently used with 1 page fault. Third 0 is marked as 0 page fault, as it is already loaded in the memory. This flow is graphically depicted as below:

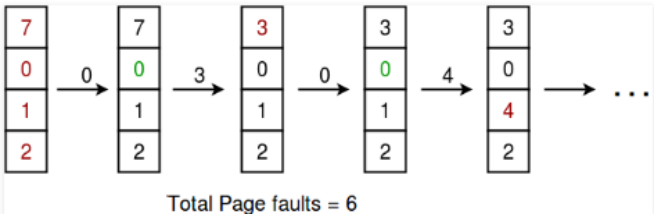

Fig. 3.LRU Technical Implementation Example

In terms of technical implementation, the algorithm uses of a binary search tree for each set, represented by bit flags. In the binary decision tree, each bit is representation of the related branch.

1: Left node has been referenced more recently than right
0: vice-versa

Based on the value of flags, tree traversal is executed. During this traversal, node flag is set to denote the direction, which is opposite to the taken direction. Sample execution is represented in the below diagram.

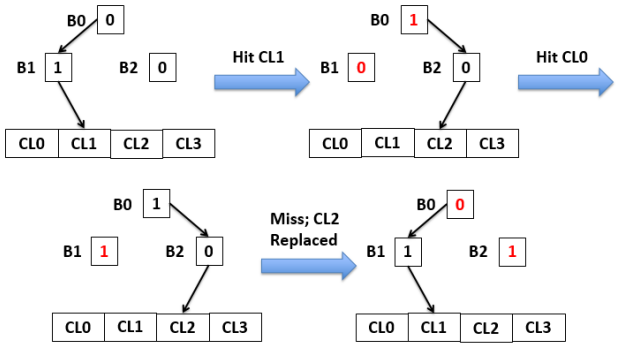

Fig. 4.LRU Technical Implementation Example

C. Proposed Hybrid LRU

An improvement of the existing LRU is termed as Hybrid LRU, in which parameterized priority queue logic is extended to obtain the execution performance benefits. On comparison with the normal queue concepts, Priority Queue has the below properties.

1. Priority is associated with each item.
2. Higher priority element is dequeued before the lower priority element.
3. In the event of the same priority, their queue order will take the priority.

A typical priority queue supports three operations. 1) push is to insert an item with given priority. 2) pop is to fetch and remove the highest priority item 3) peek is to get the highest priority element in the queue without removing it. Pseudo code of the logic will be:

```
// Node class contains data and priority index
public class NodeObject
   {
      public int dataStore;
      public int priorityNum;
      public NodeObject next;
   }
```

```
NodeObject function newNode (int d, int p)
{
   NodeObject temp = new NodeObject();
   Assign parameter d into temp.dataStore;
   Assign parameter d into temp.priorityNum;
   Assign null to temp.next;
   return temp;
}

int function peek(NodeObject head)
{
   return (head).dataStore;
}
NodeObject function pop(NodeObject head)
{
   Assign temp NodeObject as head;
   (head) = (head).next;
   return head;
}
NodeObject function push(NodeObject head, int d, int p)
{
   Assign start NodeObject as head;
   Assign temp NodeObject  as newNodeObject(d, p);
   If ((head).priorityNum > p)
   {
      temp.next = head;
      (head) = temp;
   }
   Else
   {
      While (start.next != null &&
            start.next.priorityNum < p)
      {
         Move start into start.next;
      }
      Assign temp.next as start.next;
      Assign start.next as temp;
   }
   return head;
}
```

Key differentiator of a priority queue is processing by the given priority instead of traditional first in first out model. In terms of data structure, it is an abstract data type to capture the idea of a container, in which elements are attached with priorities value. In essence, an element with highest priority is positioned at front of the queue. On removal of the element, the processing algorithm picks up the next highest priority element to the front.

