International Journal of Research in Advent Technology (E-ISSN: 2321-9637) Special Issue
National Conference "NCPCI-2016", 19 March 2016
Available online at www.ijrat.org

An Optimization of Backup Storage using Backup History and Cache Knowledge in reducing Data Fragmentation for In_line deduplication in Distributed

Abhijit Goswami[1], Nitin Shivale[2]
Department of Computer Engineering[1,2,3]
Bhivrabai Sawant Institute of Technology & Research Pune, India
Email: goabhijit239@gmail.com[1]
nitinrajni3@gmail.com[2]

Abstract: The chunks of data that are generated after the backup are physically distributed after deduplication in backup system, which creates a problem know as fragmentation. Basically fragmentation basically comes into sparse and out-of-order containers. The sparse container adversely affect the performance while restoring the database and garbage collection effectively , while the out-of-order container brings an adverse effect on the performance issue if the restore cache built is small. To overcome this fragmentation problem , we propose a method of History-Aware Rewriting algorithm (HAR) and also Cache-Aware Filter (CAF).HAR will gather the historical information in backup systems to define, identify and reduce sparse containers, and CAF acknowledges restore cache knowledge to find the out-of-order containers that impacts restore performance. CAF supports HAR in datasets where out-of-order containers are prominent. To get rid of metadata of the garbage collection, we exploit Container-Marker Algorithm (CMA) to gather valid containers instead of valid chunks. My output helps to prove how HAR significantly improves the restore performance.

Keywords:chunkHARfragmentation,metadataCMA

1. Introduction

The present challenge in backup storage infrastructure is the management we need to handle the ever increasing volume of data. To face known challenge and to convert management scalable, de-duplication technique is very well known and new techniques and research are being done to make this technique more optimized for future use. Data deduplication is a special technique used in data compression. These work by eliminating the duplicate copy in storage. Hence it helps in improving the total experience and utilization of the data storage in dataset. Also help in managing the data transfer via network where the amount of data transfer will get reduced significantly. In deduplication we keep only one copy of data and eliminated redundant data and refer other data which are redundant to the same copy original copy. Deduplication can happen either in the file level system or can happen in the block level. In file-level it removes duplicate copies of the files which are same. The fragmentation are classified into two categories: sparse containers and out-of-order The former reduces restore performance, which might be self-addressed by increasing the size of cache used for restoration. The latter reduces each restore performance and garbage collection, and that we need a editing rule that's capable of accurately distinguishing sparse containers. So as to accurately establish and reduce sparse containers, we have a tendency to observe that sparse containers stay sparse in next backup, and hence propose HAR..HAR considerably improves restore performance with a small decrease of deduplication ratio. We have a tendency to develop CAF to take advantage of cache information to spot the out-of-order containers that might hurt restore performance. CAF is employed within the hybrid theme to boost restore performance underneath restricted restore cache while not a major decrease of deduplication magnitude relation. So as to scale back the data overhead of the garbage collection we have a tendency to propose CMA that identifies valid containers rather than valid chunks within the garbage collection. Deduplication also can also happen at the block level, thus eliminate duplicate set of blocks of knowledge that occur in non-identical files.Although the data deduplication methods brings lots of advantages to user along with security and privacy issues. Still the users' sensitive data can be in danger from insider and outsider attacks. Ancient encoding, while providing the knowledgeconfidentialityto the user, is incompatible with data deduplication. Specifically, ancient encoding needs completely different set of users to cypher their own data of which they have their own keys. Thus, identical dataset copies of completely of various users can result in different cipher texts, creating deduplication not possible. Convergent encoding has been planned to enforce dataset confidentiality whereas creating deduplication possibleAnalyzing the visual content may not be sufficient to capture users' privacy preferences. Tags and other metadata are indicative of the social context of the image, including where it was taken and why [4], and also provide a synthetic description of images,complementing the information obtained from visual content analysis.

