International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 15 (2017) pp. 5040-5046
© Research India Publications. http://www.ripublication.com

An Efficient Inline Data Deduplication with Data Relationship Manager for Cloud Storage

Venish A[1,*] and Sivasankar K [2]

[1]Computer and Engineering, Noorul Islam University, Kumaracoil, Tamilnadu, India.

Orcid id: 0000-0003-0742-8248

[2]Information Technology, Noorul Islam University, Kumaracoil, Tamilnadu, India.

Orcid id: 0000-0001-6963-7905

Abstract

In the current digital and cloud world, the data handling is a biggest challenge for all the service providers. One way or other way we need more storage backup system to handle the disaster recovery. For the high performance, the video and audio streaming application, database application, multimedia applications need more memory and high speed process. Using data deduplication concept in the storage system, we can reduce memory space requirement and can improve system performance.

This paper is focusing on Metadata management and Replacement Algorithm to provide the high throughput and minimal resource utilization. By implementing LIRS replacement algorithm in the deduplication system for caching, it reduces the IO access compare to other replacement algorithm. Also this system improves the LIRS Meta data lookup speed using Data Relationship Manager. We have explained the complete Architecture, Write and Read process, Algorithm implementation and discussed the different experiment results.

In our study the LIRS with Data Relationship Manager improves the time taken for the deduplication and reduces the IO access. We compared 4000 files with 5 different data pattern, the result shows the LIRS with Data Relationship Manager works well on the weak locality data pattern. The result clearly shows that without relationship manager, LIRS is taking much more time for the deduplication. The Time and Workload pattern comparison shows the result is improved and it reduces the memory access when we have Data relationship manager. LIRS with Data relationship manager improves the deduplication efficiency and throughput.

Keywords: Deduplication; Cloud Storage; LIRS; Data Relationship Manager; Inline Deduplication;

INTRODUCTION

All the replacement algorithm is based on the Recency or Frequency. Recency is focusing the last reference time whereas the frequency is focusing a block reference count. LIRS is a recency based algorithm. When file read request comes, before it goes to secondary memory, first the request goes to the cache. If file is there, then it is called as 'HIT', if file is not there then it is called as 'MISS'. The good replacement algorithm is decided based on the hit ratio. There are different components involved in the data deduplication system. First method is Chunking-the incoming data splits in to smaller chunks using fixed or variable chunking methods. Hashing is Assigning unique identification value or fingerprint to each of these chunk using hash algorithm. Once hashing is over the duplication detection method checks the existing stored fingerprint index for deduplication detection. The final steps is Index Updating & Storing. If the index is exists then the chunk is replaced with a reference pointer or the chunk is written to the disk as a new unique data chunk.

A. Chunking Method:

Chunking method splits the data in to smaller chunk using different chunking algorithm. There are two level of chunking, one is file level and another one is block level. In the file level, the hash value is created for the complete file whereas the block level, the file is divided into fixed size chunk or variable size chunk. Fixed size chunking algorithm divides the data into fixed size such as 4KB, 8KB, and 16KB and so on. Variable size chunking algorithm can be in the form of content aware chunking, delta encoding and sliding window which divides the data into variable size based upon the set of rules. Chunking is very important key factor in the deduplication system. Wrong selection of chunking may affect the duplication ratio and system performance.

B. Hashing Method:

Once chunking is over, the fingerprint or hash value is created by using different hashing algorithm, such as MD5, SHA and Robin Fingerprint for each chunk. In some sporadic cases, the two different chucks may have same hash value. It is called false positive ratio. So the system decides it is duplicate chunk and adding only the reference point for the chunk, instead of storing the second chunk. It causes data loss. To avoid this issue, some of the study reveals that combination of the hash algorithm and grouping the hash value can decrease the false positive ratio [1].

C. Duplicate Detection:

In order to remove the duplicate data, hash index lookup or comparison has to happen with existing stored hash value. The metadata value of hash value is stored into the main memory and secondary memory. First the hash value is compared with main memory, if data not found the same value is compared with secondary memory metadata. In the small scale system the stored fingerprint value is less, so the comparison of index value is not much complex and less time consume. But when we implement the deduplication system in the large scale storage system, we can expect the amount of fingerprint value is higher. Apparently in this case the main memory cannot hold all the index value, so we need external disk access to read or write the fingerprint. When we access the disk for fingerprint read or write, the performance of the deduplication system also will go down. But we cannot avoid to store the fingerprint in the disk for large scale storage system. There are various studies carried over to improve the finger print lookup and reduce the IO access by selecting correct replacement algorithm or improving search criteria.

