2016 International Conference on Computational Intelligence and Networks

The efficient use of Storage Resources in SAN for Storage Tiering and Caching

Abhijith U
Hewlett Packard Enterprise
Bengaluru, India
abhijith.u@hpe.com

Mohanta Taranisen
Hewlett Packard Enterprise
Bengaluru, India
taranisen.mohanta@hpe.com

Ashish Kumar
Hewlett Packard Enterprise
Bengaluru, India
ashish.kumar@hpe.com

Leena Muddi
Hewlett Packard Enterprise
Bengaluru, India
leena.muddi@hpe.com

Abstract— **with the rapid growth of the Hyper Convergence (HC) the use of the Solid State Device (SSDs) as the intelligent Storage Tiering and caching has increased immensely. It plays an important role to provide high throughput and performance with maximum utilization of the SSD storage space considering the life cycle of SSDs and workload. Efficient Utilization of such resources is one of the main criterion of the virtualization technology. By considering the rise of the VDI workload the proposed implementation of a sequential write detection algorithm, we try to maximize the utilization of higher tier storage such as SSDs (Tier 0) by placing the data which fall under the category of sequential writes into the lower tier disks (Tier 1, 2…n) such as SAS (serial attached SCSI) / SATA (serial advanced technology attachment).**

Keywords- SAN, Tier, RAID, random write,VDI, sequential write, storage virtualization.

I. INTRODUCTION

The term storage virtualization refers to separation of the storage into physical implementation level of storage devices and logical representation level of storage for use by operating systems, applications and users [6]. The storage virtualization is very helpful because we can combine disk capacity from several arrays into a single virtual volume. One can do replication from one array to another for disaster recovery or offline backup by the use of RAID [6, 7] technologies. We can also have the benefit of data migration from older array to the newly purchased equipment without interrupting data access for the user's application. One other benefit is the monitoring of the entire system by use of central management console [6].

A Storage Area Network (SAN) is a high availability, high-performance dedicated storage network that provides access to consolidated, block level data storage. SAN enhances storage devices like tape libraries and disk arrays. The main advantage of SAN is efficient use of Storage Resources by data migration. There are mechanisms to administer the profiles of the data and to determine which data is required and how often. In this manner it is possible to control the distribution of the data on fast and slow storage devices in order to achieve a high data throughput for frequently required data. The fast and slow storage devices are therein called the tiers [2].

Storage Tiering is an emerging technology on the storage platforms with the rise of flash storage. It is the art of voluntary movement of data between the storage tiers based on the access patterns. The data that is accessed frequently (hot data) can therefore occupy the faster tiers and the infrequently (cold data) accessed data is placed in the slower tiers. This act of placing frequently accessed data on high performance storage, and infrequently accessed data on low cost storage is called Tiering [3]. This is very similar to swap space and main memory in the operating system [4, 5]. We call Tier 0 faster tier and Tier 1 slower tier.

Least recently used is the algorithm to decide which data blocks to drop from the tiers. The Least Recently used replacement policy selects that data for replacement which has not been referenced for the longest time. For a long time, LRU is considered to be the one of the optimum algorithms. The problem with this approach is the difficulty in implementation. One approach would be to tag each block with the time of its last reference. LRU policy does nearly as well as an optimal policy, but it is difficult to implement and imposes significant overhead [1, 2]. As a variant of LRU, Least Frequently Used [LFU] can be implemented wherein there is a count value associated with the data blocks [1]. LFU involves keeping track of the number of times a block is referenced in memory. Each time a reference is made to that block the counter is increased by one. When the faster tier becomes full the block with the lowest frequency is evicted. This method too has a disadvantage. Consider that some block was referenced repeatedly for a short period of time and is not accessed again for an extended period of time. Due to how rapidly it was just accessed its counter has increased drastically even though it will not be used again for a decent amount of time. This leaves other blocks which may actually be used more frequently susceptible to purging simply because their frequency was low [1].

