International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 12 28– 32

An Advanced Caching Solution to Cluster Storage Environment

Aaishazun Basheer
M. Tech in Information Technology
Dept. of ISE,R. V. College of Engineering
Bengaluru, India
aaishazun@gmail.com

Dr. ShantharamNayak
Professor, Dept. of ISE,
R. V. College of Engineering
Bengaluru, India
shantaram_nayak@yahoo.com

Abstract- Clustered storage is the deployment of multiple data servers working together to improve reliability, capacity and performance. Clustering divides workloads to every storage server to control and monitor workload transfer and file access between servers without taking into account of the physical location of the file. Solid State Drives (SSD) can be considered as a more sophisticated version of a USB memory stick since the memory stick does not have any moving part associated with it and moreover, data is stored in microchips.

In this paper, we give an overview of an advanced caching solution to improve IO and application performance by using flash storage in cluster storage environment. It is a cluster storage solution with two highly scalable servers with optimizations to ensure fast service failovers and deploying one or two solid state drives as the cache devices for faster and better performance.

The software supports write-back caching policy where both read and write requests on hot regions of drivesare cached. With write-back, write requests to the hot regions are acknowledged immediately after it is written to the cache device and this (dirty) data will be flushed to back-end virtual drive in the background. Flushing of dirty data will be performed by the flush manager of the software under different scenarios like amount dirty data reaches a threshold, IO activity during a time interval is low etc. The solution effectively harnesses the flash storage performance potential by retaining only frequently accessed data in flash for quick retrieval. The solution provides unmatched efficiency, performance, support and reliability for enterprises or storage world.

Keywords: *Cluster Storage Environment, Solid State Device (SSD), Hard Disk Drive (HDD), Caching, Flushing*.

*****

I. INTRODUCTION

An SSD is based on NAND-based flash memory which is non-volatile so that even if the disk is turned off, the data is still retained and remains intact [1]. The advantages of a HDD is that compared to an SSD, it stores lots of data cheaply. But, it is slow in read/write performance, emits noise and fragmentation affects the performance. Storage I/O is very important when it comes to cluster storage environment and should be scalable with unmatched performance and efficiency. A Cluster storage solution [6] is employed with two highly scalable servers with optimizations to ensure fast service failovers and deploying one or two SSDs as the cache devices for faster and better performance.

Caching is the technique of storing a copy of data temporarily in memory so that further requests to read them are serviced from the cache. The proposed system uses flash storage as cache and retains only frequently accessed data in cache for quick retrieval [2]. There are three types of caching which are write-through caching, write-around caching and write-back caching.

- Write-through cache: redirects write I/O onto cache and to permanent storage before acknowledging I/O completion.
- Write-around cache: write I/O is directed to permanent storage bypassing the cache.
- Write-back cache: write I/O is directed to cache and completion is acknowledged to the host immediately [3].

The proposed system uses write-back caching. Another feature of the system is it handles pinned cache while SSDs are going offline/online.

II. MOTIVATION

The proposed system has many benefits which are:

- Best suited for small IO, highly random block-oriented applications frequently read from a working unit.
- Improves throughput performance.
- Write-back caching and hence low latency and high throughput for write-intensive applications.
- Failover/failback design to ensure fast service failovers when servers are integrated [5].

III. METHODOLOGY

The solution is an advanced caching solution designed to improve IO and application performance by using flash storage in cluster storage environment. The solution effectively harnesses the flash storage performance potential

by retaining only frequently accessed data in flash for quick retrieval.

Two controllers are connected to a common drive pool containing SSDs and HDDs. Each controller has its own instance of the software running. Each controller contains a set of SSDs and HDDs configured as a group under the software. Application ensures that the two groups have different drives from the common pool. Suppose that there are two controllers A & B such that group "G-A" is configured on controller A and "G-B" on controller B. When controller A fails, failover command will be executed on controller B upon which "G-A" (the configuration which existed on controller A) will be activated on controller B. However "G-A" configuration will not be updated with host ID of controller B. When controller A comes back, during boot up, it triggers failback command on controller B. With this, the newly activated cache group "G-A" would be removed from controller A. Since Host ID of G-A was unchanged, G-A would be activated on controller A during boot.