D. Index Updating & Storing:

If the fingerprint exists in the index table, the data chunk/block is replaced with a pointer to data chunk/block. If the fingerprint does not exist, the data is written to the disk as a new unique data chunk and the entry is made in the index table.

MOTIVATION

In the cloud storage environment, there is a lot of necessity to handle the large amount data with minimal storage space and less resource access. The data backup and data recovery also should be taken care. So the deduplication system needs to provide high throughput and good deduplication ratio. To achieve this key factor the hashing and metadata management is very important. Also unwanted main memory and disk access hurts the deduplication performance very badly. So in this system we concentrated to use the correct replacement algorithm to decide the element movement between the main memory and Meta data disk. The combination of the good replacement algorithm and structured Meta data manager can reduce the metadata search time and IO access. We implemented the LIRS [4] algorithm with improved history node data relationship manager. This approach provides considerable results with different IRR value and DDR size.

In summary, this paper contributes inline deduplication, overall structure of the system and LIRS Replacement algorithm with Data relationship manager.

MATERIALS AND METHODS

The main objective of this system is to focus on deduplication concept in the Inline Process and provide the effective solution to handle the large amount of data within expected time.

This system is storing Metadata in the format of B-Tree and using replacement algorithm (LIRS) to overcome the main memory metadata overhead and disk seek problem. Also provides good efficiency and disk throughput. It handles Metadata effectively. Fig. 1 shows the system structure and its components.

System Architectural overview.

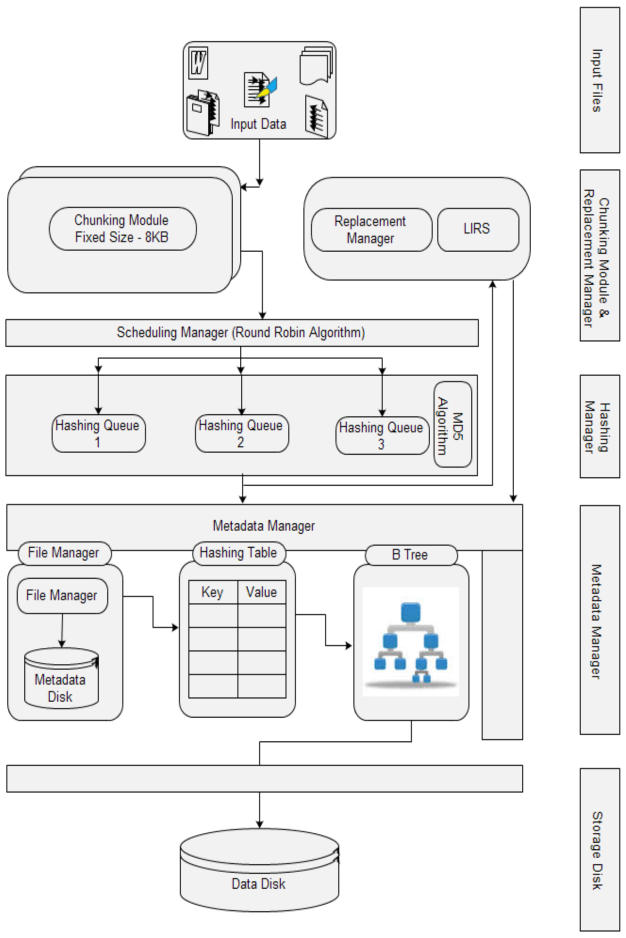

Figure 1: The system overview.

A. Chunking Module:

Chunking module is responsible for the data chunk. This system uses fixed size chunking with 8 KB. Each incoming file is divided into 8 KB fixed size length.

B. Scheduling Manager and Hashing Manager:

To speed up the hashing process, this system uses scheduling and hashing manager for the parallel process. Once the data passed through the chunking module, the chunked data are moved into the scheduling manager. This scheduling manager is using Round Robin algorithm which distribute the chunk data to hashing manager. There are three hashing queue in the hashing manager. Each hashing queue is responsible for creating hash value for each chunk by using MD5 hashing algorithm.