Other algorithms can be used to evict the pages from the fastest tier. The simplest page-replacement algorithm is a first-in, first-out (FIFO) page replacement algorithm. One disadvantage of this algorithm is it experiences Belady's anomaly [5]. The second chance algorithm is yet other page replacement algorithm. It is also called CLOCK where each page is tagged with a reference bit which is set to 1 when the page is accessed [1, 5]. This algorithm can be implemented as a circular queue. There is a pointer which indicates the pages to be replaced next. When a page is needed pointer advances until it finds a page with reference bit 0. As the pointer is moved reference bits which are set to 1 is cleared. Once a page with reference bit 0 is found, a new page will be inserted in that position with the reference bit set to 0. When all the bits are set

2375-5822/16 $31.00 © 2016 IEEE
DOI 10.1109/CINE.2016.28

the page replacement algorithm degenerates to FIFO .The Not Recently Used (NRU) replacement algorithm is an algorithm that favors keeping pages in memory that have been recently used. The pages are divided into four categories: 3. Referenced, modified; 2. Referenced, not modified; 1. not referenced, modified; 0. not referenced, not modified. Categories 0 happens when a categories 3 page has its referenced bit cleared by the clock interrupt. The NRU algorithm selects a page randomly from the lowest category for removal. So out of the above four pages, the NRU algorithm will replace the not referenced, not modified. The Not Frequently Used (NFU) page replacement algorithm requires a counter, and each and every page has one counter associated with it which is initially starts with 0 Thus, the page with the lowest counter values can be swapped out when required. The main drawback with NFU is that it keeps track of the frequency of use of pages without considering the time span of use of the pages. Thus results in poor performance. Belady's Min is one other theoretical approaches which says discard the pages which will not be used for the longest time in the future [1, 9]. It is impossible to predict whether these pages will be needed in the future or not, hence not suitable for any real time application. Adaptive Replacement Cache (ARC) is an adaptive page replacement algorithm extends LRU which keeps track of both frequently used and recently used pages. This algorithm solves some of the problems of fast tier miss [1, 9].

There is a difference between Caching and Tiering. Tiering is act of placing frequently accessed data on high performance storage, and infrequently accessed data on low cost storage whereas Caching is a duplicate copy of data that is stored in high speed media (Solid State SSD or PCIe card). Content of cache changes dynamically from minute to minute. Caching algorithms are usually used to decide which data blocks to drop from cache in order to store a newly accessed data block in high speed media [2]. Tiering can involve more than two types of storage areas whereas cache is typically two tiered. Tiering can be manual or automatic. The choice between the two is often limited by what alternatives are available. If performance is the goal, then the extra cost of having data plus a copy in cache is not very relevant. Caching provides the most responsive form of optimization [8]. The algorithms such as LRU, LFU, FIFO, CLOCK, NRU mentioned above can be used to evict the pages from cache.

There would have been severe performance degradation of I/O's if there was no Caching or Tiering solutions. There would not have been distinction between the hot data and cold data and all data would have occupied the lower tier storage. On the other hand it is impossible for all to afford all SSD storage as the cost of the SSD's are considerably high. In this era of high speed network, the enterprise storage systems expect storage devices to provide very high IOPS at minimal cost and faster access (low latency). So Tiering and caching would prove to be one of the best cost effective and relatively high performance solution.

II. PROBLEMS SOLVED

Workloads change over time and most of the times are unpredictable. Hence tiering proves to be one of the best solutions for such unpredictable workloads. By adding a tier of solid state drives (SSDs) to accelerate workloads, we can tackle the performance challenges more cost-effectively than adding a separate storage pool based entirely on the Flash.

Since the higher tier has limited storage space and there should be intelligent and efficient way to utilize the storage space. So the proposed method is to determine the workload which may not use the storage space immediately and place the data on the lower tier. The important and hot data will reside on the higher tier to serve efficiently and effectively.

