CoprHD: ScaleIO Driver based on Southbound SDK

AN ABSTRACT OF THE THESIS OF

Prathamesh Pramod Patkar for the degree of Master of Science in Computer Science presented on December 5th, 2016.

Title: CoprHD: ScaleIO Driver based on Southbound SDK.

Abstract approved: _____

Dr. Rakesh Bobba

Software Defined Storage is a term for data storage software to manage policy-based provisioning and management of heterogeneous data storage system abstracting underlying hardware. CoprHD is a software defined storage controller and API platform which enables policy-based management and cloud automation of storage resources for block, object and file storage providers. CoprHD is a management software which creates abstraction layer over the heterogeneous storage and controls them. Primary aim of this project is to implement storage driver for the ScaleIO storage system on CoprHD, with a new Southbound SDK which act as intermediate driver layer (abstraction layer). South-Bound SDK interfaces storage device and CoprHD which enables a developer to code without understanding complex system. The project is comprehensive process of back ground study for software defined storage and setting up development environment to design, implement and test driver for volume protection operation clone.

CoprHD: ScaleIO Driver based on Southbound SDK

By

Prathamesh Pramod Patkar

A PROJECT

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Presented December 5th, 2016

Commencement June 2017

Master of Science project of Prathamesh Patkar presented on December 5th 2016

APPROVED:

Dr. Rakesh Bobba, representing Electrical Engineering and Computer Science

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Prathamesh Pramod Patkar, Author

ACKNOWLEDGEMENTS

I would first like to thank my Project Advisor Dr. Rakesh Bobba of EECS Department at OSU. The door to Dr. Bobba's office was always open whenever I ran into trouble or had a question about my academics, project or research. He consistently allowed me to do my own work, but steered me in the right direction whenever he thought I needed it. His courses such as Distributed System and System Security helped me to create foundation for my project.

I would like to thank my Committee Members Dr. Bella Bose and Dr. Eric Walkingshaw for constant support and guidance. Dr. Bella Bose (EECS Head) for his constant support and guidance during my master's program. Dr. Eric Walkingshaw Programming Languages helped me to think about programming languages in a different way.

I would like to thank my Manager and my project guide Shayne Huddleston, Director at IT infrastructure Services OSU, for providing me an opportunity to work on the project. His constant support and guidance was responsible for steering the project towards an accomplishment. I would like to thank the experts involved in this Project: Evgeny Roytman, Jason Davidson from EMC Corporation(Dell-EMC) and Anjaneya Chagam (Reddy) Principal Engineer at Intel for his guidance and supports towards our team. Without their passionate participation and input, this survey would not have been successfully completed. I would also like to acknowledge my team members Shujin Wu, Varun Rajgopal and Taylor Cuilty for being supportive and grateful.

I would like to grateful towards EECS department OSU and Nicole Thompson, for constantly providing me financial support in the form of GTA throughout my project and support.

Finally, I must express my very profound gratitude to my Parents, Mr. Pramod Patkar and Mrs. Pradnya Patkar; Shruti and Chaitali for providing me unfailing support and continuous

encouragement throughout my years of study, and through process of development. This accomplishment would not have been possible without them. Thank you

Prathamesh Patkar

Table of Content

Page

List of Figures

Page

1. Introduction:

In IT Infrastructure world, storage vendors are promoting an idea of purchasing a single system which would manage every type of storage, but this isn't always in the best interest of the consumer. Such All-in-one solution typically limit the IT team's ability to address specific issues or meet current budget realities. Heterogeneous(Mixed) Storage system environments can provide this flexibility, but it always increases the cost of management. For e.g.: We might need to have different kind of API's or Tools to perform a specific type of operations on different Storage System. [1] So, that why we need a third-party storage management tool to reduce the cost of management at a point where such heterogeneous storage systems can be managed as one, as if they were all created by single vendor. CoprHD which is a SDS Controller, is one of the third party storage management tool, which provides flexibility of having heterogeneous storage system environment with reduce cost of management.

Software defined storage(SDS) is a term to for a storage software which is responsible for policy based provisioning and management of data storage independent and unknown about internal hardware. SDS includes a form of storage virtualization which isolates the storage hardware from the software, which also manages storage infrastructure. This also features policy management functions such as deduplication, cloning, replication, snapshots, thin provisioning and backups.

CoprHD is an Open source SDS controller that discovers, storage pools and automates the management of a heterogeneous storage system. [2] It enables policy management and cloud automation of storage resources for block, object and file storage providers. CoprHD is a management software which creates abstraction layer over the heterogeneous storage and control them. This heterogeneous multi-vendor system can be Google Ceph, IBM storage arrays, EMC

arrays (ScaleIO, ExtremeIO). CoprHD is an open sourced project, of EMC2 Vipr Controller project. Vipr Controller will be based on the development of project CoprHD and will continue to be available as commercial product from EMC2.

The release of CoprHD as open software product is expected to get attention from 3-rd party storage vendors with a vision to integrate their arrays into CoprHD. This might need a roadmap to integrate support for 3-rd party storage drivers into CoprHD. In the earlier versions, storage driver was hard coded with CoprHD, which required comprehensive understanding of CoprHD functionalities and services. These heterogeneous storage systems might have unique set of services, capabilities and own functionalities, which may require modification to several components within CoprHD core- UI, persistent classes, export orchestrators, placement matchers. Thus understanding this model and adding new storage system on to the CoprHD would be challenging for open-source community. Therefore, the CoprHD community decided to write a South Bound SDK for Storage driver.