C. Design of Metadata Manager:

Metadata Manager: Metadata Manager is designed with following important components.

File Manager: File manager is responsible to store File related information and Metadata information. This file manager is directly connected with the Metadata table. Most frequently used metadata value is kept into the main memory, and least used Metadata is swapped into Metadata disk by using replacement algorithm to reduce the disk seek operation. Each Metadata value is stored in hash table in the form of B-Tree. The new incoming hash value is compared with B-Tree through hash table. If the hash index is found in the B-Tree, only the reference count is added, the data is not stored. If the hash index is not found in the B-Tree, the hash index is added into B-Tree in the Hash Table and the data is added into storage disk.

Replacement Manager: This manager uses Heuristic based replacement algorithm (LIRS) to reduce the main memory Metadata overhead. This manager keep on checking the allocated buffer memory size. Once it exceeds the threshold value, this manager will flush the main memory metadata to metadata disk and make some free space for new incoming data in the main memory also accordingly changes the B-Tree value. Fig 2. Shows the Replacement Manager overview and the process.

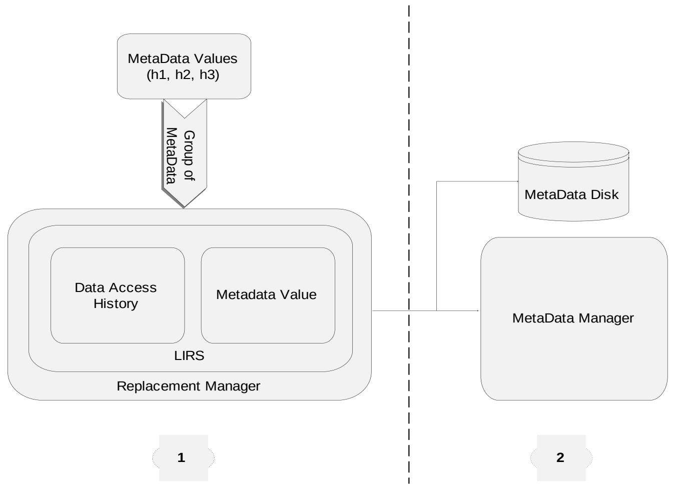

Figure 2: Replacement Manager Overview

The replacement manger is responsible for below functionalities,

1. A Group of Metadata will be given as input to Replacement Manager. The replacement manager uses the Low Inter-Reference recency Set (LIRS) algorithm to replace the data. This LIRS maintains the Data access history and metadata value.
2. File manager will keep track of new and existing metadata value along with file information.
3. We are maintain a hash table which will have all the hash index grouping information, and all the index value will be kept in the form of B-Tree. By storing index value in the B-Tree, the comparison of existing index with new index is very fast.

D. Algorithm Overview:

In our previous study [9] we explained the detail design of LIRS and its Stack. LIRS has High Inter-reference Recency (HIR) and Low Inter-reference Recency (LIR) set to keep track of the Residence LIR and HIR chunks based on the Inter-Reference Recency (IRR) value. This IRR value is calculated based on the chunk access. The HIR element will be replaced when new chunks comes in, the LIR element will not be replaced until it moves to the HIR stack. LIRS has very important History tracking functionality which is not in the LRU replacement algorithm. This History Tree represents the Non-Residence elements and arrange the elements in the B-Tree format. All removed elements form the HIR and LIR set will be traced in the History Tree and the corresponding data of the element will be moved to the Metadata disk.

E. System Design and Implementation:

In our previous work [9] we implemented the LIRS and LRU algorithm separately in deduplication system with different workload pattern. The analysis of this implementation proves that LIRS performance is higher than the LRU when we use weak locality pattern and time taken for the deduplication is lesser than the LRU. For the strong locality pattern LRU and LIRS makes no difference.

The LIRS algorithm uses the history tree to track the recently used indexes. In the write request, the incoming element index is not found in the LIR and HIR stack of LIRS, the search goes to LIRS history tree. If the element found in the history, then the corresponding element will be given priority and will be updated in the LIR and HIR stack of the LIRS. If the element is not found in the History tree then the search request goes to Metadata disk to perform the search operation for the incoming element. So the Metadata disk read is required if the element is not found in the History tree. But in the cloud storage or big data process system, the search operation in the Metadata disk affects the Deduplication performance heavily.

