2014 1st International Congress on Computer, Electronics, Electrical, and Communication Engineering (ICCEECE2014)
IPCSIT vol. 59 (2014) © (2014) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2014.V59.6

Data Comparison Algorithms for Remote Replication Storage Systems

Dhanaraj Maruthachalam [1], Taranisen Mohanta [1] and Amulya Ratna Swain [2]

[1] Storage R&D Lab, HP India Software Operations, Bangalore, India

[2] School of Computer Engineering, KIIT University, Bhubaneswar, India

dhanarajm2003@yahoo.com, taranisen_mohanta@yahoo.com, and swainamulya@gmail.com

Abstract. In Storage Area Networks (SAN), Remote Replication (RR) is a solution that provides business continuity by making use of highly available logical disks. It is achieved by establishing relationship between a minimum of two storage arrays. In general, the second storage array is remotely located and hence, the logical disks continue to be accessible during disasters in one of the locations. One or more number of logical disks are pooled together to create a group. This group is then replicated to the destination using proprietary mechanisms. A host application is interested in comparing data (Auto-compare) between two logical disks that are owned by two different arrays. In this paper, we propose a couple of methods to perform the Auto-compare efficiently and then introduce a self-learning algorithm which dynamically utilizes the free storage resources to improve the storage performance. In order to perform the load balancing, the best suited method is dynamically decided based on the resource utilization/availability and hence, it helps to improve the overall throughput of storage arrays. The performance of various methods is compared using theoretical analysis and simulation results.

Keywords: Storage; SAN; LUN; SCSI; LDisk; Remote Replication; Auto-compare; checksum; IO;

1. Introduction

Traditionally, the host or server is responsible for running a set of required applications, handling the client requests, storing and managing the data. When the amount of storage space grows drastically, the performance of a host system comes down. A SAN provides an external storage management facility to the host by which the performance of host is not impacted. A typical SAN system is illustrated in Fig. 1. In this figure, the storage array is a separate entity that capable of storing and managing the data [1][2]. A storage array consists of one or more controllers and back-end disks. The host talks to the storage array using SCSI protocol. When a host and Storage array are connected by Fiber channel cables, the storage array can be placed with the distance ranging from a few meters to thousands of kilometres [3]. A large number of disks are connected to a storage array using FC-AL or SAS protocols. A storage array is responsible to manage the disks and provide the storage space with RAID (Redundant Array of Independent Disks) support. A Logical Unit Number (LUN) or Logical Disk (LDisk) is a basic storage entity which is exported to the host. A host performs the read/write (IO) operations on the LDisk which has the size ranging from a few gigabytes to many terabytes.

Remote Replication (RR) allows business continuity seamlessly. In order to provide access to LDisks without downtime during the failure of software/hardware problems, dual controller design is adapted throughout the storage industry. In case of disasters, the entire site becomes non-operational. In order to provide continuous access during disasters, the dual controller design is extended to the next level and named as Remote Replication [4]. Fig. 2 illustrates the RR storage systems. The host and the first storage array are located locally and the second storage array is located in a remote site. The connection is established between the storage arrays and host. A set of LDisks are grouped and replicated across the two sites. This is called remote replication group. One of the storage arrays acts as a source which first receives the data from host. Then, the source array replicates the data to the destination array. This one-to-one (source-destination) configuration can also be extended as one-to-many and cascading RR relationships [5].

Based on the mode of operation, a destination LDisk is either in-sync or out-of-sync with the source LDisk. In case of a synchronous mode, the LDisks are synchronized with each other and a site failure allows