Primary aim of this project is to focus on implementing storage driver for the ScaleIO storage system with a new Southbound SDK. This South Bound SDK uses intermediate driver layer which interfaces storage device and CoprHD. The abstraction layer creates a generic platform to support heterogeneous storage system. In the new South Bound SDK, it does not perform Cassandra, Zookeeper, task completers from API methods and intermediate layer provides a similar scope of storage operations as current CoprHD device access layer.

To write a driver for ScaleIO, there were 5 major operation to be implemented, namely it Storage Discovery with its pool and ports, Volume Storage Services such as Create, Delete and Expand volume, and Volume Protection Services such as Snapshot, Clone and Export. So, to balance the project work, it was decided that each of the team member would be working on

individual components. Varun Rajgopal was responsible to implement Storage discovery along with it pool and ports. Taylor Cuilty was responsible to implement Volume Storage Services such as Create, Delete and Expand. Volume Protection Services were implemented by two engineers namely Shujin Wu and me, Prathamesh Patkar. Shujin Wu was responsible to implement Volume Protection Operation called as Snapshots operation and partially on Export operation. I, Prathamesh Patkar was responsible to implement Volume Protection Operation called as Snapshots operation and partially on Export operation.

The report is organized as follows: Section 2, Problem Statement explains issues faced while managing a Multi-Vendor storage system. Section 3 briefly explains about software defined storage and its characteristics. Section 4 explains CoprHD and its architecture; It also explain Southbound API, it's issues and solution for it, the New Southbound SDK. Section 5 briefly describes about ScaleIO system and its underlying features. Section 6 explain about implementation strategy and work done. Section 7 shows our Achievements. Section 9 concludes by stating contribution for this project.

2. Problem Statement: Managing a Multi-Vendor Storage System

It is always a dilemma for IT managers to find a single storage platform that meets all their needs, one that's fast, scalable, reliable and affordable, which always difficult to find. There is always different kind of systems which are featured to be suited for particular task. If one-size fits all strategy is implemented for project which is product specific, may cost an organization a competitive edge or simply not to be practical given the reality of IT budget. [1]

Addressing a specific business needs whether it be performance, capacity or cost, having a multi-vendor storage always allow to have flexibility. However, storage management becomes a

critical part of such strategy, so that the flexibility and cost saving it provides but does not get burdened in its administrative complexity. [1] IT managers need a set of best practices so that once several project-specific products are implemented they can still be effectively managed. [1] According to George Crump, Senior analyst at Storage-Switzerland there are few best practices to effectively manage storage system.

- Unify and Normalize your view: Here data is collected from all the different storage platforms and centralize into a single interface or management suite. The software provided by each storage vendor, switch vendor or host bus adapter vendor provides report of its individual devices operating condition, but does not provide a big picture of all Storage Infrastructure.
- Optimize Your Storage Assets: Fusion of storage information onto a single storage management tool has more value than simply providing reports for management. Different storage systems performing different roles, with this combined view a storage manager can see volume by volume, the activity performed by each volumes and better balance workloads.
- Unified Monitoring: Managing a mixed storage vendor individually, increases time taken to “Check-in” with the system individually and also increases risk monitoring. Sometimes it becomes a ritual to check the system in intervals, to check with problems. This increases the risk running into error which might go un noticed. Unified storage management accomplishes two goals, first the storage manager no longer spend time on logging every storage system, and second we can get errors any time or performance condition occurs.

- Operational Preparedness: Unified view of the storage system allows the storage manager to predict it near real time storage trends, which is among the most difficult task for an IT department while creating storage budget.

3. Software Defined Storage [3]:

Software Defined Storage is a term for data storage software to manage policy-based provisioning and management of heterogeneous data storage system abstracting underlying hardware. The basic concept in Software Defined Storage is to design a storage system by intelligent application of distributed computing techniques. [3] Power of distributed computing with commodity hardware, innovative storage optimization for space efficiency, performance efficiency, manageability and scalability together form a solution for cost effective management of explosive data growth: [3]

1. Elastic, Scale-Out, Software designed to run on commodity hardware
2. Multi-pool data access.
3. Built in geo replication mechanism.
4. Everything wrapped in a simplified and far more scalable management experience.

"The Net effect for an enterprise is capital and operational cost saving due to more flexible, more agile and scalable storage management experience and simpler and more modern consumption model for application teams. This is Software Defined Storage." [3]

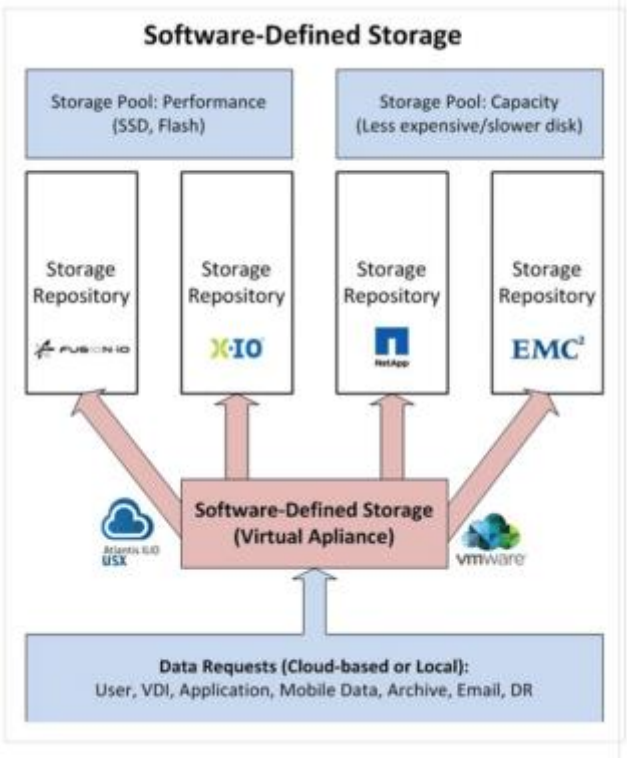

Figure 1 Software Defined Storage [4]