To avoid the complete search operation for the each History Miss index in the Metadata disk, we have implemented Data Relationship rules which keeps track of Metadata disk index location in the Metadata disk. If the incoming element search is Miss in the history tree, then the index search is happening on Data Relationship Tree, which is having the location where the search operation has to perform to find out the index. In this way we can avoid complete linear search in the disk and we can navigate the particular position for the index search. Fig 3 shows the write request with Data Relationship Manager.

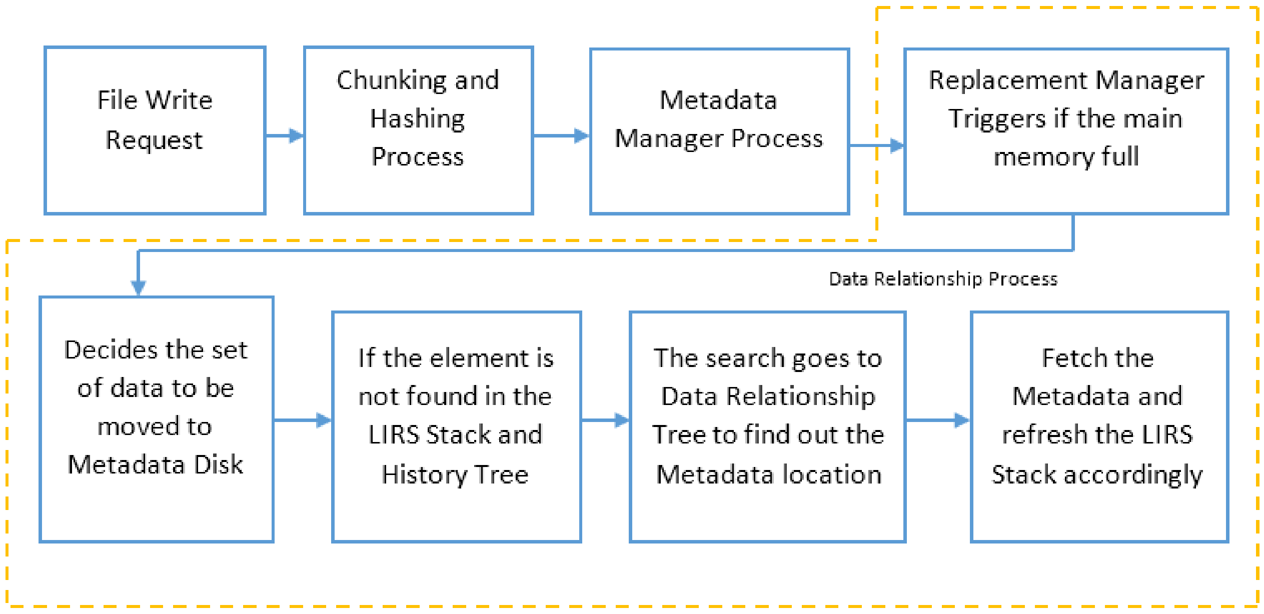
File Write Request
Chunking and Hashing Process
Metadata Manager Process
Replacement Manager Triggers if the main memory full
Data Relationship Process
Decides the set of data to be moved to Metadata Disk
If the element is not found in the LIRS Stack and History Tree
The search goes to Data Relationship Tree to find out the Metadata location
Fetch the Metadata and refresh the LIRS Stack accordingly

Figure 3: File write request flow with relationship manager

The existing linear search approach does complete search on the Metadata disk which causes the disk seek performance. We have implemented the Data Relationship tree to improve the Metadata search performance. The Data Relationship B-Tree stores the set of block location information based on the index key. The incoming search index value is compared with Data Relationship Tree before start the Metadata disk search. The DR Tree search tells the possible search location based on the incoming index and DR Tree block address. So the search will happen on the particular Metadata disk location and can skip the complete disk search.

F. File – WRITE & READ

When the file write request comes, this system divides the file into fixed size chunk and Creates the hash value for each chunk using MD5 Algorithm. It saves the Metadata value into File Manager. To compare the new metadata value with existing metadata value, it picks up the hash index from the Hash Table and search it in the B-Tree. If this new index is found in the B-Tree then add the reference value for that index, if not found add to the B-Tree, the B-Tree will be adjusted accordingly and it writes the file in to the file disk. Fig. 3 shows the file write request with Data Relationship Manager detailed process and Fig. 4 shows the element search in the History and DRT.