The procedures involved while running I/Os are:

- To flush cached data to disc.
- To disable/enable caching (cache invalidation).
- To set/get flush parameters while flushing cached data.
- Failover support.
- To handle pinned cache while SSDs or HDDs going offline/online.
- Cache Device and Disc Failure Handling.

A. OBJECTIVES

- Solution provides unmatched efficiency, performance, support and reliability for enterprises or storage world.
- Flash storage as cache device offers a significant improvement in I/O performance and reliability.
- Retains only frequently accessed data in cache device for fast retrieval.
- To handle pinned cache while SSDs or HDDs going offline/online.
- Integration of two data storage servers to eliminate single points of failure.

B. SYSTEM ARCHITECTURE

The software uses Flash memory technology to enhance performance for those applications that are primarily stored on spinning HDD media [7],[8]. The solution accelerates the performance of applications that use either Storage Area Network (SAN) attached or Direct Attached Storage (DAS) by identifying the frequently accessed data. This data is copied into low latency flash storage.

The solution improves the I/O performance to meet the requirements of the high-performance applications and uses the high-performance SSDs as a secondary tier of cache to provide faster read and write access to maximize the transactional I/O performance of the application. The solution uses the write-back caching policy and is designed to accelerate the I/O performance of the applications that are limited by the HDD performance by only requiring a small investment in the flash technology-based solutions.

The solution consists of filter driver, caching library, and management solution. The filter driver filters I/Os that are destined to VDs. The caching library determines the hot regions in the VDs. The management solution consists of Storage ManagerTM application and CLI utilities that can be used to manage the solution. The system architecture is shown in figure 1.

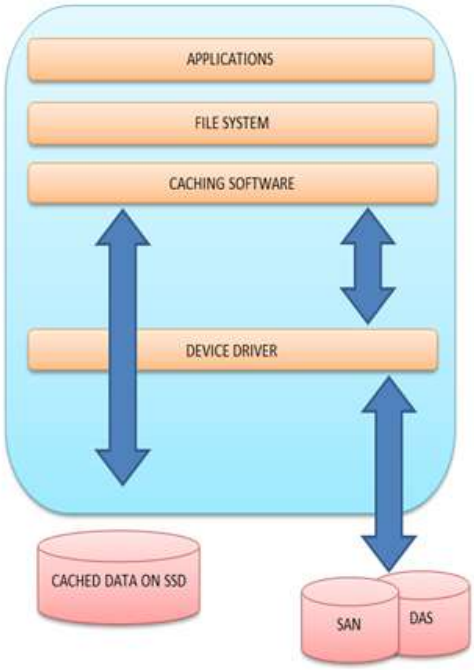

Figure 1: System Architecture

C. DETAILED DESIGN

The detailed design of the software implemented is shown in figure 2. Here, filter driver and OS dependent functions are implemented as device mapper target drivers. Core caching library is compiled as a Linux kernel module with well-defined APIs. Work at the block layer is transparent to file system and applications. It consumes flash devices and provides caching function across DAS/SAN volumes. Core caching function is implemented as an OS portable library with well-defined interfaces. Filter Driver in OS stack intercepts IO and routes through Cache Management Library for caching functions.