The above diagram is features high level architecture over view of Software Defined Storage. Storage pools can be of different types and can be used according to purpose. Storage pool can either be used to boost performance, using SSD, Flash Storage, PCIe; or it can be used to deliver high capacity requirement using less expensive disk such as HDD or Tape drives. So according to Storage pool requirement, different storage solution can be applied on the storage system whichever is best suitable with the company. To manage all this storage system together and resolve the issues managing multi- vendor storage system (as mentioned above), Software defined Storage(SDS) is implemented. SDS creates an abstraction layer over this multiple storage system, which result in simpler management, improved performance and efficient risk management of all the storage system. The request from the cloud based or local entity for the storage system is processed via SDS, which in return forward processes with an appropriate storage system.

3.1 Elasticity, Scale and Simplicity: -

Elastic Scale-out is an ability to incrementally add capacity to storage system by adding in more commodity nodes. [3] With the increase in demand, an elastic storage predictably grow to meet changing demands. Whereas in traditional storage system would simply run out space, and might need independent management.

3.2 Hardware Inclusion:

Software defined storage is designed to work on any system independent of the underlying hardware, but it does not mean SDS is a software only solution. While some major enterprises may outright buy a storage software and build their own software defined storage, but this might not be practical for other companies. To meet such wide requirement storage with legacy hardware in appliance form factor, gives all the benefits of software defined stack.

3.3 Solves Datacenter Challenge:

Software defined storage is one elements of the software defined data center, whereas the other two is software defined compute (virtualization or containerization) and software defined networking (SDN). [3] Designing a system that reliably stores your data with scale-out, multi-protocol access, geo distribution, in-place analytics capabilities, and with a simple management experience is a very hard problem. [3]

Software Defined Storage enables enterprises to create giant geo-distributed pools of storage that can be flexible with very little effort on the part of IT and its users. The economic benefits are unprecedented, as commodity-based platforms increasingly provide the foundation for next generation storage, replacing more expensive purpose-built storage arrays.

4. CoprHD:

CoprHD is an open source software defined storage controller and API platform. It enables policy-based management and cloud automation of storage resources for block, object and file storage providers. CoprHD is a management Software which creates abstraction layer over the heterogeneous storage and controls them. This Heterogeneous Storage System can be multi-Vendor for e.g. Google Ceph, IBM Arrays EMC arrays or any other arrays. CoprHD is an open sourced project of EMC2 Vipr Controller. ViPR Controller will be based on the development project CoprHD and will continue to be available as a commercial product from EMC.

4.1 CoprHD Architecture:

CoprHD is a software platform for building a storage cloud. It is designed with two key goals:

- Make an enterprise or a service provider datacenter full of storage resources (block arrays, filers, SAN and NAS switches) look and feel like a single massive virtual storage array, automating and hiding its internal structure and complexity. [5]
- Extend a full set of essential provisioning operations to end-users. This can range from simple operations such a block volume creation, to fairly complex operations such as creating disaster-recovery-protected volumes in different data centers. A key element of this capability is to enable them to use it in a truly multi-user, multi-tenant fashion by providing tenant separation, access control, auditing, usage monitoring, and other features. [5]

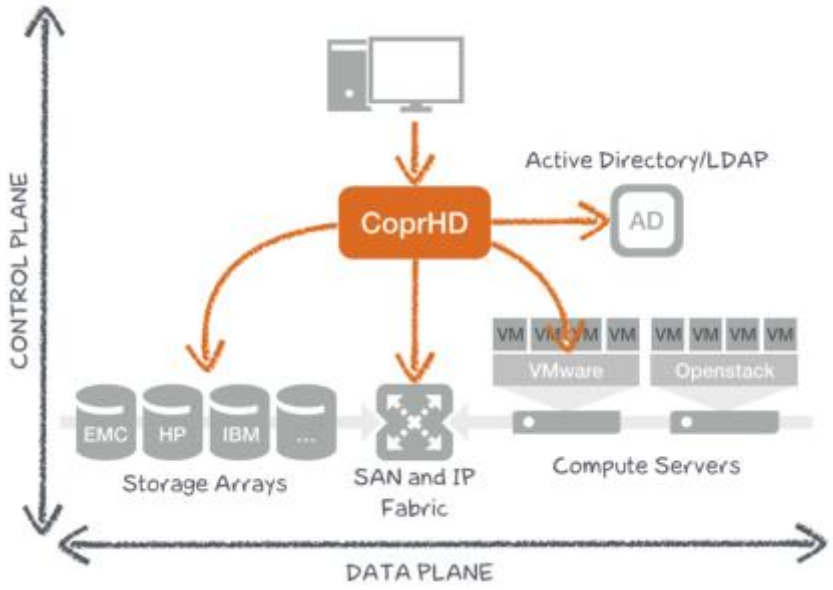

Figure 2 CoprHD Architecture [5]

CoprHD itself does not provide storage. [5] It holds an inventory of all storage devices in the data center and understands their connectivity. It allows the storage administrator to group these resources into either:

- virtual pools, with specific performance and data protection characteristics or, [5]
- virtual arrays, segmenting the data center infrastructure along lines of fault tolerance, network isolation, or tenant isolation. [5]

Storage administrators have full control over which users or tenants can access these virtual pools and virtual arrays. End-users use a single uniform and intuitive API to provision block volumes or file shares on virtual pools or virtual arrays in a fully-automated fashion. CoprHD automates all hidden infrastructural tasks, such as finding optimal placement for a volume or necessary SAN fabric configuration. Also, CoprHD adds intelligence of its own with features like the ability to apply metrics-based algorithms to port selection when exporting volumes. Finally, CoprHD provisioning goes beyond creating volumes. [5] It has the ability to perform complex

orchestrations that cover the end-to-end provisioning process - from volume creation, to SAN configuration, to mounting the newly created volume on a host. And, it has built in support for environments that include VMware and Vblock technologies. [5]

From the architectural perspective, CoprHD is an operating system for a storage cloud.

- It provides a well-structured and fairly intuitive set of “system calls” for storage provisioning in the form of REST API. [5]
- It is a multi-user, multi-tenant system. As such, it enforces access control (authentication, authorization, and separation). [5]
- The “system calls” have transactional semantics: they seem atomic from the user perspective, and when they fail on one of the operations, they either retry the failed operation or they roll back the partially-completed work.
- It is highly concurrent, allowing multiple "system calls" to execute in parallel, but internally enforcing fine-grained synchronization to ensure internal consistency. [5]
- It implements its own distributed, redundant and fault-tolerant data store which holds all of the provisioning data.
- It allows common business logic to remain device-agnostic by using a set of “drivers” capable of interacting with a bunch of storage devices. [5]

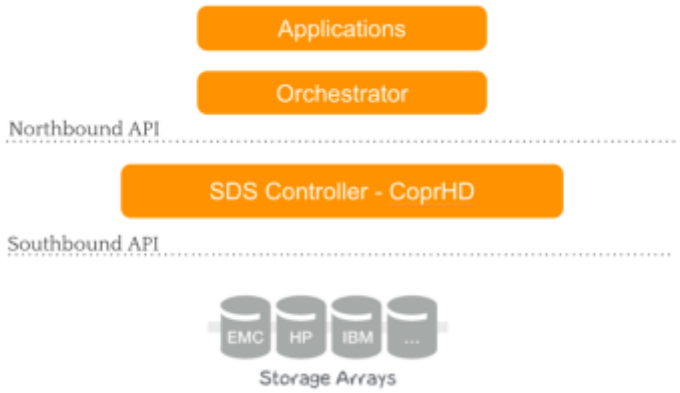

Figure 3:CoprHD High-level Architecture

CoprHD as a SDS controller lies in the control plane which lies between Northbound API and Southbound API (Figure 1). The Northbound API consist of Application and Orchestrators such as containers like Kubernetes, Mesos. So the developer has easy access to management using their favorite tool. The Southbound API consist of device driver and support for heterogeneous storage system. *My work consisted of Southbound driver, where we developed driver for ScaleIO system.*

4.2 South-Bound API:

The South bound API is mostly concentrated on device driver and support for multivendor and heterogeneous storage system. On the device driver level, the API is divided into four parts; Discover, Metering, Monitoring and Provisioning.

- *Discovery API:* Discovery API scans storage provider to get list of storage systems. It then Discover's storage system properties (serial number, IP address, firmware version, etc.), discover storage pools, discover storage ports, discover brownfield resources. It is responsible to populate all data in Cassandra/ZK which ViPR needs for provisioning operations. [6] Also it should update CoprHD task with operation status.

- *Metering API:* Metering API collects statistic information about storage resources such as: allocated/provisioned capacity of volumes and file systems, bandwithIn, bandwithOut for volumes and file systems, snapshot capacity, etc. It populates Stat time series data in Cassandra and then update CoprHD metering task with operation status. [6]
- *Provisioning API:* Provisioning API provision's storage resources on physical storage system. It populates physical properties for provisioned resources in Cassandra. Operation's such as Clone, Mirror, Export and Snapshot come under Provisioning. [6]
- *Monitoring:* Monitoring API collects events and alerts for storage arrays and populate Cassandra time series Event table. It updates ZK monitoring queue and update CoprHD task with operation status. [6]

