Virtual Point in time access

Assaf Natanzon * Eitan Bachmat †

Abstract

Continuous Data Protection or CDP is a method for capturing all changes occurring to a storage device, allowing fine granularity restore of objects from crash consistent images. In this paper we introduce a method for creating a virtual image of a block storage device,using a CDP journal log and an image of the device at one point in time. The creation of the disk image for any point in time is created on demand. The creation algorithm is very efficient and takes only a few minutes for multiple TeraBytes of changes. The algorithm for creating the image can be formalized as a map/reduce algorithm and can be parallelized easily over multiple machines to reduce the creation time. The creation on demand of the virtual image allows a much more efficient handling of the data inside the journal of the CDP and allows the use of CDPs for enterprise class applications.

1. Introduction

Continuous data protection (CDP) is a feature which allows the system to role back a given data set or the entire data to its state in any given point in time. The creation of snapshots of data sets, i.e., frozen in time copies of a dataset as it existed at some given time have been present in enterprise level (mainframe) storage systems since the 90's [1]. However, the resources needed to support multiple snapshots of the same data set increase with the number of points in time. This means that this method cannot scale to provide continuous rolling back to any point in time.

There are multiple recovery use cases for which the user may use a CDP. One major use case is single object recovery. A single object may be a single file or a single email. To recover the single object, a virtual image of the point in time state is created and the file system containing the desired object is mounted to access the image. Once the host can access the file system on the logical unit (LU) at the time before the object was lost or corrupted, the object can be restored and copied to the production environment.

Another use case is Disaster Recovery (DR) testing. The user may want to test an image to see if an application can recover from the specific image. To test such a hypothesis, the system may create a virtual image of the point in time that the user wishes to test and the user may run his hosts and test the image. One major example is the VmWare site recovery manager (SRM) which allows a full DR test of a replicated virtual environment, [16].

One more use case is fail-over. Usually users fail-over using a replica LU. One option is to roll the journal to the point in time that the user requested, and allow the user to directly access the logical unit. The main issue with this approach is that rolling to an old image may take significant amounts of time. Using the virtual image of the volume, the user may access the image immediately.

In the present paper we describe an efficient implementation of CDP that is widely used by thousands of enterprise customers, allowing fine granular recovery of objects and storage devices. This is the first commercially available implementation of a CDP. Our method is to create a full copy of the original volume and a journal of the changes. The full copy requires more data but if the journal is long lived, the journal size is more significant than the full copy. The full copy along with the CDP journal may be stored at a remote site allowing replication integrated with continuous data protection and disaster recovery. In cases where the full copy is stored at the local array, it may be stored as an efficient snapshot allowing space savings.

Access to a point in time is done by building a virtual image of the block device being protected by using pointers to either the journal or the copy of the volume. Building the point in time is done in an efficient way, on demand, and requires no resources during the normal mode of access. The system allows the creation of a virtual point in time both for single object recovery (e.g. recovery of a single file or a single mail box), or for full fail-over to a previous point in time. The system allows full restore of the virtual image in the replica storage device while users access the point

* Assaf Natanzon, EMC Corp.,Tel Aviv, Israel and department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel, 84105. assaf.natanzon@emc.com

† Eitan Bachmat, Department of Computer science, Ben-Gurion University, Beer-Sheva, Israel, 84105. ebachmat@cs.bgu.ac.il

in time image, the system rolls the changes from the journal to the replica volume and once rolling is complete the system allows users to directly access the replica volume. Our system allows rolling of the data from a journal to the replica device, while users have real time access to the device. Once the rolling of the image completes, the user hosts access the replica device directly, or though a copy on first write mechanism. The CDP has been implemented commercially as part of the EMC RecoverPoint data replication and protection system.

The next section presents some background on the CDP system, and the management of the journal.

Section 3 discusses the method and algorithms for creation of the virtual image of the disk.

Section 4 describes some performance analysis of both creation of the virtual disk image and access to the virtual disk image.

Section 5 summarizes our main conclusions and discusses future work.