In terms of technical implementation, LRU element is identified with the combined logic of parameterized priority queue element. In real world example [5], New York Stock Exchange equities pricing is cached using LRU logic for nearby usage. This paper improves the caching logic with the given parameter like trade price, volume, etc., as the priority element of cache replacement logic.

V. EXPERIEMENTAL RESULT

A. Data Analysis

On experimenting, the below data set was executed with multiple cycle against traditional LRU and hybrid LRU algorithms.

In caching logic, worst-case scenario is defined by always requesting the page, which was just evicted. Best-case scenario is defined by high retention rate of requesting page with less Cache miss.

B. Execution Time Optimization

Execution time is measured by the time taken to fetch the required element from the cache store either cache hit or miss.

Table- II: Experimented Result – Execution Time

Algorithm vs Data Volume	Data Set Execution (milliseconds)			
	10	150	1,300	10,500
Traditional LRU	1	1.7	4.2	8.6
Hybrid LRU	2.1	3	6.9	12.5

Data points are graphically represented as below:

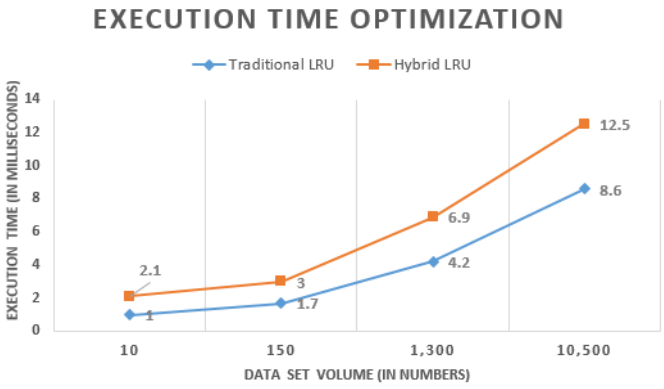

Fig. 5.Execution Time in milli seconds against data set volume

Best-case execution time is faster than the worst-case scenarios because of the delay to process the missing page.

C. Efficient Cache Hit Ratio

A cache hit occurs when an element is requested from a cache and the cache is able to fulfill that request. As an example, an end user visits a webpage, which is supposed to display an image. The client application (browser) sends a request to web server for loading the image in the browser.

If web server has a copy of the requested image in its cache store, then the request is termed as cache hit, and the requested image is sent to the browser.

A cache miss is when the cache does not contain the requested content and so new request is submitted to the origin server.

During this condition, the server will cache the image once the origin server responds, so that additional requests for it will result in a cache hit.

This paper covers the cache hit of the experiment results as below:

Table- III: Experimented Result – Cache Hit Ratio

Algorithm vs Cache Hit Ratio	Cache Hit Ratio (%)			
	10	150	1,300	10,500
Traditional LRU	99	98	94	87
Hybrid LRU	99.8	99	98.5	94

Cache hit ratio is a measurement of how many content requests a cache is able to fill successfully, compared to how many requests it receives

The formula for calculating a Cache Hit Ratio, is

$$Cache\ Hit\ Ratio = \frac{number\ of\ cache\ hits}{(number\ of\ cache\ hits + number\ of\ cache\ misses)}$$

Experience results of cache hit efficiency, is depicted in the below chart:

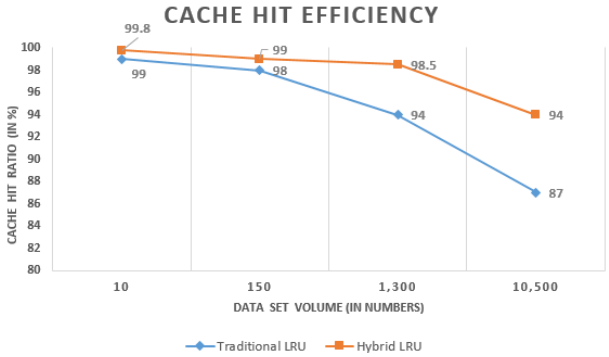

Fig. 6.Cache Hit Ratio against given data set

By experiment results, the paper shows the better execution time and cache hit ratio in Hybrid LRU than the traditional LRU algorithm.