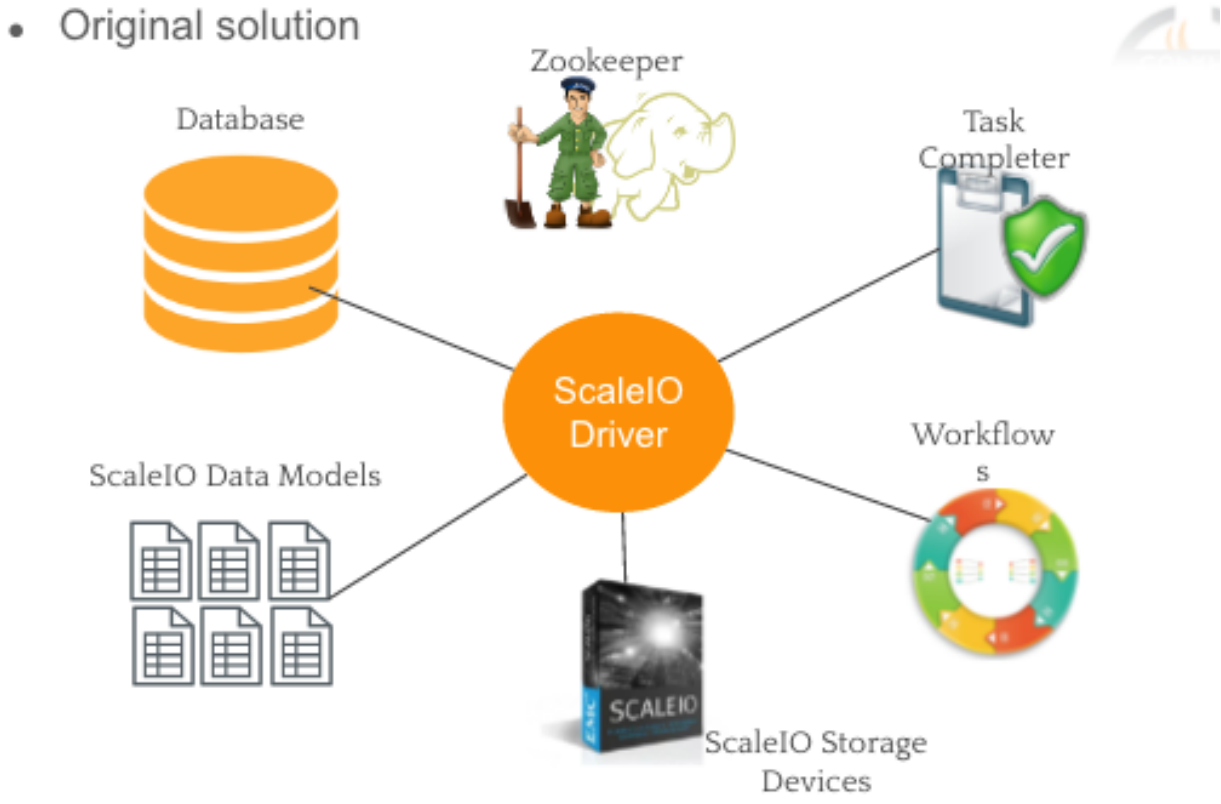

Figure 4:Existing Southbound API for COPRHD

The existing South Bound API is Tightly coupled with CoprHD infrastructure – requires driver to know how to work with CoprHD database, zookeeper, task completers, workflows. It uses persistent data model with many CoprHD housekeeping and vendor specific details. This driver model works for in house driver development, but this model is hardly suitable for third party development (Figure 4), which was challenging for open-source community. If we want to develop a driver under CoprHD it was necessary to understand the internal services of CoprHD and Data flow, which was time consuming and tedious. Also, it is difficult to understand storing and understanding registry type data in CoprHD. So, it was necessary to design a new southbound software development kit (SDK) which would make development of Southbound driver easier. [6]

4.3 Southbound SDK (Software Development KIT)

The new southbound SDK would overcome all the deficit of Southbound API and enable third party developers from open source community to freely write driver for the CoprHD without much understanding of underlying CoprHD Architecture. Figure 5 features High Level design for new South Bound SDK. For example, if ScaleIO driver want to interact with new SDK. A developer has to implement new Southbound SDK. This implemented driver would interact with Southbound driver SDK layer. This layer will in return interact with CoprHD infrastructure.

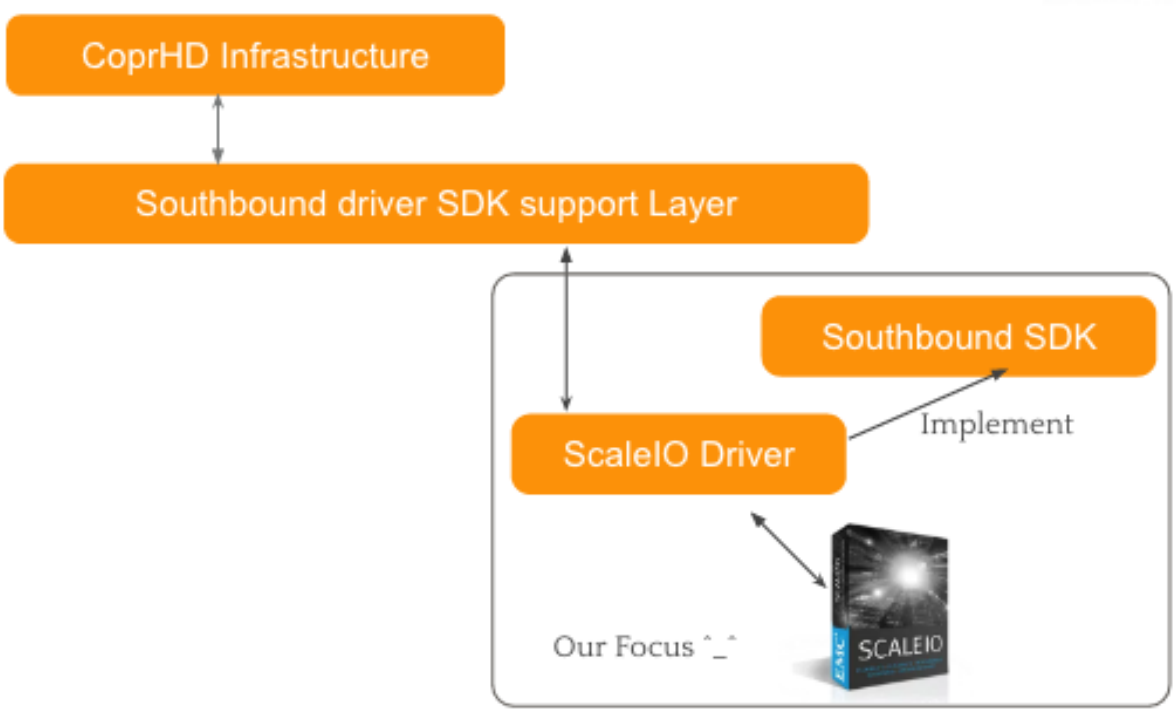

Figure 5: New South Bound SDK High Level Design

The new South bound SDK had following goals:

- Easy Plug ability for third party storage arrays.
- Should use generic object model, not bound to ViPR persistent data model classes.
- Should not have any dependency on existing infrastructure services in its implementation.
- Provide the same scope of capabilities for storage operations as current native block and file CoprHD drivers.
- Provide a way for third party drivers to advertise capability metadata and capabilities of their storage.
- Provide capability to integrate device specific orchestrations into CoprHD.
- Highly Scalable and avoid redundant technologies.

5. ScaleIO Storage System:

EMC ScaleIO is a software that creates a server based SAN from local application server storage to deliver flexible and scalable performance and capacity on demand. [7] ScaleIO converges storage and compute resources of commodity hardware into single layer architecture, combining capacity and performance, simplify management and scaling to thousands of nodes. ScaleIO combines HDDs, SSDs and PCIe flashcards to create virtual pool of block storage with varying performance tiers. [7] It provides enterprise grade data protection, multi-tenants capabilities and add on enterprise features such as QoS, thin provisioning and snapshots. [7]

5.1 Massive Scalability

ScaleIO is flexible such that it can scale from three to thousands of nodes, whereas traditional storage system, increase in number of storage devices results in increase in throughputs and IOP's. The Scalability of performance is linear with regards to growth of the deployment. [7]

5.2 Extreme Performance:

All I/O and throughput is directly accessible to any application within the cluster because every server in the ScaleIO cluster is used in the processing of I/O operations. This massive I/O parallelism eliminates bottleneck. [7]

5.3 Unparalleled flexibility:

ScaleIO features flexible deployment options, such as *"two- layer" storage only* and *hyper-converged.* The *"two-layer"* storage only is used when the application and storage are installed in separate servers in the ScaleIO cluster. This features efficient parallelism and no single point of

failure.

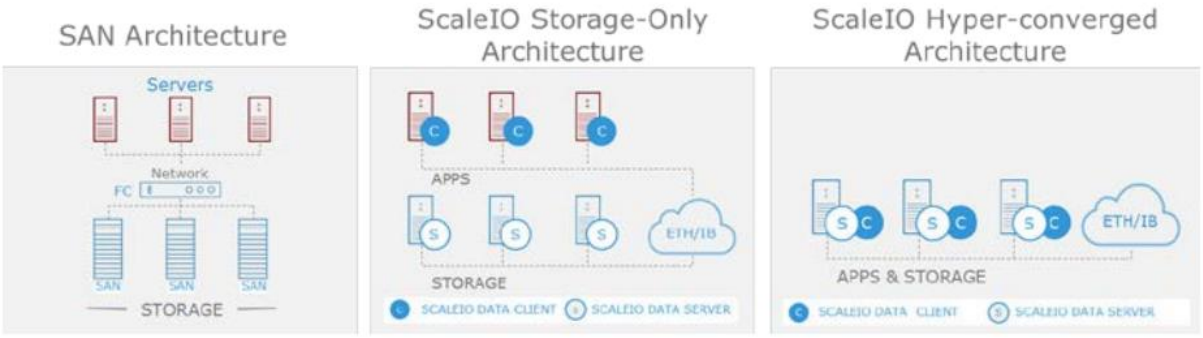

Figure 6 Unparalleled Flexibility in ScaleIO [3]

The *"hyper-converged"* option is sued when the application and storage are installed on the same servers in ScaleIO cluster. This results in single layer architecture and features the lowest footprint and cost profile. [7]

5.4 Supreme Elasticity:

In ScaleIO system, compute and storage resources can be increased or decreased whenever the needed. The system is designed to automate rebalancing date on the fly with no downtime. Addition and removal of resources can be done in large increments. No capacity planning or complex reconfiguration required. [7]

6 Implementation

6.1 Background Study:

As a fresh software developer, it took us some time to understand IT infrastructure and storage system. Understanding the overall architecture of this IT infrastructure, storage system and virtualization technology was itself time consuming due to high level of complexity and multiple domains involved. Considerable amount of time was taken to understand relation between CoprHD in IT infrastructure world and management of heterogeneous storage.

Vipr Controller was open sourced as CoprHD in June 2015, so the CoprHD community was completely new, with very less support for the open-source development. The Code base of the project was very huge, with large set of redundant and complex code tightly coupled together. Understanding the code flow was itself a challenge to new developers. Since the environment and code was developed in EMC^2, it was a Challenge to deploy it in Oregon State University, with provided documentation.

The Southbound SDK and it support layer was partially developed which slowed down the progress of understanding relationship and dependency of SDK with CoprHD. Also slowly it was improvising in terms of document. Concurrent development of ScaleIO driver with the SDK helped the community to make it stable and reliable. The aim of ScaleIO driver Development are as follows:

- Test functionalities of SDK and SDK support layer.
- Improve SDK design with driver development.
- Exemplary driver development process.
- Testing Drivers.

6.2 Development Environment:

Setting up Developer Environment was time consuming and complicated process due to two reason:1) CoprHD was newly released and had very less documentation 2) Developers Environment Instruction were strictly made for Developers at EMC2. Developer Environment was setup in stages after understanding the requirement and scope of the project.

