Speeding up Searching in B-cloud System

Shawgi Mahmoud
Huazhong University of Science and Technology
Wuhan National Laboratory for Optoelectronics
Wuhan, China

Hua Wang
Huazhong University of Science and Technology
Wuhan National Laboratory for Optoelectronics
Wuhan, China

Ke Zhou
Huazhong University of Science and Technology
Wuhan National Laboratory for Optoelectronics
Wuhan, China

ABSTRACT
With the enormous growth in data amount day by day storage demands have been imposed crucial requirements for data center, where data access plays essential role in estimation of data storage systems effectiveness. For that, there were a lot of efforts have been done to speed up searching in huge storage systems. Deduplication, one of the latest technologies in storage systems world, is founded on the principle of eliminating duplicates among data. Although this technology led to reduce vast amount of redundant data, still the matter of accessing speed of this huge amount of data is a critical problem. In this research a simple method was proposed to speed up searching in B-Cloud system by adopting Least Recently Used dual cache LRU which is able to save vast amount of time and speed up accessing time clearly. The Least Recently Used dual cache is based on the principle that elements that have been frequently demanded in the last few queries will probably be frequently demanded again in the next few times. The experiments results have shown that back up/recovery time decreased with the size of data and there is clear enhancement in both recovery and back up time (where back up time decreased 15.67% with 16KB dual cache size and 34.14% with 8MB dual cache size and recovery time decreased 15.69% with 16KB dual cache size and 32.3% with 8MB dual cache size), which led to enhance enormously the overall system performance.

General Terms
Storage system

Keywords
Cloud Backup system, Data Deduplication, Data Index , Least Recently Dual Cache

1. INTRODUCTION
In such an age of high information technology, the survival and development of enterprises depends on data more and more. People realized the great potential value of data and enterprises are increasingly focusing on data protection. The backup application was put on the business development agenda. Many companies have no hesitate to spend a large amount of money to buy data backup system [1, 2]. However, with the amount of backup data increasing year by year, it is more and more demanding for the stability of the system and the usability of the data, also the network backup system, such as the real-time backup, remote data access, Where the time play crucial role in effective management of data backup . A lot of techniques to speed up data access have been adopted specially in the hardware field such as large size bandwidth cables (DOCSIS)[3, 4],Flash Memory[5, 6] and etc. While at the same time, although with the explosive growth of data, studies have proved that a large number of duplicate data exists in all aspects of the information processing and storage, such as file systems, file synchronization, e-mail attachments, HTML documents, as well as the operating system and application software. Studies show that 22% of the HTML documents are the same, while the contents of 48% of the HTML documents are similar [7]. The traditional data protection techniques such as periodic backups will have a lot of duplicate data, resulting shortage in the resources of the network bandwidth and storage space also data management costs rising rapidly. To avoid excessive data growth, improve utilization of the resources and reduce costs, duplication in recent years slowly becomes more valuable research focus. In such a world the need for high-speed data access software based on data Deduplication [8-10], has become inevitable to backup software development. A high speed Data access information systems can provide an effective and practical tool to the backup software and equipment, provide better performance for the data and to help to make the operation of the data processing more effective, greatly reducing the time of accessing data and the cost of human efforts.

This paper proves with evidences how is the LRU dual cache speeds up searching in B-Cloud system (an online target de-duplication storage system which consist of backup client ,backup server and storage server). Simple but effective techniques had been adopted, which take advantage of already existed modules and the gain obtained from Deduplication technology. Experimental results conclude the whole performance under various conditions. The paper is organized as follows:

The first section is about overview of Different caches designs .The second section describes the design of LRU dual Cache which improves system backup performance and access time speed; LRU algorithm, reducing a huge amount of data accessing time ,improving the total performance of the system look up. Then describes the core design and implementation of LRU cache in storage server with the description details of the query mechanism during backup and recovery processes. The third section is the performance test of on-line backup service system without LRU dual Cache and compares it with the new version of system when LRU dual Cache is included in the backup system. The forth section concludes the paper and about the future work.