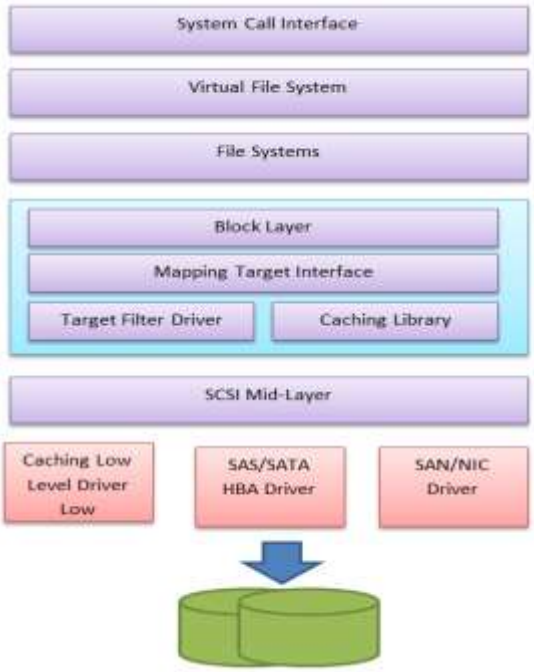

Figure 2: Linux Driver Architecture

The caching library architecture is shown in figure 3. Here, the platform interfaces are abstracted to enable portability across environments. A filter driver is required in the OS IO stack to configure and route IOs through the library. The library scales to represent multi-TB Flash/SSD devices. There are configurable parameters to control caching decisions. In-memory lookup tables are present for fast cache lookups. Intelligent multi queue page replacement algorithm is used for replacement and AVL based cache flush engine is used for efficient flush to the HDDs. It also supports cache hinting to pin a range of cache LBAs.

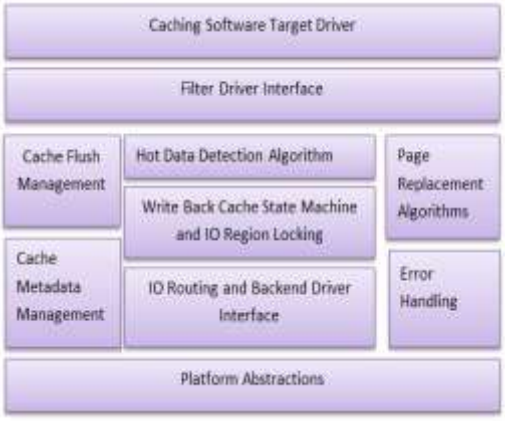

Figure 3 Caching Library Architecture

D. THEORY OF OPERATION

The solution uses a Kernel driver to review any storage I/O transaction destined for logical disks (also indicated as VDs) assigned to be cached. The caching software is a Block Device Mapper driver in Linux. Using its caching algorithms, the solution determines hot region of frequent I/O activity. The address ranges are determined by the caching algorithms and the write storage I/O transactions within this range is stored upon the Flash technology located on SSD and also into the SAN or DAS storage. The read storage I/O transactions within the address range is quickly accessed from the drive. As the data within the cache cools, it is replaced with hotter data.

i) Caching Solution

The solution offers the caching feature that can be controlled through the CLI management applications. The features are included as a package. At the time of system startup, the hardware with the software features is detected and enabled on any one or all of the SSDs connected to the server. The SAN or DAS storage that are being accelerated are referred to as cached devices or logical drives (VDs).

ii) Read Cache Persistence

The read cache metadata persists onto the cache device only during graceful shutdown of the system. It will not be persisted in case of ungraceful shutdown of the system like system crash, power-recycle etc. After successfully saving the metadata, a bit in the metadata header, called trust bit, will be updated and written to the cache device to indicate that read cache metadata on the cache device is now valid. On subsequent system start, boot code and driver init code will check this bit to decide whether the read cache is valid or not.

Filter driver will register for shutdown notification from operating system in the disk device start-device routine for the devices it filters. Filter driver will notify library about the system shutdown event using a newly exported library function which will save the in-memory copy of the metadata on to the cache device. After the metadata update is complete, trust bit in the metadata header buffer will be updated and written on the cache device. Trust bit will be cleared and written to cache device during metadata initialization to make sure that the boot code and driver initialization code ignores the read cache on subsequent ungraceful shutdown and restart of the computer.

iii)Write-back Metadata support

Metadata region will store the Cache Window metadata information for all the cache windows generated from that cache device. Since the cache policy is write-back, this data will be kept in sync (with respect to the dirty bits), with the in-memory metadata buffer. This is achieved by updating metadata whenever the in-memory dirty bitmap is modified. Valid bit map will be updated with clean shutdown of the system.

iv) Metadata Initialization

The cache policy is write-back and hence, memory for the metadata will be allocated and initialized for each cache device that is added to the cache group. The size of metadata is computed based on the number of cache windows needed for the cache device. One instance of cache window metadata structure will be maintained for each cache windows and hence size of metadata buffer will be number of windows multiplied by size of cache window metadata structure. Metadata entries in the buffer are indexed by the cache device window-number and stores corresponding VD window-number, valid bitmap dirty data bitmap and a VD identifier.