III. PROPOSED SOLUTION

Lack of intelligent mechanism to detect the sequential I/O's can be considered a disadvantage because Sequential I/O's are usually big and tend to occupy major part of the faster tier and are simply involved in the movement of data between the faster and the slowest tier. Once an I/O can be considered sequential, we can allocate the space for such I/O' s from the slowest tier bypassing the faster tier and saving the space on the faster tiers for the random I/O's. The proposed novel work is here to detect the sequential write workload and allocate the storage space on the slow/lower tier instead of the higher tier. The sequential workloads are mostly the video files, big size files and so on.

IV. ALGORITHM

The host I/O when comes to the storage system the proposed algorithm identifies the sequential write. Once we detect the I/O as the sequential write the allocation will happen on the slow/lower tiers instead of the higher tier.

We can keep track of certain number of I/O's to decide whether the given I/O is sequential. I/O's are in the form of range. Range is composed of length and offset.

We maintain 2 queues - singleton queue and main queue to maintain all I/O's. Singleton queue is used to keep track of random I/O' s and the count value associated with each entry in this queue is one. Main queue is used to detect sequential I/O's and count values associated with each entry in this queue is >=2. The algorithm is depicted as shown in the Figure 1.

When an I/O comes in Sequential detector first loops in through the main queue. For each iteration, we choose an item from the main queue and check whether incoming I/O range is greater than item's range. If so we check whether Ranges are overlapping or touching. If true, we perform the union of the ranges. This is shown in Figure 2 and Figure 3.

If the incoming I/O is slightly out of order and out-of-order detection is enabled, we will look forward by some window-size. If the I/O is not contiguous with current item, but falls into the peek window, we consider it as sequential (Incoming -> offset <= item -> offset + item -> length + window_size). Out-of-order detection is enabled only when item's count is greater than or equal three. In this case too we perform the union of the ranges. This is shown in Figure 4.If this is true then, we increment the count and change the range accordingly and move this entry to tail of the main queue.

Else sequential detector loops through the singleton. For each iteration we choose an item in the queue and verify the whether (Incoming I/O range > item's range && areas are overlapping or touching).If yes then we increment the count and

move the match stream to the main queue and I/O range is merged.

If the incoming I/O is not matched with the stream in the main or singleton queue, we add the new stream into the singleton with count=1.

The third I/O in the following stream (item ->count > =3) can be considered as sequential and hint can be created.

After the hint is created, the allocation of data on the tiers can be decided. This is depicted in Figure 5. If the hint is set to true, then we say that the sequential writes are detected and we can allocate the data on the slower tier. If the hint is false then we can continue allocating the pages on the faster tier treating the I/O as random.