VI. CONCLUSION

This paper describes an improved LRU algorithm, called Hybrid LRU. The core logic of LRU hybrid function is used to execute parameterized priority queue. It helps to determine the object in an optimum mode to remove from cache. On comparison of it with existing LRU algorithm, this proposal enables to obtain and prove two key benefits. One is a significant **~30% decrease of the execution time** to extract data from cache store during object cache extraction process. Second benefit is to obtain an efficient **cache hit ratio about ~10% higher** than traditional LRU algorithm.

This research paper concludes the power of Hybrid LRU algorithm, using the experimental results. Therefore, enterprise data hub can leverage these two benefits for its in-memory big data processing.

REFERENCES

1. Ronald J. Dovich and Peter J. Eppele, "Metadata-driven data presentation module for database system", US6308168B1, February 1999
2. Yan, Ling-Ling, Renée J. Miller, Laura M. Haas and Ronald Fagin, "Data-Driven Understanding and Refinement of Schema Mappings", SIGMOD 2001
3. Mo Chen, Rezaul Alam Chowdhury, Vijaya Ramachandran, David Lan Roche and Lingling Tong, "Priority Queues and Dijkstra's Algorithm", UTCS Technical Report TR-07-54, October 2007, [Online] Available: https://www.researchgate.net/publication/250152101
4. Priyanka Yadav, Vishal Sharma and Priti Yadav, "Cache Memory – Various Algorithm", International Journal of Computer Science and Mobile Computing, Vol. 3, Issue. 9, September 2014
5. Nathan Beckmann and Daniel Sanchez of Massachusetts Institute of Technology, "Modeling Cache Performance Beyond LRU", 2016, Online https://people.csail.mit.edu/sanchez/papers/2016.model.hpca.pdf
6. Jaafar Alghazo, Adil Akaaboune and Nazeih Botros of Illinois University at Carbondale, "SF-LRU Cache Replacement Algorithm", 2004, Online https://www.researchgate.net/publication/4088265_SF-LRU_cache_replacement_algorithm
7. R. L. Mattson, J. Gecsei and D. R. Slutz, "Evaluation techniques for storage hierarchies," IBM Sys. J., vol. 9, no. 2, 1970
8. M. Qureshi, A. Jaleel and Y. Patt, "Adaptive insertion policies for high performance caching," in ISCA-34, 2007
9. A. Pan and V. S. Pai, "Imbalanced cache partitioning for balanced data-parallel programs," in MICRO-46, 2013
10. G. Keramidas, P. Petoumenos and S. Kaxiras, "Cache replacement based on reuse-distance prediction," in ICCD, 2007
11. A. Jaleel, K. Theobald and S. Steely, "High performance cache replacement using re-reference interval prediction," in ISCA-37, 2010.
12. O'Neil E. J., O' Neil P. E., Weikum G., "An optimality proof of the LRU-k page replacement algorithm", ACM, vol. 46, no. 1, 1999

AUTHORS PROFILE

A. Murugan is an associate Professor & head of PG & Research department of Computer Science in Dr. Ambedkar Govt Arts College, University of Madras. His area of interest are Molecular Computation, Graph Theory, Data Structure, Analysis of Algorithms and Theoretical Computer Science. His publication covers 70 international & 15 national journal and conference and 5 books. Five of his Research Scholars completed PhD program and three of his students' Thesis were submitted. All of his students are awarded.

S. Ganesan is one of the research scholar of Prof A. Murugan, PG & Research department of Computer Science in Dr. Ambedkar Govt Arts College, University of Madras. He has 24+ years IT work experience across the globe. Since 2016, he published 3 research papers under the guidance of Dr. A. Murugan.