When the cache device is added to the cache group for the first time, metadata header region on the cache device will be initialized with the default values. Metadata region on the cache device and in-memory metadata buffer will be zeroed out.

v)Metadata Logging

On successful completion of a write request to the cache device, in-memory metadata for the cache line(s) and/or cache window(s) will be updated by setting the corresponding bits in the dirty data bitmap. Then, this will be made persistent by sending metadata update request to the cache device by invoking a function that initiates metadata update request (when all the child requests are completed for a write-back request and for a flush request). This function will generate metadata update request(s) for the page(s) in which the windows falls and completion of the write-back/flush request will be delayed until metadata update is completed. State of the request will be updated to indicate that dirty tree info needs to be updated and scheduled to the scheduler.

On clean shutdown of the system, the entire metadata will be persisted onto cache device. A flag in the metadata header will also be updated to indicate that metadata (both dirty and valid bitmaps) are valid which will be used by the boot code to validate whether entire metadata is trustable or not.

vi)Metadata Restore

On system reboot, filter driver will read the persistent configuration from the cache device and will apply the configuration to the library by adding the cache device and VD to the cache group. Now, library will read the metadata header, validate it and then read metadata into the in-memory buffer it allocated as part of cache device initialization. This will be followed by library building the caching data structures (hash table, dirty tree, etc), using the metadata in the in-memory buffer.

vii) Flushing Mechanism

A logical entity called flush manger will be defined in the caching library which can be initialized whenever cache policy is set to write-back and can be destroyed whenever cache policy is set to anything other than write-back [4]. Different parameters that are required for the flush logic can be set at the initialization time using the interface functions defined for the flush manger. Flush manager will create a background thread which on creation will wait on a semaphore till it is signaled to start the flush of dirty data. Interface functions provided to start flush of dirty data under different scenarios will set the appropriate parameters and wakes up the thread by signaling the semaphore.

Scenarios to start flush:

- Amount of dirty data hits the threshold. This will be set to 2/3 of total cache capacity.
- No or very low IO activity for a time duration.
- A VD with dirty data is being taken out of the cache group.
- Cache policy is being changed from write-back to any other policy.

Flush Logic:

When signaled, flush manager thread take following actions.

- Stops if shutdown flag is set.
- If there are pending device flush request, start flushing those VDs.
- If there are no pending device flush request, start flushing dirty data in a round robin fashion starting next VD after last_checked_vd.

Flush IO Handling:

A new IO request flag will be defined to differentiate flush request while handling completion of IO request. Child read requests will be created for each continuous run of dirty regions in the dirty window and will be dispatched to cache device. On successful completion of the child request, it will be sent to VD as a write request. If that also succeeds, metadata will be updated to reflect that data has been flushed.

E. PERFORMANCE

The performance graphs for random 100% writes with SSDs and HDDs are presented in figures 2 and 3.

IV. CONCLUSION

Caching is the technique of storing a copy of data temporarily in memory so that further requests to read them are serviced from the cache.