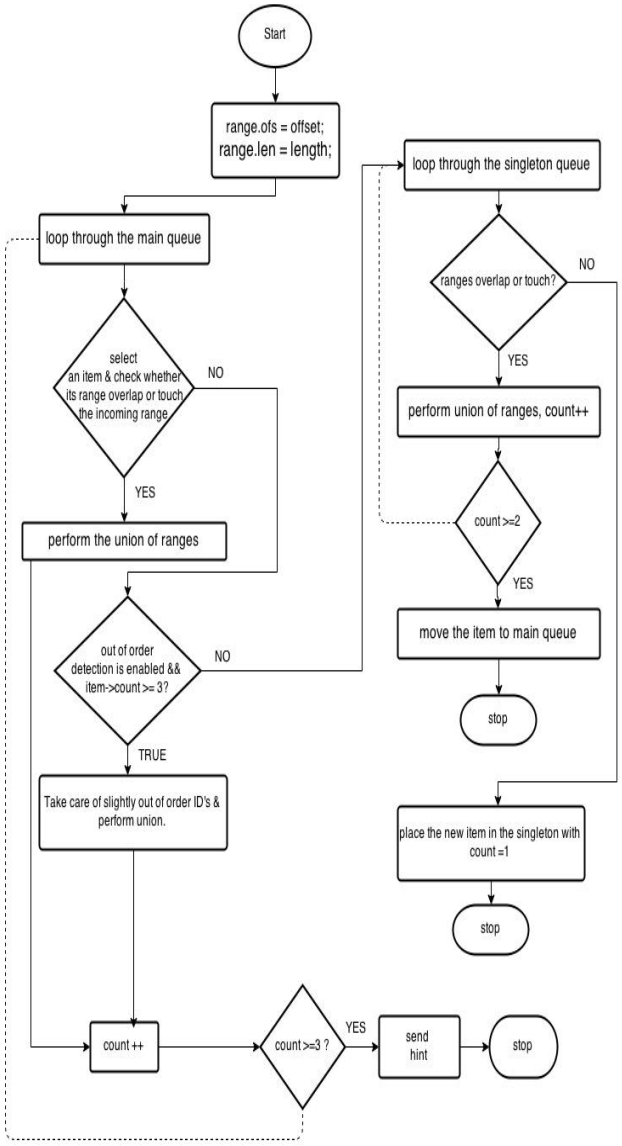

Figure 1: Algorithm of sequential write detection

Figure 2: Touching I/O's.

Figure 3: Overlapping I/O's.

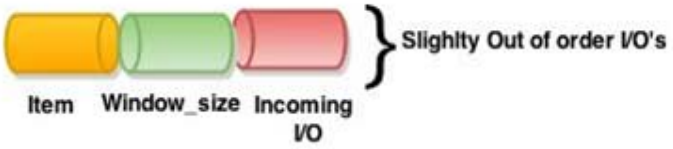

Figure 4: Slighlty out of order I/O's with window_size enabled.

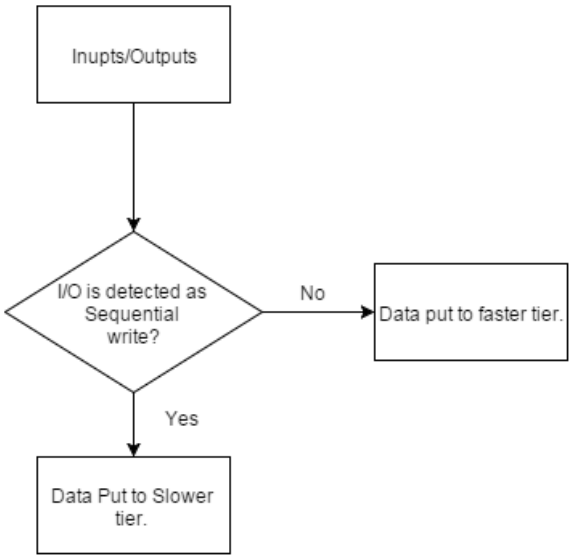

Figure 5: Allocation of data on the tiers.

V. RESULTS

As seen from the Figure 6 before the implementation of the sequential write detection algorithm, all the data occupied the faster tier, i.e., Tier 0 as can be seen from the Space Consumed attribute. It shows that the entire 1GB is occupied on Tier 0 and 0GB is occupied on Tier1. The Graph of the sequential write and random write can be seen in Figure 7 wherein both the random writes and sequential writes occupied full tier 0 space with no

differentiation between sequential and random I/O. The orange colored line depicts random writes and after passage of time occupies full capacity of tier 0 space. The black colored line depicts the data distribution of random writes on tier 1. The green colored line demonstrates the sequential write workloads and distribution of data on tier 0. It too occupies 100% space on Tier 0 with time. The blue line shows the data distribution of sequential writes on Tier 1 and is not occupied as shown by the graph of line 0.

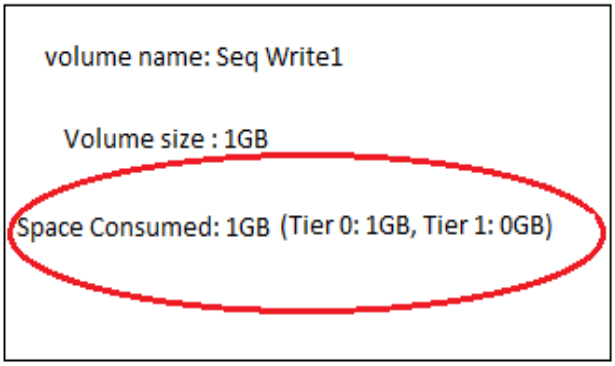

Figure 6: The space consumed in the two tiers on sequential write without the sequential write detection algorithm applied.