2. Literature Survey

A. *iDedup: Latency-aware, Inline Data set Deduplication for Primary Storage in systems.*

Deduplication technologies are more and more being deployed to scale back value and increase space-efficiency in company knowledge centers. However, previous analysis has not applied deduplication techniques to the path which is requested for latency sensitive, primary workloads. This is often primarily thanks to the additional latency these techniques were introduced. Inherently, deduplicating data that is present on the disk causes fragmentation [2] that may increases seek for sequent successive reads of a similar data thus, increasing latency. Additionally, deduplicating knowledge needs additional disk IOs to access on-disk deduplication information. During this paper, we have a tendency to propose AN inline deduplication resolution, iDedup [1], for primary workloads, whereas minimizing additional IOs and seek. Our algorithmic rule relies on 2 key insights from real-world workloads: i) abstraction vicinity exists in duplicated primary knowledge; and ii) temporal vicinity exists within the access patterns of duplicated data. Mistreatment the primary insight, we have a tendency to by selection deduplicated solely sequences of disk blocks. This reduces fragmentation and seeks caused by deduplication. The second insight permits U.S. to switch the high-priced, on-disk, deduplication information with a smaller, in-memory cache. These techniques alter U.S. to exchange capability savings for performance, as incontestable in our analysis with real-world workloads.

B. *Chunk Fragmentation Level: An Effective way to Indicate for Read Performance Degradation in*

Data deduplication has recently become very well known thing in most auxiliary storage and even in some primary storage for the capability improvement purpose. Other than its write performance, browse performance of the deduplication storage has been gaining quite importance with a good vary of its deployments. During this paper, we have a tendency to emphasize the importance of browsing in theperformance in reconstituting a knowledge stream from its distinctive and shared chunks physically spread over deduplication storage. We have a tendency to freshly introduce a browse performance indicator referred to as Chunk Fragmentation Level (CFL) [3]. We have a tendency to conjointly validate that the CFL is incredibly effective to point browse performanceof the deduplication storage by understanding theoretical performance model and intensive experiments.

C.*Optimized Hybrid Inline and Out-of-Line Deduplication for Backup Storage in database*

Backup storage systems typically take away redundancy across backups via inline deduplication that works by referring duplicate chunks of the newest backup to those of existing backups. Inline deduplication reduces restore performance of the newest backup owing to fragmentation, and complicates deletion of terminated backups owing to the sharing of knowledge chunks. Whereas out-of-line deduplication addresses the issues by forward-pointing existing duplicate chunks to those of the newest backup, it introduces extra I/Os of writing and removing duplicate chunks. We have a tendency to style and implement RevDedup[4], associate degree economical hybrid inline present in data set and out-of-line deduplication system for backup storage. It applies for the coarse-grained inline deduplication data to delete the duplicates of the newest backup, and so fine-grained out-of-line reverse deduplication to delete the duplicates from older backups. Our reverse deduplication style limits the I/O overhead and prepares for economical deletion of terminated backups

D.*Reducing the impact of data fragmentation caused due to in-line deduplication*

Deduplication results inevitably in knowledge fragmentation, as a result of logically continuous knowledge is scattered across several disk locations. during this work we have a tendency to specialize in fragmentation caused by duplicates from previous backups of identical backup set, since such duplicates area unit quite common attributable to recurrent full backups containing plenty of unchanged knowledge. For systems with in-line dedup that detects duplicates throughout writing and avoiding them to store, such fragmentation causes knowledge from the most recent backup being scattered across old backup. Then the time of restore from the most recent backup may be considerably inflated, typically over doubled. we have a tendency to propose AN formula referred to as context-based editing (CBR in short)[5] minimizing this come by restore performance for up to date backups by shifting fragmentation to older backups, that area unit seldom used for restore. By selection editing a tiny low proportion of duplicates throughout backup, we will scale back the come by restore information measure from 12--55% to solely 4--7%, as shown by experiments driven by a group of backup traces. All of this can be achieved with solely little increase in writing time, between a hundred and twenty fifth and five-hitter[8]. Since we have a tendency to rewrite solely few copies of duplicates and recent copies are basically rewritten knowledge area unit removed within the background, the entire method introduces little and temporary house overhead.