2. Description of a network based, continuous data replication solution

In this section we describe Recover-Point, which is an example of a network based continuous data protection and replication scheme. The replication system is composed of two pieces, A *splitter* and a *data protection appliance*.

The splitter, intercepts the I/O arriving at the data path, and mirrors the I/O to both the network data protection appliance, and the original storage target. The splitter is installed either at the replicated host, or inside a fabric switch, or in a storage array. The splitter can intercept data operations at several logical levels. The system implements a splitter for replication of block level devices which intercepts I/O at the `SCSI` layer.

The data protection appliance is a physical or virtual appliance, which exposes a SCSI target. It receives I/O coming from a splitter and replicates them over a WAN to a data protection appliance at the replica site. The replication appliance at the replica site manages the remote copy.

The replica data protection appliance manages a *journal* of the data at the replica site as well as a full copy of the data.

The network based replication system, allows creation of consistency groups consisting of several volumes. Since several splitters can send the I/O operations to the same data protection appliance, write order fidelity can be achieved between volumes from separate storage arrays at a single data protection appliance.

The journal is a log containing the sequence of changes that happened to each of the volumes in the consistency group. There are two streams of data in the journal, a *redo log*, containing changes which occurred to the production volume and not yet applied to the replica volume, and an *undo log* containing a list of changes which undoes the latest changes in the replica volumes.

Each entry in a journal contains, apart from the I/O data itself, the following I/O meta data:

1) A volume ID.
2) The I/O block offset within the volume.
3) The I/O length.
4) A Time stamp of the I/O.

The meta data and the data of the journal are stored in different streams. The fast creation of a point in time image of a volume is based on the creation of the image using only the meta data of the changes which occurred in the volume. The meta data in the journal also contains information whether the point in time after the write, described by the meta data, is consistent. It may also contain some user created strings describing the point in time, e.g., a bookmark. The user may create a bookmark whenever desired, each bookmark is a consistent point in time. If the user desires to create an application consistent point in time, the user needs to put the application in a consistent mode such as Oracle hot backup mode or Microsoft VSS, and request the system to create a bookmark while the application is in the consistent mode. Creation of a bookmark is just a logical procedure which adds messages with the bookmark info into the I/O stream, and thus bookmark creation typically takes less than a millisecond.

The journal is managed in the following way:

1) An I/O arrives at the replica site, and is written to the replica *redo log*. The I/Os at the replica may be cached and there is no need to flush every I/O upon arrival.

2) A batch of I/Os is read from the *redo log*.

3) Undo of the batch of I/Os read at step 2 is read from the replica volume.

4) The undo information read at step 3 is flushed to the *undo log*.

5) I/Os freed at step 2 are flushed to the replica volumes.

some implementations may discard the first two steps of journal management and use only an *undo log*. In this case, the journal management is a simple copy on write implementation. The advantages of first flushing the I/Os to a *redo log* is the sequential character of the log I/Os which allows handling of significantly higher bursts of data.

Creating an undo log only, is possible without having a full replica copy of the volume, but has impact on the production response time as each write operation results in an additional read operation at the production volume.

One major advantage of using a *redo log* is that replication of production volumes may continue while the user accesses a point in time. New I/Os which arrive are flushed to the *redo log.* We also implemented compression at the journal level, since writes to the journal are sequential keeping data compressed in the journal does not cause major alignment issues and can typically save

3. Algorithms for creation of a virtual image

The user can select a point in time for either recovery of a single object or for a complete fail-over of the environment. once the user chooses the image he wishes to access, the system will build the requested image and allow the hosts to access the logical unit (LU) containing the object. The LU is exposed with the data it had at the point in time the user requested. the next subsection describes how the image is created using a map/reduce type algorithm. Unlike other continuous data protection systems, the image is created on demand. During normal operation, the behavior is similar to copy on write snapshots.

3.1 Building the virtual image

In order to present the user meta data an image of the LU at the time the user created, the system must create a data structure which answers the following query:

Given an offset and a length, where are the relevant blocks located?