2. OVERVIEW OF DIFFERENT CACHES DESIGNS
Caches[11] in general are a simple method to enhance the performance of an application that reads data from a "slow" source such as files on disk or rows of data from a database list, but may need to re-read the same data several times. The

idea is plain: rather than disposal the data after using it, save it inside memory so that it will not be necessary to be re-read later. For instance, a straightforward method to arrange this for files might be to create a Hash Map that maps the file names to objects containing the file data. When the application needs a specific file it first checks the map to see if it already has the data; if it doesn't, it reads the file and inserts it in the map in order to make it available for the next time. The only obstacle of this simple method is that the application could use a large quantum of memory. The moment that read in, a file is in the cache for the life of the application whether it used or not later. For an application such as a server that is dedicated to stay up and running for weeks or months at a time, this is sure undesirable. So there is vital necessity for suitable replacement algorithm.

Cache replacement indicates to the procedure that occurs when the cache gets full and obsolete elements must be removed to make space for novel ones.

There are four classes of replacement strategies [11]:

1. Recency-based strategies.
2. Frequency-based strategies.
3. Recency/frequency-based strategies.
4. Function-based strategies.

Only the first three strategies will be discussed briefly in this introduction.

2.1 Recency-Based Strategies

These strategies were built on recency as a backbone. Most of them are expansions of the famous LRU strategy. LRU has been adopted successfully in various fields. LRU is built on reference locality observed in query flows. Reference locality characterizes the capability to expect future queries to elements from past accesses. There are two major kinds of locality: temporal and spatial. Temporal locality indicates to frequent accesses to the same element within squat time terms. It implies that recently referenced objects are likely to be referenced again in the future. Spatial locality indicates to access patterns where accesses to some objects imply accesses to particular other objects. It implies that references to some elements can be a predictor of future references to other elements. Recency-based strategies utilize the temporal locality observed in request flows.

There are a lot of cache replacement algorithms under this class such as:

2.1.1 Least Recently Used Algorithm (LRU)

It discards automatically objects that haven't been referenced for some time so as to bind the amount of space being used. Such as cache is called an LRU Cache. LRU stands for "Least Recently Used", which indicates to the policy of evicting the most obsolete or least-recently used objects to make space for novel ones.

LRU caches have limited number of elements that they will catch and these elements are usually organized in a table. When an element is added to the cache, and every time it is referenced after that, it is automatically moved to the head of the table. If the cache is full and a slot is required for a novel element, the cache creates room by discarding the element at the tale of the list - the least-recently used one. It is widely used in variant fields (e.g., database buffer management, paging, disk buffers).

2.1.2 LRU-Min [11]

This is a diverse of LRU that attempts to reduce the number of replaced documents. Suppose Lo and T stands for, respectively, a list and a threshold.

(1)Set T to S, where S is the requested document size.

(2)Set L0 to all documents whose size is equal to or larger than T.(Lo can be empty.)Apply LRU to L0 until the list is blank or the free cache volume is at least T. If the free cache volume is not at least S, set T to T2 and go to step (2).

2.1.3 The advantages of recency-based strategies

1-They assume temporal locality, as a vital factor. As request flow usually exhibits some sort of temporal locality, this is a useful process. In addition, these strategies are so flexible to workload changes (e.g., new very popular objects).

2-They are plain to applied and rapid. Most of these strategies adopt an LRU table. Novel requested objects are inserted at the head of the table. On a hit the object is removed from its current position and inserted at the head. Replacement occurs at the end of the table. So insertion and discarding add low complexity. As well as, searching can be supported by hashing techniques.

2.1.4 The disadvantages of recency-based strategies

1-Plain LRU diverse do not integrate recency and size in a advantageous, stable method. In different size objects environment size should be considered at every replacement. The size strategy does this; however, it is too aggressive as it places too much emphasis on the size of objects. LRU-Min is also concentrates more on size. The PSS strategy is a commendable exclusion, as it is plain to apply rapid, and integrates size and recency in a more stable method. Additionally, a properly parameterized partitioned LRU caching strategy can be very simple and fast.

2-They do not consider frequency information. This could be an important indicator in more static environments.

2.2 Frequency-Based Strategies

These strategies use frequency as a main factor. Frequency-based strategies are expansions of the common LFU strategy. They are built on the fact that different Web objects have different popularity values and that this popularity values result in different frequency values. Frequency-built strategies track these values and exploit them for future judgments.

There are various algorithms built on this strategy such as:

2.2.1 LFU [11]

It removes the least frequently requested object.