E.File recipe compression technique in data deduplication systems

Data deduplication database finds and exploit redundancy of data between totally different information blocks. The foremost common approach divides information into chunks and identifies redundancies via fingerprints. The file content is remodeled by combining the chunk fingerprints that square measure hold on consecutive during a file formula. The corresponding file formula information will occupy a major fraction of the full disc space, particularly if the deduplication magnitude relation is incredibly high. We have a tendency to propose a mix of economical and scalable compression schemes to shrink the file[6][7]. At simulation shows that these ways will compress file recipes by up to ninety three.

3 **PROPOSED METHODOLOGY:**

A Convergent encryption:

Convergent secret writing provides information confidentiality in deduplication. A user tries to derive a convergent key from every original information copy and encrypts the data copy with the help of convergent key. Additionally, the user conjointly tag for the data copy, specified the tag are accustomed find duplicates. Here, we try to assume that the correctness of the tag holds property, i.e., if 2 information copies area unit constant, then their tags area unit constant.

To find repeating copies, the user initial sends the tag to the server aspect to envision if the identical copy has been already keep. Note that each the convergent key and therefore the tag area unit severally derived and therefore the tag cannot be accustomed deduce the convergent key and compromise information confidentiality. The information which are encrypted and its respective tags are keep on the server aspect

B. HAR Architecture:

We have pool of container which can provide a storage service on disk. We can use fingerprint indexes for this. The disk is useful for keeping the finger print index, and hot part which is in memory. For writing the chunk we use container buffer which is present inside memory the dataset are assigned with the unique ID, say DS1.The transaction or historical data we have are stored on disk which consists of ID for e.g. DS1the transactional or historical data are divided into 3 main parts: Sparse container of HAR for inherited IDs, the optimal replacement cache and CMA for container manifest

C. History-Aware Rewriting Algorithm

- Whenever the backup starts, HAR holds the IDs of sparse container which are inherited to build in-memory S inherited structure. Whenever backup start, HAR rewrites all duplicate chunks whose container IDs exist in S inherited.
- Along with that HAR also maintains a structure in built knows as S emerging to monitor or housekeeping the container referred by backup and maintains their utilization factor.
- S emerging is used to record the utilization factor and each dataset or record consist of utilization factor of each container
- Once the backup is done HAR uses the technique to get the record of higher utilization from S emerging. S emerging has the list of all emerging container which are sparse.
- S emerging can be directly sent to disk as the size of S emerging is small due to our second observation
- Hence from the observation we can come to know that if the value of chunks is more than they are rewritten in next backup. It would hamper the performance and cause bottleneck issues.
- To mitigate this effect HAR set the limit as 5% for rewriting, tis would avoid too much rewrite on future backup. HAR makes use of limit defined for rewriting for segregating too many sparse container in S emerging
- HAR helps in estimating the rewrite ratio for the coming backup. Specifically it calculates the size for the chunks which are rewritten for each emerging sparse container.
- This is done using the help of the utilization factor that is calculated from container size. The rewrite ratio is later calculated as the total of all estimated size dividing by the current backup size, which may be give us approx. value of rewrite ratio for the coming backup or next backup.
- If the value which is estimated for rewrite ratio crosses the predefined value rewrite limit, HAR helps in removing the Semerging with highest utilization and directly jump to step1Else HAR helps in replacing the IDs which are old inherit sparse container with their IDs of the emerging sparse container present in S emerging. The result generated as S inherited which helps to provide as the S inherited for next backup.

The complete work flow of HAR is delineated in rule one. Figure four illustrates the time period of a rewritten distributed Container. The parallelogram may be a Container, and therefore the blank space is that the chunks not documented by the backup. We have a tendency to assume four latest backups are maintained. (1) The Container becomes sparse in a backup n. (2) The Container is rewritten in backup n + one. The chunks documented by backup n + one ar rewritten to a replacement Container that holds distinctive chunks and different rewritten chunks (blue area). But the previous Container can't be saved once backup n + one, as a result of backup n2, n1, and n still seek advice from the previous Container. (3) Once backup n + four is finished, all backups bearing on the previous Container are deleted, and so the previous container are often saved. Every distributed Container decreases the restore performance drastically in the given database of the backup recognizing it, and can be saved once the backup is deleted. Attributable to the restricted variety of familial distributed containers, the memory consumed by the S familial is negligible. S rising consumes a lot of memory as a result of it has to monitor all containers that are connected with the help of the backup. If the de-fault Container size is four MB and therefore the average utilization is five hundredth which might be simply achieved by HAR, the S rising of a one TB stream consume eight MB memory (each record contains a 4-byte ID, a 4-byte current utilization, associated an 8-byte pointer). The memory footprint is smaller than the editing buffer utilized in cosmic microwave background radiation and Capping. There's an exchange in HAR as we have a tendency to vary the employment threshold. The next utilization threshold leads to a lot of containers being thought of distributed, and so backups are of higher average utilization and restore performance however worse deduplication magnitude relation. If the employment threshold is ready to 50%, HAR guarantees a mean utilization of no but 50%, and therefore the most restore performance isn't any but 50% of the most storage information measure.