If the location was referenced in the undo area of the journal, the query must point to the earliest location in the *undo log* where the region has changed. If the region has not changed, the data in the replica LU is the correct one.

The data structure is created in the following way:

The relevant areas of the journal's undo log are sorted according to offset. The relevant entries in the undo log are all entries which are newer than the point in time the user requested. A filter is applied and only the oldest version for each offset is kept. The sorted entries of the image are written to a special log device *meta data log device*. The meta data log device is a list of sorted entries, each entry has the following content:

1. The start offset in the LU which the entry describes (8 bytes).
2. The number of blocks the entry describes (4 bytes).
3. The offset in the *undo journal* which the entry describes (8 bytes).

The entries are sorted according to their offset in the logical unit. In order to allow fast random access to entries in the *meta data log device* the system also creates a table of contents to allow fast searching of the meta data log device, the table of contents is small enough so that the system can hold the pointer table in the memory. The access table is a mapping between the offset in the logical unit and the offset in the meta data log device. The table allows fast finding of offsets in the meta log device. The data structure holding the virtual device structure is illustrated in figure 1.

Creation of the data structure can be formalized as a map reduce algorithm using the following formulation:

Each mapper gets a portion of the *undo log* meta data stream, and maps each entry to a reducer based on the starting offsets the entry covers. The volume is divided into

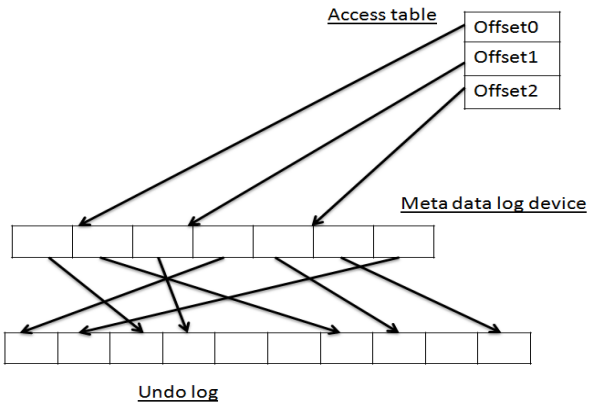

Figure 1. data structure of the virtual device.

ranges of constant sizes (say 1MB), and the mapping function maps each entry according to the range of the I/O described in the meta data. Entries which cover multiple ranges are split. The reducer receives the list of entries and finds the entry which is oldest (i.e. the entry which has the smallest pointer in the undo stream).

```
function MAP(RedoLogStartPtr,Len)
    for each entry ∈ RedoLogPortion do
        if entryRange covers multiple ranges then
            split entry
        end if
        emit (entryRange ,entry,RedoLogPtr)
    end for
end function

function REDUCE(entryOffsetRange,entry,RedoLogPtr)
    for each offset ∈ entryOffsetRange do
        Ptr ⇐ min(RedoLogPtr)
        filter entry potrion for which Ptr is minimal
        emit (entry,Ptr)
    end for
end function
```

figure 2 illustrates how the map/reduce algorithm works on the undo log data. In our implementation we implemented the algorithm on a single computer, but utilizing multiple cores. The algorithms can be easily implemented using multiple nodes for even higher performance.

3.2 Accessing the virtual image

The system exposes the virtual images of the logical unit using the original SCSI identity of the logical unit. We consider implementing the logical unit exposure using a different identity, thus allowing access to multiple points in time of the volume simultaneously. Since the original SCSI identity of the logical unit is exposed to the host, the *splitter* needs to be aware that the volume is currently being virtualized to a previous point in time. When an I/O request is intercepted by the *splitter* and the *splitter* identifies that