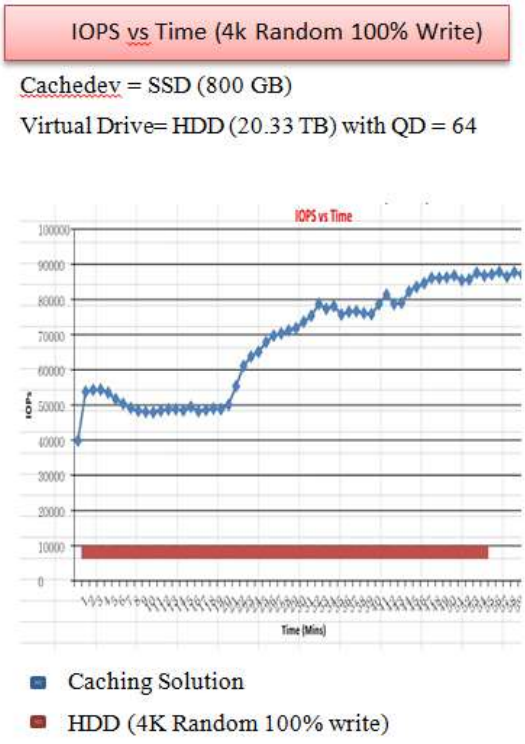

Figure 4: IOPS vs Time Performance Graph

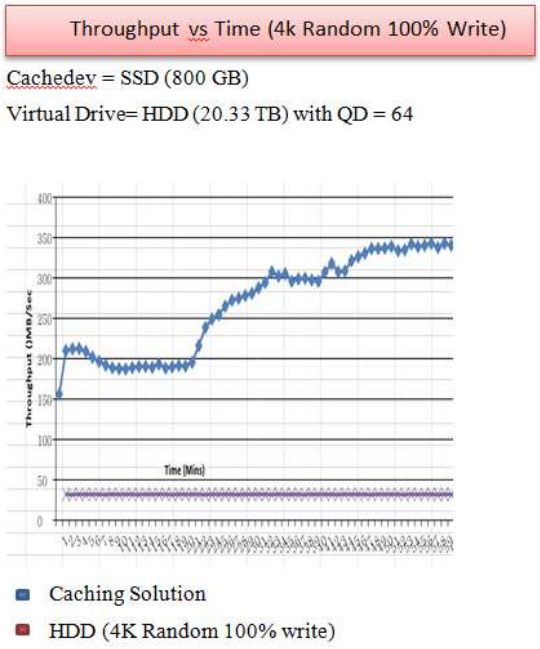

Figure 5: Throughput vs Time Performance Graph

Proposed solution uses NAND – based flash memory for caching because of their high speed and Hard Disk Drives as permanent storage because of their large capacity to store data in cluster storage environment. Two storage servers are integrated to support failover/failback of servers.

This approach can be widely used in large enterprises where data is continuously moving in and out and also in storage world which handles large I/O. The limitations are:

- SSDs as cache devices are costly compared to HDDs.
- Best suited for small IO, highly random block-oriented applications frequently read from a working unit.
- Only two data servers can be integrated together to support failover/failback.

REFERENCES

[1] D. Jiang, Y. Che, J. Xiong and X. Ma, "uCache: A Utility-Aware Multilevel SSD Cache Management Policy," High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, Zhangjiajie, 2013, pp. 391-398.

[2] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu and L. Zhou, "S-CAVE: Effective SSD caching to improve virtual machine storage performance," Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh, 2013, pp. 103-112.

[3] Y. Ko, R. Jeyapaul, Y. Kim, K. Lee and A. Shrivastava, "Guidelines to design parity protected write-back L1 datacache," 2015 52nd ACM/EDAC/IEEE DesignAutomation Conference (DAC), San Francisco, CA, 2015, pp. 1-6.

[4] C. Wang, Q. Wei, J. Yang, C. Chen and M. Xue, "How to be consistent with persistent memory? An evaluation approach," Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on, Boston, MA, 2015, pp. 186-194.

[5] W. S. Ling, O. B. Yaik and L. S. Yue, "High availability resource monitoring solution using simplified team formation algorithm," Frontiers of Communications, Networks and Applications (ICFCNA 2014 -Malaysia), International Conference on, Kuala Lumpur, 2014, pp. 1-5.

[6] S. Agarwala and R. Routray, "Cluster aware storage resource provisioning in a data center," 2010 IEEE Network Operations and Management Symposium - NOMS 2010, Osaka, 2010, pp. 647-660.

[7] H. Takishita, S. Ning and K. Takeuchi, "Trade-off of performance, reliability and cost of SCM/NAND flash hybrid SSD," 2015 Silicon Nanoelectronics Workshop (SNW), Kyoto, 2015, pp. 1-2.

[8] C. Sun, T. O. Iwasaki, T. Onagi, K. Johguchi and K. Takeuchi, "Cost, Capacity, and Performance Analyses for Hybrid SCM/NAND Flash SSD," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 61, no. 8, pp. 2360-2369, Aug. 2014.