When the file read requests comes, the search happens on the metadata table to find out the file index and location. The corresponding file related data will be picked and read out the file from the disk.

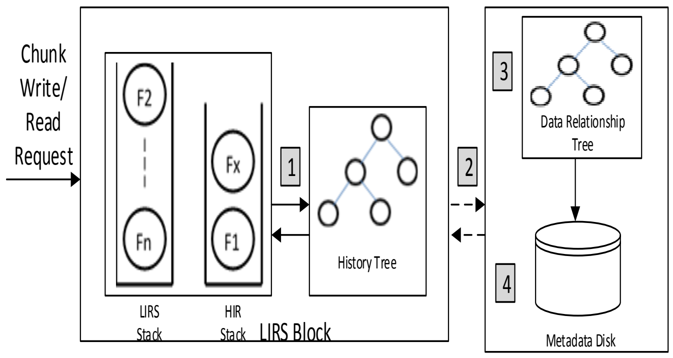

Figure 4: File search in History and DRT

RELATED WORK

There are many studies carried to improve the deduplication system performance.

In our previous study, we have analysed the complete deduplication system [8] and the different chunking methods [10] as the chunking method is key and important steps in the deduplication system. The fixed and variable chunking are used or the combination of both also used to make the correct chunking process. The next steps is to find the unique index for the each chunk. There are various hashing methods to identify the unique values [10]. We also analysed the different replacement algorithm which are used in the deduplication system. The LRU replacement algorithm is mainly used in all the deduplication system [6, 7] to move the element based on their accessibility. Hands [12] the segment based group uses LRU, LFU (native) and LFU (Working set aware) replacement algorithm. LRU is very easy to implement also provides good results in the strong locality pattern, but in the weak locality pattern it fails to provide the accurate deduplication results and it affects the deduplication ratio. LIRS provides good result for the weak and strong data pattern.

There are various studies to grouping the data before store or before index comparison. By grouping the index or elements the search would be bit faster and can reduce unwanted IO access [11]. Routing chunk data in the correct cluster node also yields the good deduplication ration. Cluster, Distributed [5] and Cloud based [3] deduplication methods are improving performance using cache mechanism. This approaches uses LRU queue to decide the recent used elements [13]. Some of the systems are using combination of Main Memory and Flash-Memory. In this way the bottleneck of RAM usage can be reduced, also the Flash-Memory is faster than the Hard Disk and cheaper than the RAM [1]. The metadata indexes are stored in the Flash drive to improve the index comparison performance [2]

RESULT ANALYSIS

This model system, totally 4000 files used and each file size is 256 KB. By using multithread application we generated the five different data pattern and analysed with the Work load, Time taken for the Deduplication and the different DDR size. This system consuming the 8K fixed size chunking model and LIRS module is configured to allow 10000 LIR elements, 5000 HIR elements and 15000 History elements. This system we ran in windows system with 1 TB 6 GBPS Seagate SATA drive for data disk, 200 GB 6 GBPS Seagate SATA drive for the metadata. We changed the size of the DDR size from 6 to 16.

A. Workload Experiment

We have selected the same data pattern which we used for our previous study [9]. Based on the data frequency access, we have categorized the five different data pattern for this experiment.

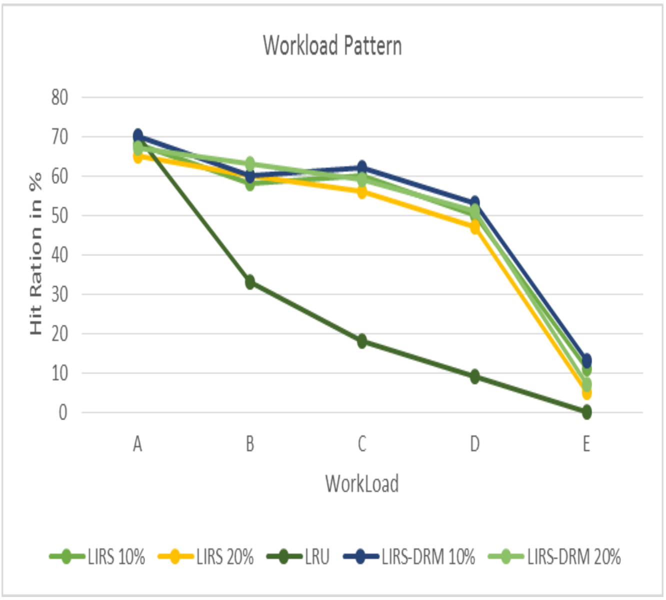

Figure 5: Hit Ratio comparison for different workload and different IRR value.