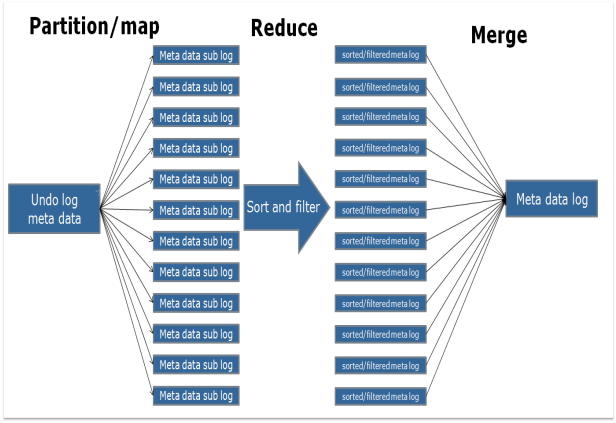

Figure 2. Description of the map reduce algorithm.

the I/O is to a volume being in a virtual access mode, the I/O is redirected by the splitter to the replication appliance and not to the original logical unit that the host is configured to see. The replication appliance then uses the virtual image data structure described in the previous sub section to handle read commands.

When the splitter is hosted inside a storage array, the situation is a bit more complex because of some locking mechanisms that the storage arrays utilizes. However, to simplify our discussion, we may at first assume that the splitter is installed at a host and thus all I/Os directed to a logical unit in virtual access mode, are redirected to the replication appliance. When reading data from a virtualized point in time, parts of the data may still be stored on the original volume, while part of the data is written in the journal. When the splitter is installed inside a storage array and a read directed to the original device is intercepted, the appliance will not be able to read directly from the original volume since the storage system is locking the location which is currently being read. In such cases the splitter installed in the storage array will read the data stored at the storage array, it will send the current data stored in the storage to the replication appliance, which will add the changes to the data which are stored in the journal and return the correct read data for the point in time to the storage.

Writes are handled in a very efficient way. The writes are redirected to a special log device, the *virtual access redo log device*, and a mapping is kept to know which locations should be served from the log device and which should be served from the original device. Since virtual access is a temporary mode (the user will either recover a single item, or will do a short DR test, or will roll the image), writes are simply written sequentially to the log device. If a write occurs again to the same location it will still be written to the end of the redo log, hence the entire process is composed of sequential writing.

Read handling is a little bit more complicated. When a read command is intercepted at the replication appliance, first the write log snapshot mechanism is consulted. If the read is to a location which was overwritten after the virtual image was created, the data structure holding the mapping to the virtual access redo log will notify where in the redo log the data is held. The read will be answered from the data within the redo log. If the data is not in the redo log, the data is either in the system *undo log* or it is in the original volume. The access table is consulted, using the access table the system reads the relevant entries from the *meta data log device*, if a pointer to the location in the *undo log*, which contains data for the offsets read, is found in the *meta data log device*, data is read from the *undo log* otherwise data is read directly from the volume. Optimizations of caching read data may be applied to improve the performance of accessing the virtual volume.

Our system allows rolling of the data from an *undo log* and the *virtual access redo log device*, while users access the replica volume is virtual image access mode. Once the rolling of the image completes, the splitter stops redirecting the I/Os to the replication appliance, and directs reads to the original replica volumes. Writes may still redirected to the replication appliance so the replication appliance may handle them in a copy on first write manner. The rolling process reads the data from the undo log and applies the data to the replica volume. The pointer in the data structure are being updated to understand a location may have transfered from the undo log to the volume. Once the system finishes transferring the *undo log* data to the replica volume, the system starts moving data from the *virtual access redo log device* to the replica volume. When all data is moved to the replica volume the splitter changes the redirection of the I/Os.

When the splitter is installed inside a storage array, there need to be differentiation between I/Os arriving from a replication appliance and IOs arriving from a user host. The splitter gets an indication on the initiator type for each I/O, and writes arriving from the replication appliance are directed to the replica volume and not split. This is done to allow the update of the replica volume to the point in time accessed by the user so that the user will be able to fail-over to the the device and write and read to it directly.