D. Optimal Restore Cache

To reduce the side effects of out-of-order containers on restore performance, we tend to implement Belady's best replacement cache. Implementing the optimal cache (OPT) has to apprehend the long run access pattern. We will collect such data throughout the backup, since the sequence of reading chunks throughout the restore is simply constant because the sequence of writing them throughout a backup. Once a chunk is processed through either elimination or over-writing its Container ID, its Container ID is understood. We tend to add access record in the information that is collected .Every access record will solely hold a Container ID. Consecutive accesses to the identical container will be incorporated into a record. This part of historical data will be updated to disks sporadically, and so wouldn't consume a lot of memory The entire sequence of access records will consume hefty memory once out-of-order containers are dominant. Forward every container is accessed fifty times intermittently and also the average utilization is five hundredth, the entire sequence of access records of a one TB stream consumes over a hundred MB of memory. Rather than checking the entire sequence of access records, we can take a help of slide window to look into a fixed-sized a part of the long run sequence, as a near-optimal theme. The memory foot-print of this near-optimal theme is thus delimited.

E. Container-Marker Algorithm:

Existing garbage pickup schemes have faith in merging thin containers to reclaim invalid chunks within the containers. Before merging, they need to spot invalid chunks to work out utilizations of containers, i.e., reference management. Existing reference management approaches area unit inevitably cumbersome owing to the existence of huge amounts of chunks. HAR naturally accelerates expirations of thin containers and therefore the merging is not any longer necessary. Hence, we'd like to not calculate the precise utilization of every container we tend to copy the Container-Marker algorithmic program (CMA) to with efficiency confirm that containers area unit invalid. CMA assumes users delete backups during a FIFO theme, during which oldest backups area unit deleted initial.

4. System Architecture

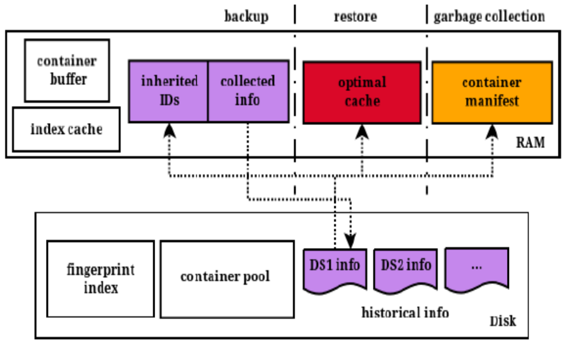

Fig. 1 System Architecture

The Main purpose of our model is to provide Security to Authorized Deduplication in daily backup and helping it to reduce the fragmentation via exploiting backup history and cache knowledge. Deduplication comes with many issues like security, privacy of user. The security measures taken in deduplication are not good enough. Specially the old measures to protect the data against encryption. The earlier method encrypt with the own key making it impossible to access.A new method knov

Convergent encryption used widely for making deduplication possible. Hence hash value is used to protect the data copy. After the generation of key the user keeps the key and sends the cipher text to backup system. As the key is derived from the main data the duplicate will have the same data which would prove out to be easier to access the data in feasible way.A new protocol is introduced for user data to identify the duplicate data is the copy of main file hence need not upload the same in the server.A convergent key can be used to decrypt the data which is encrypted in the server. This key can be downloaded from the server itself.This encryption can help to prevent the unauthorized access of data in the server for the user