2.2.2 LFU-Aging [11]

With LFU, objects that were very popular during one time period can remain in the cache even when they are not requested for a long time period. This is because of their high frequency count. To obviate this cache pollution, an aging effect can be provided. That is why LFU-Aging uses, a sill. If

all frequency counters average value exceeds this sill, all frequency counters are divided by 2. As well as, this strategy exploit a maximal value for the frequency counters.

2.2.3 The advantage of the frequency-based strategies

They regard the frequency of access. This is useful in static environments where the popularity of objects does not vary too much over a particular time interval (day, week).

2.2.4 The disadvantages of the frequency-based strategies

1. Complication, LFU-based strategies need a more complicated cache management. LFU can be adopted, for instance with a priority queue.

2. Cache pollution, Frequency counts are too static for dynamic variations in the workload. That is why, aging was provided. But aging is only a recency-based technique. It is doubtful if complicated aging techniques are better than plain recency-based techniques in dynamically various environments. As well as, they add complication to the replacement process.

3. Similar values, Many objects can have the same frequency count. In this case, a tie breaker factor is needed.

2.3 Recency/Frequency-Based Strategies

These strategies exploit recency and frequency and maybe additional factors to get an object for replacement.

There are a lot of algorithms based on this strategy such as:

2.3.1 SLRU (Segmented LRU) [11]

The SLRU strategy divides the cache into two partitions: an unprotected partition and a protected partition (dedicated for popular objects). On the first demand for an element, this element is inserted into the unprotected partition

On a cache hit, the element is moved to the protected partition. Both partitions are managed with the LRU strategy, but only elements from the unprotected partition are discarded. Elements from the protected partition are moved back in the unprotected partition as the most recently used element. This strategy needs a parameter which decides what percentage of the cache volume is allocated to the protected partition.

2.3.2 LRU[11]*

All requested elements are stored in one LRU cache. Each element has a request counter. When a cached element is hit, it is moved to the head of the table and its hit count is incremented by one. At each replacement process, the hit counter of the least recently used element is checked. If it is zero, the element is discarded. Otherwise the hit counter is decreased by one, and the element is moved to the head of the cache.

2.3.3 The advantage of Recency/Frequency-Based Strategies:

They integrate recency and frequency. If they designed perfectly, such strategies can overcome the troubles of recency- and frequency-based strategies described above.

2.3.4 The disadvantage of Recency/Frequency-Based Strategies:

Due to special procedures, most of these strategies introduce additional complexity. Only LRU* attempts to integrate the simple implementation of LRU with frequency counters. However, they do not regard size.

In addition of its previous advantages which is mentioned above and due to its practical, simplicity and because of its suitability for B-Cloud system (the practical implementation of LBDC [12] system) environment where is the size of objects are equal and its dynamic which lead to overcome all its disadvantages (LRU) strategy was selected to be implemented in dual cache design where are two integrated caches were adopted in B-Cloud storage system to speed up searching and accessing data.

3. DESIGN AND IMPLEMENTATION OF LRU DUAL CACHE

This section will explain in details design and the key technology of LRU dual cache in B-cloud system.

3.1 The main key technologies

As it was mentioned in the second section that LRU strategy was chosen to be adopted in B-cloud system due its suitability ,simplicity and flexibility .but which algorithm from this strategy we are going to select?!

3.1.1 LRU (Least Recent Used Algorithm)

It is extension of traditional LRU[12](Least Recent Used Algorithm) but the only difference here is the most recently used object is located in the tail of the cache and the least recent ones will be located in the head of it. Which is opposite of the famous LRU algorithm so the removing of the least recent elements will be faster instead of iterating along all objects in the cache first and then discard the last one in the tail of the cache .Two equal size caches were adopted in order to increase the hit rate[13] of cache.

3.1.2 The Mechanism of LRU Algorithm

As in figure 3.1 LRU dual cache mechanism can be described as follow:

1-Once the query for specific object in the dual cache starts searching along all elements in the first cache will begin looking up for that particular item.

2-Whenever that object get referenced (hit accrued) it is deleted from its position in the cache and it is re-added again in the cache tail in order to insert it in the most recent section in the cache pushing up the previous one step forward.

3-If that queried element doesn’t exist in the first cache (miss accrued) searching will start in the second one.