4. Experimental results

4.1 Experimental setup

For hardware, we used a RecoverPoint GEN4 data protection appliances The appliance has $8192MB$ of RAM, 2 quad core CPU a QLogic QLE2564 quad-port PCIe-to-8Gbps Fibre Channel Adapter. The replication appliance was connected to a CLARiiON CX4-480 storage array with 30 FibreChannel-attached disks, configured as 6 separate RAID5, $4 + 1$ RAID groups. We configured a consistency group replicating 12 production volumes. The 12 production volumes were created on 4 separate RAID5 4+1 groups, 3

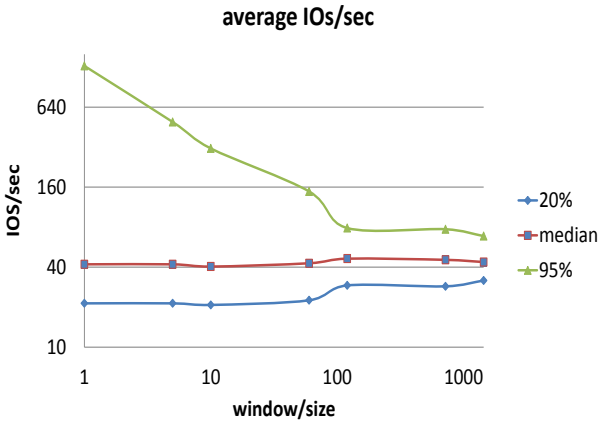

Figure 3. Average I/O per second, per time window

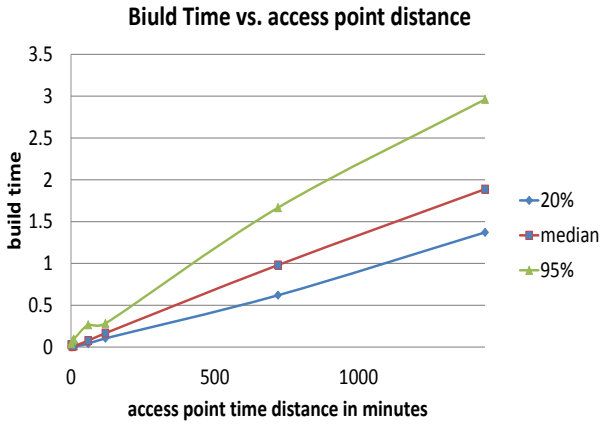

Figure 4. Average build time

volumes were created on each RAID group. The journal was striped over two volumes from two separate RAID groups.

4.2 Virtual image creation time

The Creation time of a virtual image depends on the amount of meta data which represents the changes of the image. The creation algorithm described above basically reads the meta data, then writes it to multiple streams of data. Later, each stream of data is read, filtered and sorted in memory and then written to another stream. All I/O operations are highly sequential and thus very efficient. The creation of a point in time data structure requires two sequential read operations and another two sequential write operations. Once the data is read into memory it is sorted and filtered using a sort tree. The running time is $O(nlog(n))$, where n is the amount of meta data entries. Each meta data contains 32 bytes of data (4 byte volume ID, 8 bytes volume offset,4 byte transaction length, 8 byte time stamp and 8 byte pointer to the data in journal which the meta data describes).

Sorting the entries using currently available CPU power is less strenuous in comparison to the I/O load generated in the process. Our current implementation is able to handle around 64MB of meta data per second (which generates around 256MB of I/Os per second). The creation time of the virtual image also requires transforming the configuration changes to the rest of the system and the splitters. This process might take several additional seconds before the user may access the volume. This means that the building of the virtual image access data structure can process meta data at a rate of roughly of 2M IOPS. It is important to note that the test were made in a sterile environment, where all the application I/O were pauses and no other activity except the activity of the virtual image creation was allowed. In real environment the storage array at the replica site would probably be loaded with activity of multiple consistency groups being actively replicated, and also replication for the consistency group being tested for virtual access would continue in a non sterile environment.

Figure 3 describes the average number of I/O operations per second observed for over 500 customer applications from 20 different customers from multiple industries. The graph describes the average amount of I/O for applications over multiple time windows, e.g. 1 minute window, 5 minute window, 10 minute window and 1 hour, two hours, 12 hours and a full day.