- *User Access*

User creates account in the database with his username, password, name, email, mobile no and so on.This data are used as user access management.The user needs to qualify for the data access

- *File Uploading and its encryption*

User need to upload files in the backend.
Encryption of file is done using Convergent Encryption.
Convergent encryption algorithm is used to follow data confidentiality while making Deduplication feasible.
It encrypts/decrypts a data set with a convergent key, which is produced by applying the cryptographic hash value of the content of the data copy.
After the generation of the key and database encryption, users will have the keys and send the cipher text to the backup storage. Since the encryption operation is produced from the data content, identical data copies will produce the same convergent key and thus we will have the same cipher text.

- *Fragmentation reducing in the system*

To overcome the Fragmentation, we have introduced History-Aware Rewriting algorithm (HAR) and Cache-Aware Filter (CAF). HAR exploits historical information in backup history systems to mark correctly and reduce sparse containers, and CAF exploits restore cache knowledge to understand the out-of-order containers that will impact restore performance

5. ALGORITHM

History Aware Algorithm

Input: S inherited IDs of inherited sparse container
Output: Semerging IDs of emerging sparse container

```
1. First you need to initialize Semerging
2. Using while unless the backup is done
3. Need to receive the chunk of the data produced
   and lookup for its fingerprinting in the
   fingerprint index
4     if the chunk is duplicate
5     ifthe container ID exists in S inherited
Then
6         Rewrite the chunk to a new container
7     else
8         Need to eliminate the chunk data
9     endif
10    end
11    write the chunk to new container
12    endif
13    update the utilization record in Semerging
14    endwhile
15  Remove all utilization record of larger utilization
16   calculate the estimated rewrite ratio for the next
     backup
17  while the estimated rewrite ratio is larger than the
     rewrite limit do
18  remove utilization record largest utilization in
    Semerging
19  Update the estimated rewrite ratio
20  endwhile
21  return Semerging
```

6. CONCLUSION

There is considerably decrease in restore and garbage collection efficiencies because of fragmentation in deduplication backup systems. Fragmentation can be categorized in 2 forms as sparse container and out-of-order container. Sparse basically tell us about the maximum restore and out-of-order tells us about restore performance. HAR help us to identify and rewrite sparse with the knowledge of history. We also came to the conclusion that an optimal caching scheme which is optimal and hybrid algorithm act as a complementary to HAR for reducing the impact of out-of-order case. HAR and OPT helps to optimize the restore performance in deduplication ratio. HAR helps to optimize both deduplication ratio and restore performance. As to reduce deduplication in hybrid scheme we involved CAF to reduce deduplication ratio in hybrid scheme. We can adapt CAF for optimizing the rewriting algorithms. Container-Marker Algorithm (CMA) is introduced to identify valid containers instead of valid chunks. CMA is bounded by the number of containers; it is more cost-effective than the number of chunks.

REFERENCES

[1] .K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, “iDedup:Latency-aware, inline data deduplication for primary storage,” in Proc. USENIX FAST, 2012.
[2]Y. Nam, G. Lu, N. Park, W. Xiao, and D. H. Du, “Chunk
fragmentation level: An effective indicator for performance

degradation in deduplication storage," in Proc. IEEE HPCC, 2011.
[3]F. Guo and P. Efstathopoulos, "Building a highperformance deduplication
system," in Proc. USENIX ATC, 2011.
[4] J. Wei, H. Jiang, K. Zhou, and D. Feng, "MAD2: A scalable highthroughput
exact deduplication approach for network backup
services," in Proc. IEEE MSST, 2010
[5] M. Kaczmarczyk, M. Barczynski, W. Kilian, and C. Dubnicki
[6] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise,
and P. Camble, "Sparse indexing: large scale, inline deduplication
using sampling and locality," in Proc. USENIX FAST, 2009.
[7] D. Meister and A. Brinkmann, "dedupv1: Improving deduplication
throughput using solid state drives (SSD)," in Proc. IEEE MSST, 2010.
"Reducing impact of data fragmentation caused by in-line deduplication,"in Proc. ACM SYSTOR, 2012.