4-Once that missed element is gotten in the second cache it will be added to the tail of the first cache in order to be available for future demand.

5-If that queried element doesn't exist in the second cache (miss accrued) searching will start in the main data index table.

6-If that queried element is gotten in the main data index table it will be added to the tail of the first cache otherwise false will be returned.

When the first cache gets full the item in its head will be discarded but it will be re-added to the tail of the second one. With that mechanism dual cache keep the most used object in the tail of the first cache while the most least used items in the head of the second one ready to be discarded when that cache get full.

3.1.3 The Components of LRU Cache

In storage server which is the B-cloud system part where all hashes is stored CDataSpace" and "CIndexSpace deal with reading and writing storage space data and maintains metadata of main records so we built a dual cache between (CStorageSpace) and (CDataSpace&CIndexSpace) classes as it is shown in fig 2 which is represented with a(CCache) class in order to act as dual cache so CStorageSpace class will search in it for the DataIndexID first before looking up for it in the CIndexSpace class.

The dual cache is two double linked list [14] each time new object is added the previous one will be moved to the head of the cache so we will end with the least recent used object in the head of the both two caches ready to be discarded when the dual cache get full which will lead to increase system performance.

4. SYSTEM PERFORMANCE EXPERIMENTS AND ANALYSIS

This section will analyze and explain how the LRU dual cache technology adopted by the B-Cloud system impacts on system performance. The backup-recovery speed is tested in different situations of various sets of backup data; compare it with the speed of these two operations without using dual cache in order to estimate the enhancement of system performance. We also studied the effect of LRU dual cache on the system speed all that with Variety of dual cache size. We used four groups of different size of data sets to do the experiments.

4.1 Experiments Environment

One server and one PC，backup server and storage server are installed on the server; client end is installed on PC, which is connected by D-Link DES-1024 D 100 M switches, the specific configuration:

Storage server：windows server 2003，Intel(R) Xeon(R) CPU X3220 2.4GHz，memory 4GB，disk 14TB.

Backup client：Windows XP，Intel(R) Pentium(R) Dual CPU E2180 2.00GHz，memory 0.99GB，disk 160GB.

4.2 Backup/Recovery Performance Experiments without LRU dual cache

Table 1 shows backup and recovery results with different size of data sets to the system without LRU dual cache ,The unit of backup time is second, denoted hour: minute: second. The total time they all used is recorded and the transmission speed is weighted mean value.

It is noticeable from the previous table above that file size plays the main rule in determining the backing up /recovery speed which is in turn impact in the required time to complete the operation. So it is clear that the fastest backed up file is the smallest one in size as it is shown in table 4.1.

4.3 Backup/Recovery Performance Experiments with 16 KB LRU dual cache

The effect of 16 KB LRU dual cache shown in table 2. Here it is so obvious evident if we calculate the average value of all back up time in table 2 that the LRU dual cache decreased greatly both backup and recovery time with 15.67% & 15.69% respectively speeding dramatically up the overall system speed which lead to improve the system performance clearly.

4.4 Backup/Recovery Performance Experiments with 8MB LRU dual cache

It is time to determine the effect of caches sizes in the system performance as it is shown in table 3. In table 3 the impact of 8MB LRU dual cache will be checked on the different size of data sets in the system performance. If table 3 was compared with tables (1 & 2),it would shown that there is also obvious improvement when the dual caches sizes go larger which in turn will lead to great enhancement in system performance where it can be seen clearly in the backing up time decreasing with 34.14% from the case without using any LRU and with 21.9% from the case where the LRU dual cache sizes were 16 KB because of the huge size of dual cache which will lead to increase in look up speed and this will be clear when the size of file get large at that moment the improvement in backing up time will be so obvious the same thing it can be said about recovery time where it is decreased with 32.3% from the case where is there is no LRU dual cache at all and with 19.69% when the size of LRU dual cache was 16 KB. From all of above it can be concluded that there is great enhancement in both backup and recovery time when LRU dual cache is used whether the size of dual cache were 16 KB or 8 MB but the improvement will be greater when dual cache sizes get larger. Fig 3 illustrates the impact of different sizes of dual cache in backup time. As it is can be easily noticed in that fig that there is obvious decrement in backup time when LRU dual cache are used and this decrement will increase when the sizes of dual cache get larger because most of hashes will be available inside dual cache which will lead to speed up look up dramatically and end up with great enhancement in back up time.