1. **IDE Setup:** After defining scope of the project, it was really important to use an IDE which is user friendly, can work with GitHub, provide advance tools to check the code formatting,

and misc. functionalities which would help in code development. Intellij was preferred over Eclipse because of better functionality and UI.

2. **CoprHD:** To understand functionality and behavior, and test the code on CoprHD, setting it up was really important. CoprHD at OSU was deployed on OpenStack framework, with each engineer having their own OpenStack and CoprHD node to accelerate development and testing process. To Host CoprHD minimum requirement were 60Gb of root filesystem and 6gb of ram. The instruction provided by EMC2 were only available for Vcenter Deployment, whereas Team OSU was able to host it on OpenStack by their own efforts.
3. **ScaleIO System:** ScaleIO system was hosted on VMware, with 3 nodes of each 100gb and one management node. It was hosted molding Vagrant script AIO provided by Engineers at Intel Corp, who were the contributor for this project. Thus storage pools created by ScaleIO was made available to CoprHD by assigning appropriate IP addresses.

To test the code, IP address from ScaleIO system were directly connected to Unit test file, whereas result was checked on ScaleIO system each time REST API call was made, to verify the changes. The code was deployed on CoprHD System, using command line "Deploy" which uploaded the part of storage driver on the recent CoprHD setup.

Development process was made easier using Industry standards tools. Tool's such as Jira was used to track the project status, Confluence to get any suggestions/update project status, document project development, Jenkin to deploy and check the code, and Hip-Chat to interact with community members.

6.3 Driver Development:

ScaleIO driver development was straightforward, and easier way to communicate with storage system using SDK then previous API development. Architecture was pretty simple to understand, just send request and process the result from the storage system and update CoprHD for the operations performed. The ScaleIO Storage Driver did not have any dependency on the existing code, it just calls the REST API call and perform operations on it. Figure 4 and Figure 5 provides a big picture of comparison between existing South Bound API and Southbound SDK. While Figure 7 explains interaction between ScaleIO REST API, Storage driver interface and CoprHD.

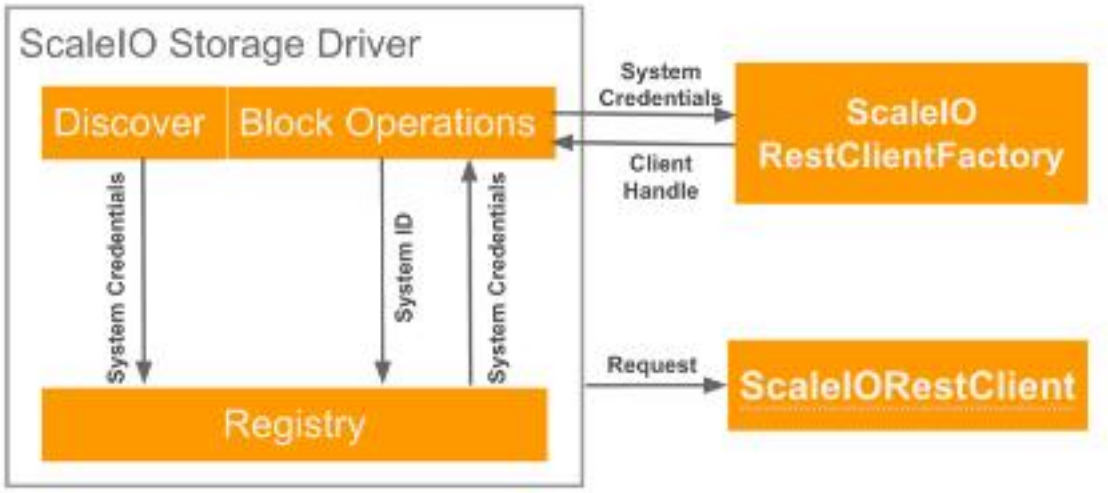

Figure 7:ScaleIO communication in new Southbound SDK

Work was split according to different supported functionalities such as Discover, Volume Provisioning, Clone, Snapshots and Export operations. My part of work was to write code for Clone operation, and partial process of Export operation.

High level functional overview and design:

- Implement storage driver for ScaleIO storage array based on Southbound SDK.

- Implement and test ScaleIO device driver based on ScaleIO storage driver. (Existing implementation)
- Usage of ScaleIO Rest API as access points to ScaleIO array.

6.3.1 Clone Operation and Export Operation

- **Clone Operation:**

  A clone is copy of an existing volume. Here, the existing volume is the parent of the clone. When the cloning operation is done, the clone is entity thought they might share the same virtual disk, with parent clone. After cloning, Changes made to parent volume does not affect the new clones, and vice versa. A clone's MAC id and UUID is different from parent volume. Also, a group of volumes performing similar or same job can be cloned into another group of clone volumes in a storage array, which is called as consistency group clone. A Consistency group is a collection of base volumes in your storage array. These base volumes, which are the source of snapshot image, are referred to as member of volumes of a consistency group.

  Clones are useful when there is a need to deploy multiples volumes or virtual machines and to have a backup in case of failure. There are two types of clones; Full Clones and Linked Clone. The Full clone is an independent copy of a volume that shares nothing with the parent virtual machine after the clone operation. Ongoing operation of a full clone is entirely separated from parent volume. A linked clone is a copy of a virtual machine that shares virtual disk with the parent machine in ongoing manner. This conserves disk space and allows multiple virtual machine to use same software utilization. A linked clone is made from snapshot of parents.

  - Create Clone: It creates a clone/ copy of the entire parent volumes.