When looking at a specific time window, for example, an open hour window, we look at the average activity for every one hour period for each application and take the median of these activity, the 95 and the 20 percentiles of the activity. We present the average of the medians for all the appliances for each time period, and this is the graph presented as the middle graph of figure 3. For example, when looking at 1 minute windows, the average of the 95 percentile most active minutes for all the applications is 1300 IOPS, while the average of medians is roughly 40 I/O /sec. If we look at a daily average we see that the difference between the average median daily and the average peak daily activity is much smaller, on a daily average there are around 40 IO/sec for the median and around 70 IO/sec for the top 95 percent active day.

Figure 4 describes the average amount of time the creation of the virtual access data structure takes. As can be observed, building an image for a one day old point in time takes on average about 3 seconds, which is practically immediate, if we take into account that just booting the machine would take a much more significant amount of time. The creation time grows roughly linearly with the point in time distance since the number of changes grows roughly linearly with time.

Since averages can be very misleading, we also present data specific to several randomly selected customers. We chose randomly 5 customers from the 20 customers for which the above analysis was done, for each customer we chose the most active application which was replicated and took the 95 percent most active period of time for each such application. Figure 5 shows the amount of IOPS each cus-

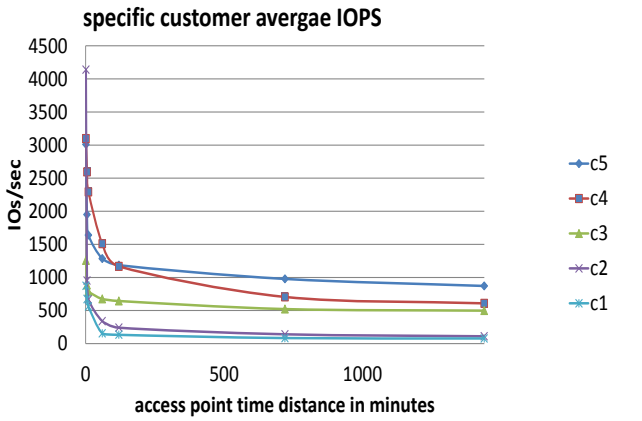

Figure 5. Average IOPS for specific customers

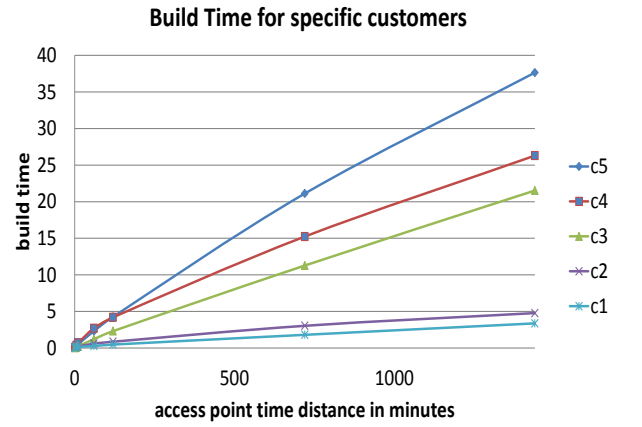

Figure 6. Average build time for selected customers

tomer had. For instance, the most active customer had almost 5000 IOPS on his most active application on the 95 percent most active minute, and almost 1000IOPs on average per second on a daily base. Figure 6 shows that even for such a customer building the image of a point in time 24 hours ago would take roughly 40 seconds. which is still a relativity short period.

Even for extremely active applications which generate 20000 IOPS on average the system can create a virtual image from 24 hours ago in roughly 10 minutes.