Pattern A – Large number of Hot spots, Pattern B – Multiple Hot spots, Pattern C - Limited number of Hot spots with random access pattern, Pattern D – Limited number of Hot spots with sequential access pattern, Pattern E – Pure random access pattern. To perform the WEAK LOCALITY process, the Pattern C and Pattern D are selected. Hit ratio for these different data pattern are various for the both LRU and LIRS. The changing of IRR value in the LIRS also tested and the results are captured. The same IRR value is tuned up to 20% and noted the significant improvements.

The observation of the results Fig. 5 shows the LIRS provides considerable improvements than LRU with Data Relationship manager. The Data Relationship manager improves the search efficiency and saving the time compare to the same LIRS.

B. Time Take for Deduplication

The standalone replacement module is used to perform this experiment. We tested with and without Data Relationship Manager for the LIRS replacement algorithm and compared with LRU algorithm. The result shows that the LIRS with Data Relationship Manager improves the time which taken for deduplication process. The notable point for this experiment Fig. 6 shows that the disk access is reduced for the Metadata comparison in which the time taken for the process also reduced. For the weak locality access pattern, the LIRS with Data Relationship Manager can be a considerable option as this complexity of deduplication time is very less. But the LRU I taken bit more than the LIRU.

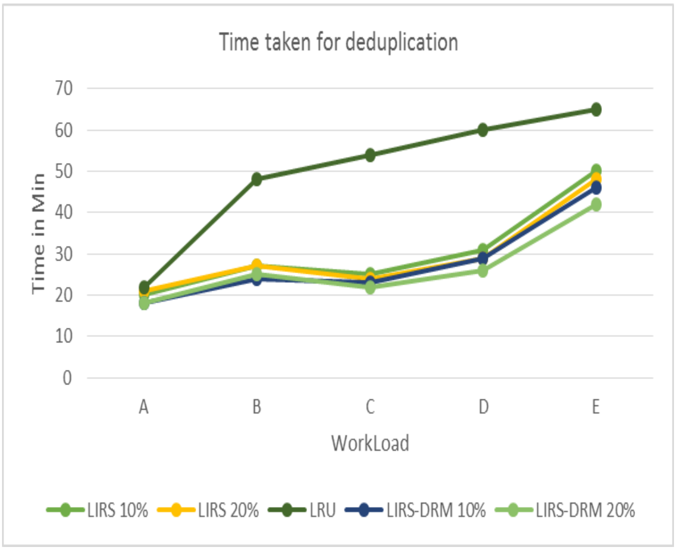

Figure 6: Time taken for deduplication.

C. DDR Size Analysis

In this experiment, Fig. 7 we have analysed with different DDR (RAM) size and tuning different IIR values between 10% and 20%. When we increase the DDR size the Meta Data disk access is lesser than the previous experiment. Because the Main memory can have more Meta data which reduces the Meta data disk access. The result for the LIRS with Data Relationship Manager is improved when compared to the LIRS process.

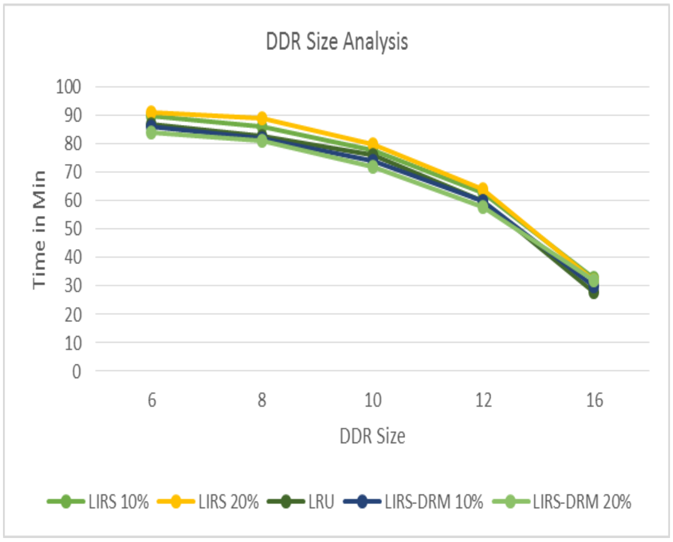

Figure 7: DDR Analysis.