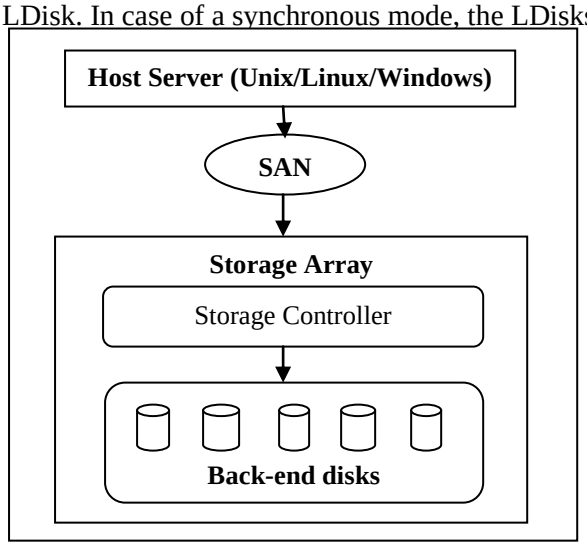

Fig. 1: Illustration of a SAN system

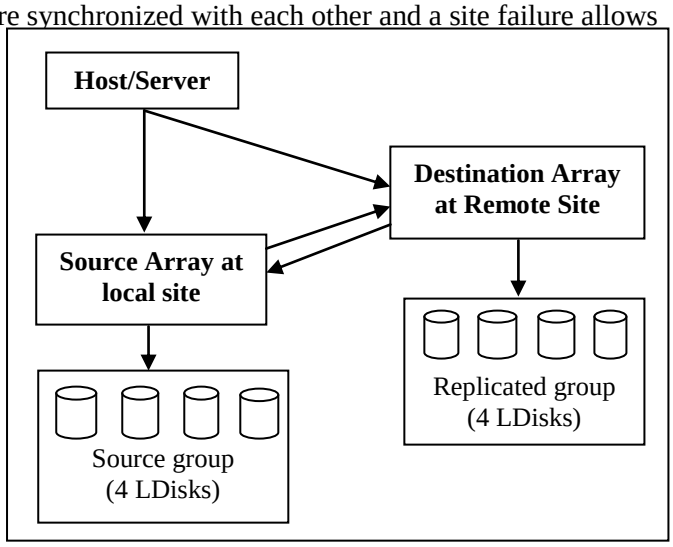

Fig. 2: A typical Remote Replication Storage System

the host to use the destination LDisk seamlessly. A host application compares (Auto-compare) data between source and destination for the sake of maintaining data integrity or security purposes. After the data is read from both arrays and transferred to the host, the host compares the data. This consumes resources at both arrays. If the operation is initiated for an entire LDisk, it can affect the performance of on-going IOs.

2. Related work

In SAN and other storage systems, remote replication becomes necessary during the disaster recovery situations. Most products from the key storage vendors like HP, EMC, NetApp, Hitachi, and IBM implement such solutions for business critical data. XP, 3PAR, EVA, Symmetrix VMAX, and FAS are the successful products in the storage market [5][6][7]. Due to software/hardware issues, the data gets corrupted. It is dangerous, when the corruption occurs without any notification. Since the data is replicated in RR systems, there is a facility to check the correctness of the data through Auto-compare. Many of the above mentioned products provide this feature. 3PAR has the implementation of basic method, which is explained in Section-3. The checksum is used in Unix/linux/windows systems to compare files [8]. To the best of our knowledge, the checksum based comparison and self-learning are the techniques which are not yet explored in RR system.

3. Our Contribution

Auto-compare is one of the resource consuming operations due to read/compare operations for the entire disk data. During this process, the two impacted resources are CPU cycles and network bandwidth which are critical to SAN performance. Our first goal is to offload the tasks from the host to one of the storage arrays. This helps in reducing the network traffic which is proposed as basic method. The next method calculates the checksum and then compares the checksum rather than comparing the entire data blocks. This Checksum method also reduces the network traffic. Then, we introduce a self-learning algorithm which dynamically decides the best suited method and the array in which the comparison has to be performed.

3.1. Traditional Data comparison algorithm

In SAN, SCSI-verify command is internally triggered to perform Auto-compare [9]. The detailed steps during the Auto-compare operation are as given below.

1) Read block of data from source disk (T_{read1}) and transfer to host (T_{tfr1})
2) Read the same from destination disk from same logical address (T_{read2}) and transfer to host (T_{tfr2})
3) Compare and verify data between these blocks (T_{cmp})
4) If data is same then increment the logical address and go back to step 1. If data is different, raise an alarm for data inconsistency. If the logical address reaches the end of the LDisk then exit.

T_{total} is the total time needed to complete Auto-compare for a data block and it is calculated by adding the time periods of all the above five operations. After the comparison is completed, the user can take the relevant steps necessary to either correct the data, or disregard the data altogether. From the above, it is

noticed that the most time intensive and IO intensive operation occur in step 1 and step 2. The host CPU gets busy with processing many IOs based on the size of the LDisk.