4.3 Access time

While building the virtual access data structure is relatively fast, access is significantly slower than accessing volumes directly. Write I/O access is relatively painless, since I/Os are basically logged in the log device. since the appliances are not battery backed up no write caching is allowed, thus writes must be flushed immediately to the log device on the storage. We have tested random read and random write performance in our lab for several test cases. The results are described in figure 6. We tested completely random read and writes. Direct results are the performance when writing directly to the storage volumes. Since we wrote fully random 8KB I/Os and the back-end storage for the devices contained only 2 separate RAID 5 groups, the IOPS are rather low. As we can see the random writes directly to the storage volumes reach around 1800IOPS and random reads to the volume achieve 1200IOPS. When accessing the system in virtual access write I/Os become sequential to the disk, since they are basically journaled sequentially, thus the write performance reaches almost 7000 IOPs when the queue depth of the writes is 16. The write performance is also not dependent on the distance between the virtual image and the physical image (i.e. even if there are many changes in the virtual access data structure the write performance is not affected). Reads on the other hand are very dependent on the distance between the virtual and current image of the volume. The graph describing virtual random reads describes an access to a virtual image close to the physical image, meaning the virtual access data structure is almost empty. In such a case each read translates to an access to the journal for checking where the data exists, and then most of the time finding out the data is in the original volume. The performance is degraded compared to direct reads since there is an extra hop to the replication appliance, and also since we did not yet implement a caching layer for the read meta data and thus there is an extra read which can be avoided. As can be seen when we send one I/O at a time we reach around 150IOPs compared to 250 IOPS in direct access, and the ratio remains similar when the queue depth is 16 and we reach around 1200 IOPS directly and 750 IOPS through virtual access.

The long distance virtual read access graph shows a very long distance point in time access, where most if not all of the volume has changed. In this case since we limited ourselves to 8KB random reads and writes, each read request will most likely translate into two separate reads from the journal, and since the system is drive bound the performance degrades accordingly. We get around 100 IOPS when reading single I/O at a time and reach around 550 IOPS at a queue depth of 16.

4.4 Related work

There are several methods of implementing continuous data protection systems. Some implementations of CDP systems add CDP functionality to the production storage adding snapshot like mechanisms to the storage array,[5] [6] [7]. In some implementations, only an undo log is implemented. The full replica copy, together with the journal provides a Continuous Data Protection (CDP) system, allowing the user to access any point in time. Another example of a CDP system, which differs substantially from the one we implemented, is presented in [7]. A system which offers multiple point in time views can also be found in [12]. CDP is an extension of storage array snapshots, which were introduced much earlier, but instead of allowing single point in time CDP allows access to any point in time in the past. Snapshot mechanisms are presented at [14]. Our systems inte-

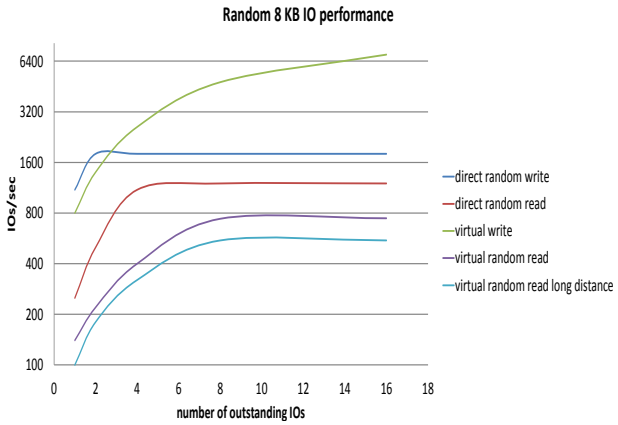

Figure 7. Average response time vs. number of outstanding IOs

grates replication with CDP, for more background on replication technologies see [15] Efficient asynchronous replication techniques are described at [4] and [3]. Replication solutions are also used for allowing simultaneous access to a distributed volume across multiple sites, and implementation of such a geo-replicated system is described in [2] .