After the implementation of the sequential detection algorithm there were changes wherein only some data occupied Tier 0 and remaining data occupied tier 1. This can be seen in the Space Consumed attribute of Figure 8. Tier 0 contains only 8 MB of the data and remaining data of 1016MB is occupied on Tier 1.

The graph is also plotted as shown in the Figure 9 where it shows that for the sequential writes, some data occupy tier 0 while remaining data occupy the tier 1 storage. As shown in the graph the Orange line depicts that random writes, occupy 100% of the tier 0 storage as usual. The Green line shows the percent utilization of tier 0 by a sequential writes and it occupies only small part of tier 0 and remaining data goes to the tier 1 which is shown by Blue colored line. Thus there is differentiation between the random writes and Sequential writes and thus occupies a small part of tier 0 in case of sequential writes and all the remaining data occupy the tier 1 storage.

VI. CONCLUSION

In the storage industry there are constant efforts to optimize storage space to provide high performance which is a value add to the business by saving the cost and utilizing the resources effectively. Other benefits of the above solution is improved performance for important applications by distributing data carefully on the tiers. The proposed solution can be used to make efficient use of the storage resources and especially resources such as SSD's which are very critical for high performance. Figure 8 and Figure 9 capture the algorithm's results. With the above results we can bifurcate I/O's into sequential and random and there by optimize the SSD's space for random writes.

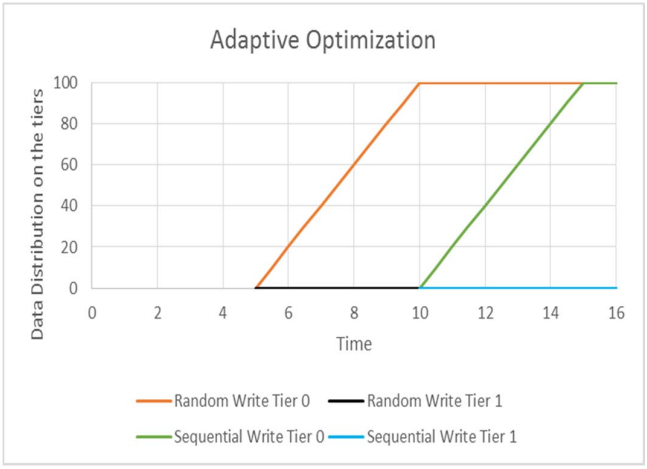

Figure 7: The distribution of data in the two tiers graphically.

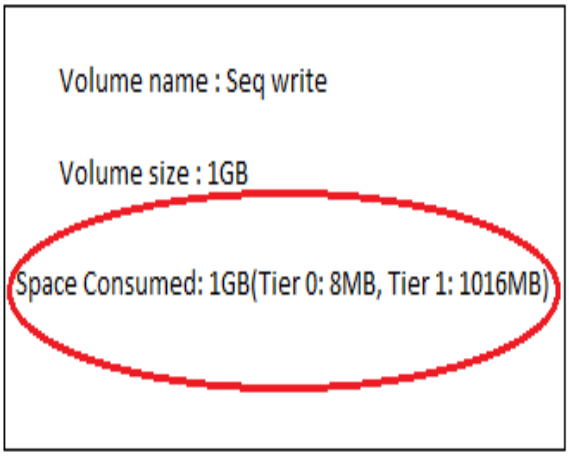

Figure 8: The space consumed in the two tiers after applying the algorithm.