3.2. Basic and Checksum methods

The comparison of the data block is carried out at source array, instead of performing it at the host. The destination array sends the data to the source array, instead of sending to the host and hence, the comparison can be performed at the source array itself. This reduces the network traffic by 50%. The comparison result is communicated to the host from the source array. This approach is called basic method.

Checksum method is explained as follows: A source array reads the data block by block and calculates the checksum for an entire stripe. It maintains a queue of stripes that are read and checksums. It also maintains a read indicator and a verify indicator. The source array sends the read requests to the destination array and asks it to send the checksum of the address specified. The read indicator is progressed as and when the requests are sent out. When checksum is received from the destination array, the data in the queue of stripes is checked using the verify indicator. If the checksum matches, then the verify indicator is progressed.

The checksum method reduces the network bandwidth at the cost of consuming extra CPU cycles. This method is preferred in the following scenarios: 1) if system has network bandwidth limitations 2) if there is a pre-calculated checksum stored and 3) if arrays have rich CPU resources.

4. Novel Self-learning Algorithm

The storage configuration and availability of resources vary across the storage products. Hence, the Auto-compare can be dynamically adapted accordingly. The self-learning algorithm has two selection policies such as comparison method and storage array selection. In case of checksum method, there is a trade-off between the network bandwidth and the CPU cycles to calculate the checksum. For a given storage array configuration and workload, one method performs better than another method. The storage arrays can be made aware of a self-learning algorithm, which identifies the best suited method dynamically. This is the first selection policy.

In case of Basic and Checksum methods, the comparison of data/checksum can be performed either at the source or destination array. This can also be dynamically decided through the self-learning algorithm. In order to balance the load across the storage arrays, this array selection policy is implemented.

The array keeps track of the resource availability/utilization parameters like network bandwidth/load, CPU load, IO performance/load, etc. In addition, the storage arrays can measure T_{total} for the following five combinations: 1) basic method at source array 2) checksum method at source array 3) basic method at destination array 4) checksum method at destination array and 5) checksum method at host. Then, the array can dynamically decide one of the combinations which produce the minimum response time. This algorithm needs to be run periodically and re-election to be made dynamically based on the best suited policy.

Some storage products store the checksum along with the data block. Then, the checksum based comparison can be opted directly. In general, the Auto-compare operation is not a priority operation, unless a data corruption is suspected. The process/thread which performs the comparison operation is given the lower priority and hence, the on-going IO performance is not impacted.

5. Theoretical Analysis

In this section, the proposed methods are compared using theoretical analysis. As mentioned in Section-3, the time taken for each operation is measured using T_{read1}, T_{read2}, T_{tfr1}, T_{tfr2}, and T_{cmp}. T_{total} is the total time taken to complete the comparison of one (or) a set of blocks.

For traditional data comparison algorithm, T_{total} is given in equation (1). Since there is no optimization, the sum of all the elements is measured as T_{total}. For Basic method, the data transfer time is reduced by 50% and hence, equation (2) reflects that one of the transfer time elements is taken out. For checksum method, the equation (3) shows the formula for calculating total time and the same is simplified to (4). The time taken to perform checksum is measured using T_{cksum1} and T_{cksum2}. The time taken for transferring the checksum to source array and perform the comparison of checksum are measured using T_{ckTfr2} and $T_{cmpCksum}$ respectively. If the storage array has the in-built checksum, then the total time is measured using equation (5). The time taken to read checksum from disk is measured using $T_{rdCksum1}$, $T_{rdCksum2}$.

$$T_{total} = T_{read1} + T_{read2} + T_{tfr1} + T_{tfr2} + T_{cmp} \Rightarrow (2 \times T_{read}) + (2 \times T_{tfr}) + T_{cmp} \quad (1)$$

$$T_{total} = T_{read1} + T_{read2} + T_{tfr2} + T_{cmp} \Rightarrow (2 \times T_{read}) + T_{tfr} + T_{cmp} \quad (2)$$

$$T_{total} = T_{read1} + T_{read2} + T_{cksum1} + T_{cksum2} + T_{ckTfr2} + T_{cmpCksum} \quad (3)$$

$$= (2 \times T_{read}) + (2 \times T_{cksum}) + T_{ckTfr2} + T_{cmpCksum} \quad (4)$$

$$T_{total} = T_{rdCksum1} + T_{rdCksum2} + T_{ckTfr2} + T_{cmpCkSum} \quad (5)$$

It is hard to formulate the resource configuration and system workload. Even though the total time is not derived for the self-learning algorithm, it is understandable that this algorithm outperforms other two methods. We support our theory using the simulation results.