CONCLUSION

The good replacement algorithm decides the Deduplication system performance, Ratio and Throughputs. The use of LIRS for the weak locality access patterns provides the considerable results and improve the Time and reduces the IO access. Further improving the results we have identified the potential changes on the LIRS algorithm by maintain the Data Relationship Manager. This manager is responsible to keep track of the Metadata disk saved indexes. The results shows this implementation improves the results further for the weak locality data pattern. The LRU algorithm is not suitable for the data pattern like File Scanning, Looping Data Pattern and the Different Frequencies Data Pattern. The LIRS algorithm provides better results for these access pattern and can handle large numbers of Meat Data effectively. LIRS with Data Relationship Manger experiment results also improved than the LIRS. Implementation of Date Relationship Manger proves the efficiency and the time taken for the deduplication is improved and can handle the more Metadata in the inline data deduplication system.

Further improvement of the Data Relationship Manager also possible. Using Flash or High speed drive for the Data Relationship Manager can provide the improve results. Also there is a scope of implementing this Manager in the cluster system or different node system can improve the search performance.

REFERENCES

[1] Biplob Debnath, Sudipta Sengupta, and Jin Li. 2010. ChunkStash: speeding up inline storage deduplication using flash memory. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). USENIX Association, Berkeley, CA, USA, 16-16.

[2] Biplob Debnath, Sudipta Sengupta, and Jin Li. 2010. FlashStore: high throughput persistent key-value store. Proc. VLDB Endow. 3, 1-2 (September 2010), 1414-1425. DOI=http://dx.doi.org/10.14778/1920841.1921015.

[3] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, and Franck Youssef. 2015. Transparent Data Deduplication in the Cloud. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 886-900. DOI:https://doi.org/10.1145/2810103.2813630

[4] Jiang S, Zhang X (2002). "LIRS: An Efficient Low Inter reference Recency Set Replacement Policy to Improve Buffer Cache Performance", In Proceeding of (2002) ACM SIGMETRICS, June (2002), pp. 31-42.

[5] João Paulo and José Pereira. 2016. Efficient Deduplication in a Distributed Primary Storage Infrastructure. Trans. Storage 12, 4, Article 20 (May 2016), 35 pages. DOI: http://dx.doi.org/10.1145/2876509

[6] Min J, Yoon D, Won Y (2011). "Efficient deduplication techniques for modern backup operation," IEEE Trans. Comput., 60(6): 824-840.

[7] Srinivasan K, Bisson T, Goodson G, Voruganti K (2012). "idedup: Latency-aware, inline data deduplication for primary storage," in Proceedings of the 10th USENIX conference on File and Storage Technologies. USENIX Association, 2012.

[8] Venish A, SivaSankar K (2015). "Framework of Data Deduplication: A Survey", Indian J. Sci. Tech., 8(26).

[9] Venish A, SivaSankar K (2016). "HIDE - Heuristics Based Inline Data Deduplication for Cloud Storage", Transylvanian Review, Vol. XXIV, No. 07, 2016.

[10] Venish A, SivaSankar K (2016). "Study of Chunking Algorithm in Data Deduplication", Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015, Springer India, 2016, pp. 13-20.

[11] Wang L, Zhang X, Zhu G, Zhu Y, Dong X (2013). "An Undirected Graph Traversal Based Grouping Prediction Method for Data De-duplication," 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Comput. (SNPD), pp. 3-8.

[12] Wildani A, Miller EL, Rodeh O (2013). "HANDS: A heuristically arranged non-backup in-line deduplication system," 2013 IEEE 29th Int. Conf. Data Eng. (ICDE), pp. 446-457.

[13] Yinjin Fu, Hong Jiang, and Nong Xiao. 2012. A scalable inline cluster deduplication framework for big data protection. In Proceedings of the 13th International Middleware Conference (Middleware '12), Priya Narasimhan and Peter Triantafillou (Eds.). Springer-Verlag New York, Inc., New York, NY, USA, 354-373.