Fig 4 shows the great gain in recovery time when dual cache are used and the impact of the size variety in the system performance. There is obvious improvement in recovery time because most of blocks will be available in dual cache which are smaller than main hash table so the look up speed will be so fast as it is shown in that fig where is this enhancement grow more with expansion of dual cache sizes because of great increment in hit rate at that case which will lead to enormous improvement in recovery time.

So from the previous two figs (3&4) and tables (2 & 3) it can be concluded that there is wide improvement in both backup and recovery time and the existence of objects inside dual cache (old data) in both backup and recovery operations

specially when the size of that object is small and the sizes of dual cache are big play the crucial rule in system speed where it can be guaranteed fast and perfect system performance.

5. CONCLUSION

In today's Information Age, people are increasingly dependent on data, data loss; damage often could be fatal to organizations. In some conditions, the impact of data loss or other problems is immeasurable. Thus, data protection is particularly important. Network data backup as a key technology to protect data has become one of the core IT infrastructure. This paper describes the impact of LRU technology to improve the look up speed in an on-line data backup service system which can implement block-level data de-duplication, the performance of the system was tested. Test results proved that the LRU technology storage server used improves the performance of both backing up and recovery speed; achieve the purpose of experimental study. This major work done can be summarized as followings:

(1) First, a preview of different caches design with their advantages and disadvantages was introduced.

(2)After that in details the key technologies to implement and design the LRU dual cache including the algorithm, the main components and its mechanism were explained.

(3) Finally, the impact of the LRU dual cache in the backup and recovery speed of the storage server system was tested and analyzed, and based on these tests the appropriate conclusions were drawn.

Except the technology and implementation, other aspects of the online data backup service system are required to be improved as well. For instance backup time always more than recovery time(because of hash values checking process) although LRU dual cache decrease it a lot and this decreasing will increase dramatically with the growth of dual cache size but more efforts are still need to be done in order to decrease the difference between them as much as possible by suitable techniques.