6. Simulation results

Here, we compare the performance of different Auto-compare methods. We have used HP storage simulator to simulate different methods. The SAN/FC bandwidth is configured to be 4GB. The backend disks are setup with RAID-0 configuration.

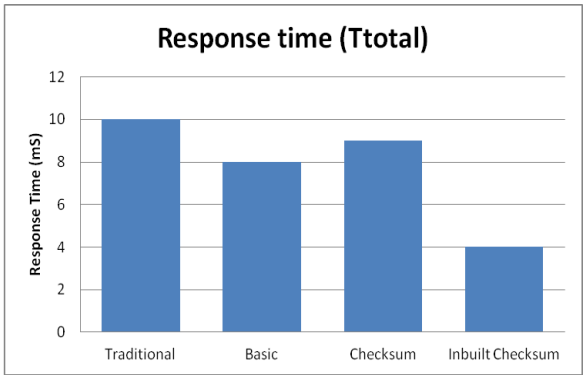

Fig. 3: Total time for various methods

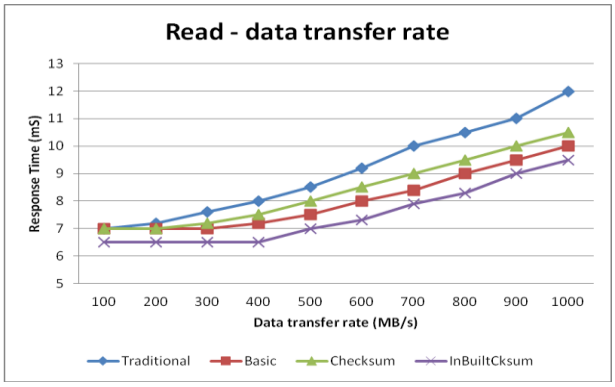

Fig. 4: Read workload performance

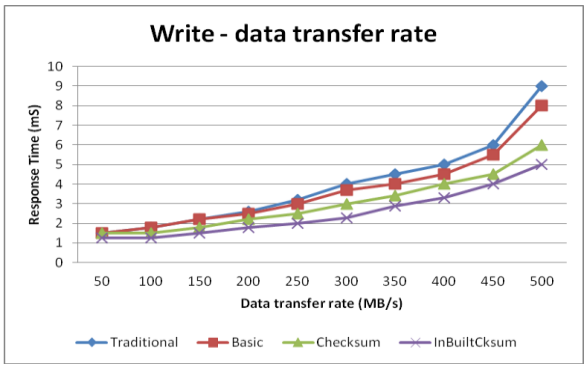

Fig. 5: Write workload performance

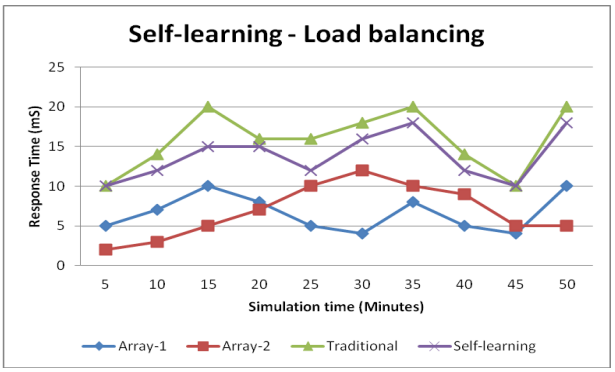

Fig. 6: Self-learning algorithm performance

6.1. Total response time

This simulation is performed, when there is no workload supplied to storage system. The average time to compare 100MB data is measured. Fig. 3 shows T_{total} response time for different methods. The simulation results reflect the performance that we formulated in Section-5. The basic and checksum methods reduce the latency by 20% and 10% respectively. The checksum computation is an overhead and consumes the extra time. In all the results, the inbuilt-checksum method produces the best results, because Auto-compare reads the checksum directly from disk instead of reading the data and calculating the checksum.