5. future work

Our testing of the access time for reads and write for virtual image access were very limited. We did the tests in our labs and tested two extreme scenarios, one scenarios where most of the data is in the storage array, and another where most data is split across two separate locations in the journal. Write performance in virtual image access less dependent on the IO patterns, and in fact in random access behaviors we see significant performance improvements using virtual access. Reads performance in virtual access heavily depend on the access pattern as a single reads can be translated to multiple reads depending on the write access of the production volume. Unfortunately we currently do not collect read information on our appliance. Getting read access pattern from customers will allow us to simulate the behavior of reads in the lab and provide a more accurate estimation for the performance of read access, and to how many reads on average on read really translates too. Since most of the customers use the feature for single file recovery from point in time the build performance is much more critical than the actual read performance, in the future we can implement additional caching layers for the data structure in order to achieve better read performance. Since once the data structure is built the access becomes similar to file systems like WAFL and ZFS we do believe proper caching can achieve high performance for read access as well. Another interesting research is to journal using flash devices, since flash devices are very high performing, virtual access to the data can perform in an almost production like performance with flash device. Another research are is to integrate virtual access with array snapshot, and allow the system to keep periodical array snapshots along with journals, since in such implementation the journal can be significantly smaller the virtual access will most likely read data from the snapshot. Such implementation will require more storage , since some of the changes will be stored in both snapshot and journal. since the journal has sequential characteristics, it can be stored on lower end devices like SATA disk, or even on a different cheaper storage array since loss of the journal when integrated with snapshot just means loss of the any point in time granularity.

References

[1] Hitz D., Lau J., Malcolm M.A. 1994. File System Design for an NFS File Server Appliance, *Proceedings USENIX Winter Conference*, 235-246.

[2] Transactional storage for geo-replicated systems, SOSP Yair Sovran, Russell Power, Marcos K. Aguilera, Jinyang Li

[3] Yan R., Shu J. and Chan Wen D. 2004. An implementation of semi-synchronous remote mirroring system for SANs, In ACM Workshop of Grid and Cooperative Computing (GCC).

[4] Weatherspoon H., Ganesh L., Marian T., Balakrishnan M., and Birman K. 2009. Smoke and Mirrors: Reflecting Files at a Geographically Remote Location Without Loss of Performance, Proceedings of FAST 09, the 7th USENIX Conf. on File and Storage Technologies, 211-224.

[5] L. Shrira and H. Xu, Snap: Efficient snapshots for back-in-time execution, in ICDE05: Proceedings of the 21st International Conference on Data Engineering. Washington, DC, USA: IEEE Computer Society, 2005.

[6] L. Shrira, C. van Ingen, and R. Shaull, Time travel in the virtualized past: Cheap fares and first class seats, Haifa Systems and Storage Conference, SYSTOR 2007.

[7] R. Shaull, L. Shrira and X. Hao, Skippy: a New Indexing Method for Long-Lived Snapshots in the Storage Manager, *ACM SIGMOD Conference 2008*,Vancouver, Canada,2008.

[8] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, S.T. Klein: The design of a similarity based deduplication system. SYSTOR 2009: 6.

[9] B. Zhu, K. Li and H. Patterson, Avoiding the Disk Bottleneck in the Data Domain Deduplication File System, Proc. of FAST 08, the 6th USENIX Conf. on File and Storage Technologies, 279-292, 2008.

[10] A. Azagury, M. Factor, and W. Micka, Advanced functions for storage subsystems: Supporting continuous availability. An IBM SYSTEM Journal, 2003.

[11] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality, Proceedings of USENIX File And Storage Systems conference (FAST), 2009.

[12] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules and G. R. Ganger. Self-securing storage: protecting data in compromised systems. In Proc. of the 4th OSDI, pages 165-180.

[13] R.H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman and S. Owara, SnapMirror: File-System-Based

Asynchronous Mirroring for Disaster Recovery, Proc. of FAST 02, the 1th USENIX Conf. on File and Storage Technologies,117-129, 2002.

[14] L. Shrira and H. Xu, Thresher: An efficient storage manager for copy-on-write snapshots, in USENIX 06: Proceedings. Berkeley, CA, USA: Advanced Computer Systems Association, 2006.

[15] Ji M., Veitch A. and Wilkes J. 2003. Seneca: remote mirroring done write, Proceedings of USENIX Technical Conference (San Antonio, TX), 253-268, June 2003.

[16] VMWare Site recovery manager http://www.vmware.com/products/site-recovery-manager/overview.html