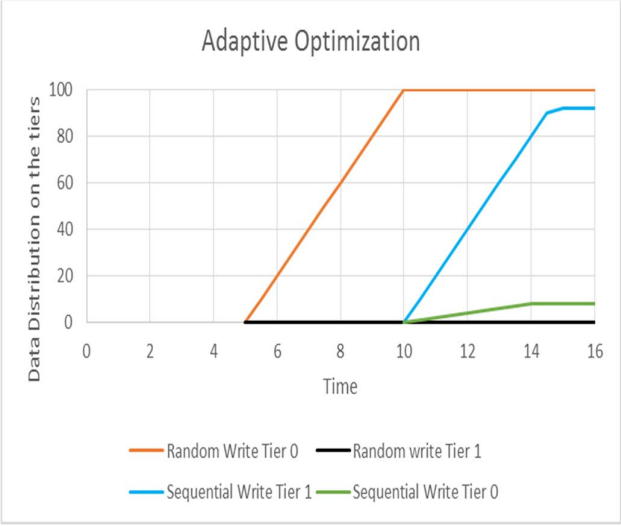

Figure 9: Graphical representation of distribution of data in the two tiers

REFERENCES

[1] Amit S. Chavan, Kartik R. Nayak, Keval D. Vora, Manish D. Purohit and Pramila M. Chawan, "A Comparison of Page Replacement Algorithms", International Journal of Engineering and Technology, Vol.3, No.2, April 2011.

[2] Nimrod Mgiddo Dharmendra S. Modha, "Outperforming LRU with an Adaptive Replacement Cache Algorithm", IBM Almaden Research Center.

[3] Bhatta Jagdish, Saud Arjun Singh, "Recency and Prior Probability (RPP) based Page Replacement Policy to Cope with Weak Locality Workloads having Probabilistic Pattern", International Journal of Computer Applications (0975 – 8887) Volume 59– No.15, December 2012.

[4] William Stallings , "Operating Systems: Internals and Design Principles" 7th Edition.

[5] Silberschatz, Galvin and Gagne, "Operating System concepts" , Eight Edition.

[6] Ulf Troppens, Rainer Erkens and Wolfgang Muller, John Wiley & Sons , "Storage Networks Explained" , 2003.

[7] SANs Richard Barker and Paul Massiglia, "Storage Area Network Essentials: A Complete Guide to understanding and Implementing" ,John Wiley India, 2002

[8] Ari, Gottwals, M. ; Henze, D , "SANBoost: automated SAN-level caching in storage area network", ISBN: 0-7695-2114-2, 17-18 May 2004.

[9] Hasan M H Owda , Munam Ali Shah, Abuelgasim Ibrahim Musa, Manzoor Ilahi Tamimy ,"A Comparison of Page Replacement Algorithms in Linux Memory Management", International Journal of Computer and Information Technology (ISSN: 2279 – 0764) Volume 03 – Issue 03, May 2014.

[10] N. Meigiddo, and D. S. Modha, "ARC: A Self-Tuning, Low overhead Replacement Cache", IEEE Transactions on Computers, pp. 58-65, 2004.

[11] Pancham, Deepak Chaudhary, Ruchin Gupta, "Comparison of Cache Page Replacement Techniques to Enhance Cache Memory Performance", International Journal of Computer Applications (0975 – 8887) Volume 98– No.19, July 2014.

[12] Tucson , "Relative Competitive Analysis of Cache Replacement Policies" LCTES'08, Jan Reineke Daniel Grund, June 12–13, 2008, Arizona, USA. Copyrightc 2008 ACM.

[13] Andrew S Tanenbaum ,"Modern Operating Systerm third edition".

[14] S. Albers, S. Arora, and S. Khanna, "Page replacement for general caching problems," Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, pp. 31–40, 1999.

[15] S. Jiang, and X. Zhang, "LIRS: An Efficient Policy to improve Buffer Cache Performance",IEEE Transcations on Computers, pp. 939-952, 2005.

[16] Nayaka B Govindaraja , Majeed Zameer , Taranisen Mohanta, "Policy Driven Dynamic LUN Space Optimization based on the Utilization", Proc. of Int. Conf. on Advances in Communication, Network, and Computing 2013.