6.2. Impact on performance for varying workload

Fig. 4 shows the response time for a varying read workload. The Auto-compare is performed in the back-ground and the response time for the host IO is measured. The Auto-compare is configured to run with the rate of 500MB/s and 500MB data is compared across source and destination arrays every second. The host is configured to send the random read requests to get the data size of 8KB block per request. The X-axis indicates the read data rate at the host in terms of MB/s and Y-axis indicates the response time per request in terms of mSec. As the workload increases, the response time increases drastically for all methods. The performance of basic method is better than that of checksum method, because the checksum method consumes the extra CPU cycles.

Fig. 5 shows the performance of different methods for a varying write data workload. The Auto-compare is configured to run with the same rate of 500MB/s. The host is configured to send the random write requests to write the data size of 8KB block per request. As the workload increases, the response time increases drastically for all methods. Fig. 5 shows that the checksum method out-performs the basic method.

Fig. 6 illustrates the performance of self-learning algorithm which achieves the lowest response time by selecting the best suited method. The X-axis shows the simulation time in terms of minutes and Y-axis shows the response time per request in terms of mSec. At different time intervals, the workload of array-1 and array-2 are not evenly distributed and hence, the response time differs. The response time of Auto-compare methods (Traditional and self-learning) is compared in the same figure. The total number of IO requests is varied from 5000 to 20000. The auto-compare also performs the data comparison for 8KB block size and the average response time for 8KB data comparison is plotted in the figure.

7. Conclusion

In SAN, the remote replication is a key feature to achieve high-availability across the sites. The verification of data consistency is important for business critical data. Hence, the Auto-compare is playing an important role. By doing Auto-compare effectively, the over-all resource utilization and the performance of storage system are improved. We proposed basic and checksum methods which improve the network bandwidth utilization. Then, we proposed a self-learning algorithm which dynamically executes the two selection policies such as Auto-compare method and array selection. The simulation results proved that the self-learning algorithm improves the Auto-compare performance up to 20%. Even though the Auto-compare performance is tightly coupled with the storage configuration and available resources, our algorithm finds the best suited way to achieve the best performance and load-balancing across the sites.

8. References

[1] Leong, D. A new revolution in enterprise storage architecture. *IEEE Potentials*, vol.28, no.6, pp.32-33, Nov-Dec 2009.

[2] Katal, A., Gupta, N., Sharma, S., Goudar, R. H. Information storage on the cloud: A survey of effective storage management system. *Students Conference on Engineering and Systems (SCES)*, 2012, pp. 1-6, March 2012.

[3] M. Sachs and A. Varma. Fibre Channel and Related Standards. *In IEEE Communications Magazine*, vol. 34(8), pp. 40-50, 1996.

[4] http://www.computerweekly.com/feature/Remote-replication-Comparing-data-replication-methods

[5] ftp://65.38.160.56/pub/3par/Documentation/3PAR-RemoteCopy-wp-09.0.pdf

[6] http://www.emc.com/storage/symmetrix-vmax/srdf.htm

[7] http://h18000.www1.hp.com/products/quickspecs/11617_div/11617_div.PDF

[8] http://www.softpedia.com/get/System/File-Management/Checksum-Compare.shtml

[9] http://www.t10.org/scsi-3.htm

Dhanaraj Maruthachalam was born in Coimbatore and completed B.E (CSE) degree from Bharathiar University, Tamil Nadu in 2002. He has done M.S (by research) from IIT Madras in 2005 with 5 IEEE conference publications in the area of wireless sensor networks. He started his career with Sun Microsystems and worked on Solaris kernel components. Then, he has worked on various storage products and his current job title is System Specialist in HP, Bangalore. He has 8 years of industrial experience and the research interests include operating systems and storage.

Taranisen Mohanta was born in Orissa and done his M. Tech from IIT, Delhi (Electrical Energy System) in 1999. He has worked for Alcatel, Oracle and Hewlett-Packard in system and application software. He currently works as Storage Architect in HP, Bangalore. He has 15 conference and journal publication in storage and cloud. He has filed 3 patents. He has guided 2 M. Tech thesis . He has 13 years of industrial experience and the research interests includes in operating systems , storage and cloud.

Amulya Ratna Swain received his PhD degree from the Department of Computer Science and Automation, Indian Institute of Science (IISc), Bangalore, India. Currently, he is working as an associate professor in the School of Computer Engineering, KIIT University, India. He has 4 years of teaching experience. His research interests include wireless sensor networks, distributed computing and operating systems.