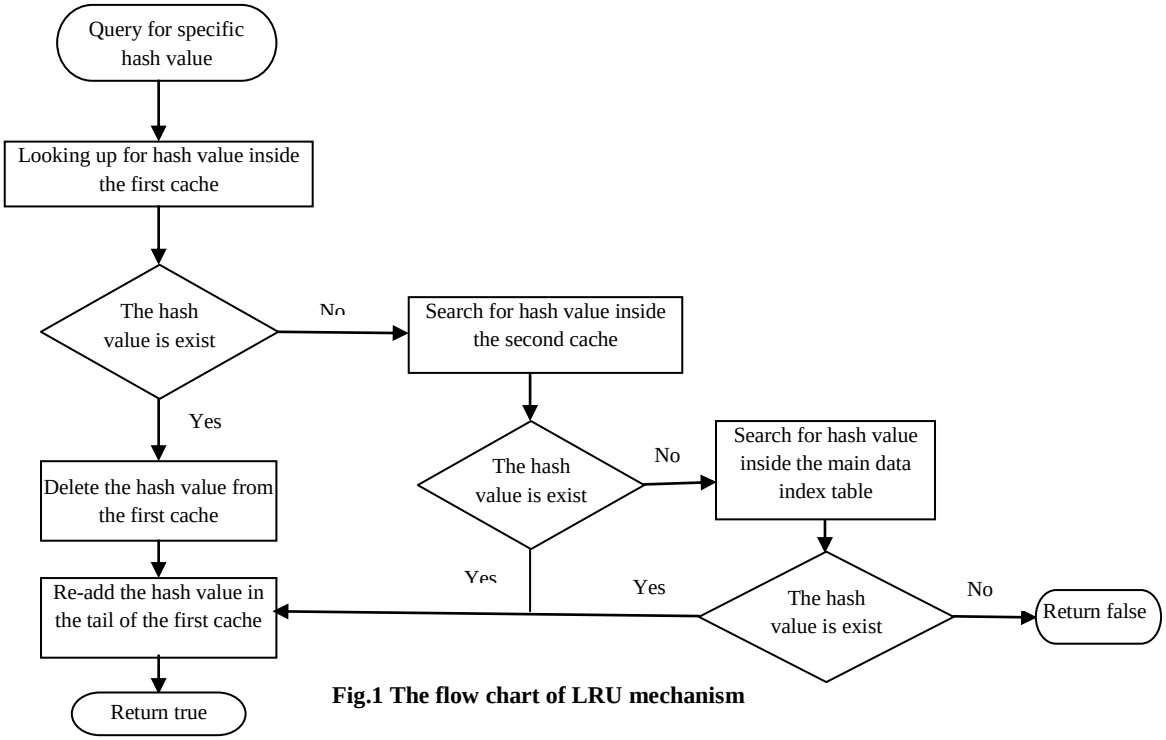

Fig.1 The flow chart of LRU mechanism

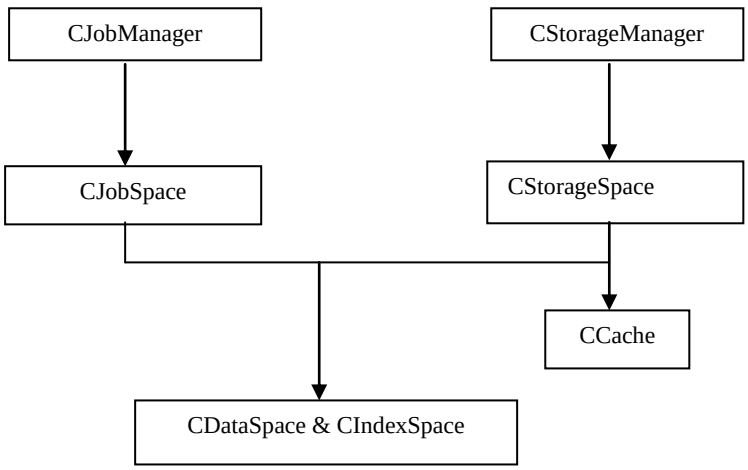

Fig.2 Modified hierarchal relationships among classes in storage server

Table 1. the results of system performance with different size of data sets without LRU dual cache

Size of data sets MB	Job type	Job time (hour:minute:second)	Job speed MB/S
585	Back up	00:01:11	8.24
585	Recovery	00:00:35	16.73
1024	Back up	00:02:14	8.41
1024	Recovery	00:01:12	15.65
2048	Back up	00:04:13	16.02
2048	Recovery	00:02:17	18.21
5120	Back up	00:06:05	14.09
5120	Recovery	00:05:04	17.02

Table 2. the results of system performance with different size of data sets and 16 KB LRU dual cache

Size of data sets MB	Job type	Job time (hour:minute:second)	Job speed MB/S
585	Back up	00:00:54	10.84
585	Recovery	00:00:33	17.74
1024	Back up	00:01:27	9.66
1024	Recovery	00:00:58	18.47
2048	Back up	00:03:38	17.40
2048	Recovery	00:01:41	19.20
5120	Back up	00:05:35	15.88
5120	Recovery	00:04:30	18.67

Table 3. the results of system performance with different size of data sets and 8 MB LRU dual cache

Size of data sets MB	Job type	Job time (hour:minute:second)	Job speed MB/S
585	Back up	00:00:40	13.15
585	Recovery	00:00:30	18.96
1024	Back up	00:01:05	9.80
1024	Recovery	00:00:43	19.75
2048	Back up	00:02:28	18.38
2048	Recovery	00:01:19	20.33
5120	Back up	00:04:49	18.35
5120	Recovery	00:03:39	19.38

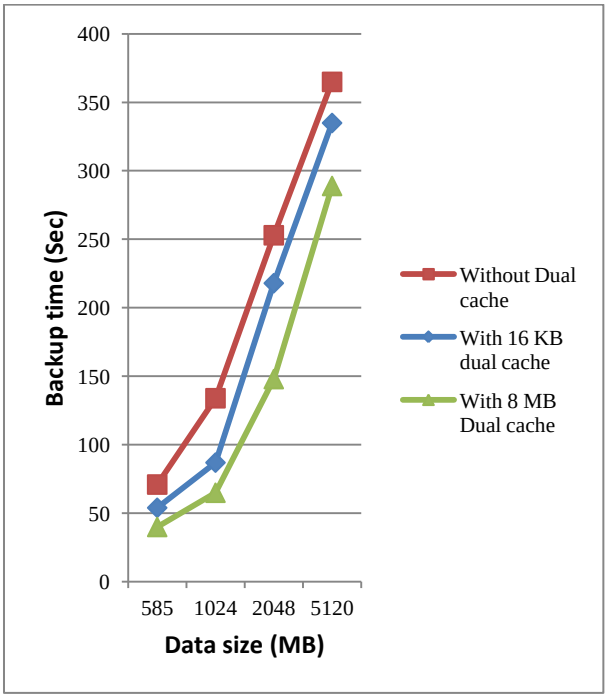

Fig.3 backup time performance for various data sets sizes without cache and with different dual cache sizes (16KB&8MB).

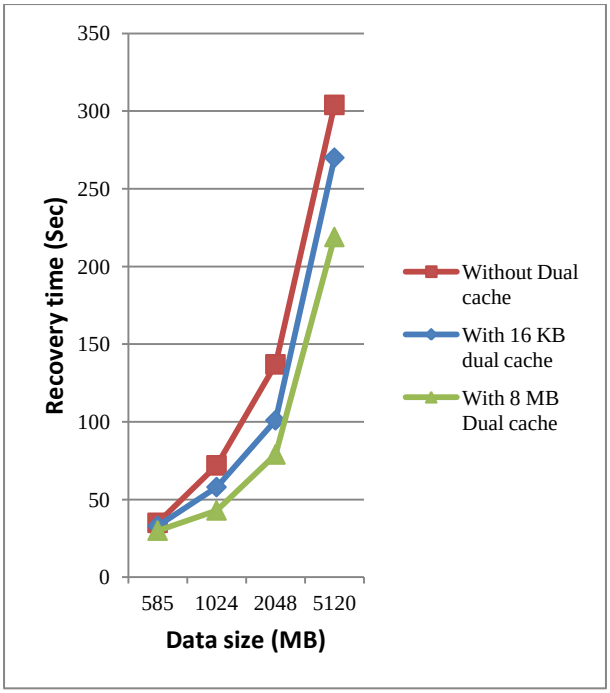

Fig.4 recovery time performance for various data sets sizes without dual cache and with different dual cache sizes (16KB&8MB)

6. ACKNOWLEDGMENTS

I dedicate this paper to my father, mother, family, university, supervisor, colleagues and everyone who helped me in hoping that the completion permeated by interest.

7. REFERENCES

[1] G. Peng, *et al.*, "Backup data storage system technology research and analysis," in *Computer security*, 2003, pp. 71-72

[2] X. He, *et al.*, "NDMP-based block-level backup/recovery method and its implementation," in *Jisuanji Gongcheng/ Computer Engineering*, 2007, pp. 82-84

[3] R. F. I. Specification, "Data-Over-Cable Service Interface Specifications," in *SP—RFIv2. 0 —*, ed, 1999, pp. 1-58

[4] W. K. Kuo, *et al.*, "Improved priority access, bandwidth allocation and traffic scheduling for DOCSIS cable networks," in *Broadcasting, IEEE Transactions on*, 2003, pp. 371-382

[5] B. Debnath, *et al.*, "ChunkStash: speeding up inline storage deduplication using flash memory," 2010, pp. 16-16

[6] R. Bez, *et al.*, "Introduction to flash memory," in *Proceedings of the IEEE*, 2003, pp. 489-502

[7] N. Shivakumar and H. Garcia-Molina, "Finding near-replicas of documents on the web," in *The World Wide Web and Databases*, 1999, pp. 204-212

[8] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," 2004, pp. 6-6

[9] M. W. Storer, *et al.*, "Secure data deduplication," 2008, pp. 1-10

[10] G. H. Forman, *et al.*, "Data de-duplication," 2007, pp. 1-3

[11] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," in *ACM Computing Surveys (CSUR)*, 2003, pp. 374-398

[12] J.Czebotar.(2011, Jan 14). *LRU Cache in C with uthash*. Available:http://jehiah.cz/a/uthash

[13] Grund D and Reineke J, "Estimating the performance of cache replacement policies", in MEMOCODE, IEEE Computer Society,2008, pp. 101–112

[14] Kendell A. Chilton," Methods and apparatus for accessing a doubly linked list in a data storage ",Patent No US 6,941,308 BI,6 sep ,2005 ,pp,1-19