- Detach Clone: It detaches the cloned volume from the parent volume
- Delete Clone: It deletes the clone volume for the parent volume.
- Restore Clone: It would clone a backup to the parent volume.
- Create Consistency Group Clone: Create a group of clone volume from a group of parent volumes
- Delete Consistency Group Clone: Delete a group clone.
- Detach Consistency Group Clone: Detach a group of clone from the parent volume.

- **Export Operation:** Export Operation performs different types of authority provisioning operation between a list of initiators and set of volume or virtual machines. Initiators are set of client devices. It assign's authority to initiators in need of volume resources with proper authorization.
  - GetITL: It is block export operation. Here, it gets export mask for a given set of initiators. Initiators are client devices. It exports objects (can be multiple volumes) to the given set of initiators
  - Export Volumes to Initiators: Volumes are assigned and exported to initiators through given set of port.
  - Unexport Volumes From Initiators: Unexport set of volumes from set of initiators

6.3.2 Implementation:

The implementation was primarily based on Test Driven Development. In this development, the development team was responsible to write test cases for each of the Operation and implement them, and then check with the ScaleIO system. The ScaleIO system connectivity was tested using assigning IP address of our ScaleIO system in the test cases. The test cases were designed to check

if the Create/Delete/Detached/Restore, if it is empty, if it has partially done the operation, or failed to perform operation. With the test cases completed, team proceeded to write drivers. It was necessary to understand REST API for the ScaleIO system, to understand interaction between REST API and CoprHD. Not all the function mentioned below makes REST API call, some of them performs update operation to CoprHD. Clone operation in ScaleIO is treated as snapshot operation, so the snapshot REST API call is made. There is some task which are supported by the ScaleIO, while some are not.

- Create Clone: It is treated as snapshot, and REST API call is made to snapshot the volume, along with updating all the supporting functions for CoprHD
- Detach Clone: No REST API call is made in this function and just replication status is set to detached in CoprHD, because in ScaleIO a volume is immediately detached after its creation.
- Delete Clone: A REST API call is made to remove the volume. It is just treated as simple volume, and remove operation is performed.
- Restore Clone: Restore Operation is not supported in ScaleIO.
- Create Consistency Group Clone: It is treated as group snapshot operation, whereas Snapshot multi volume API call is made, along with updating few parameters for CoprHD
- Delete Consistency Group Clone: A REST API call is made to remove the set of volumes. It is just treated as simple volume, and remove operation is performed along with updating parameters for CoprHD.
- Detach Consistency Group Clone: No REST API call is made in this function and just replication status is set to detached in CoprHD, because in ScaleIO a consistency group clones are immediately detached after its creation.

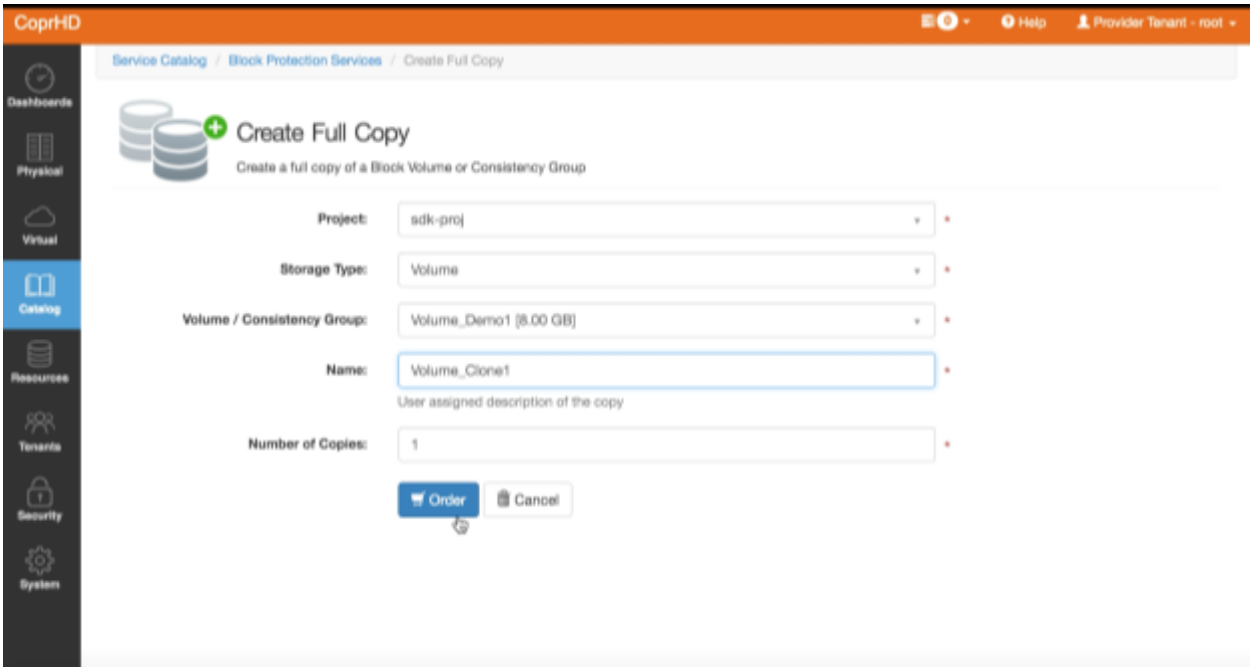

Figure 8: Create Clones

Figure 8 shows create clone functionality; it comes under volume protection services in CoprHD. Select the type of project, type of storage which may vary according to need. Here type of project is the new ScaleIO using Southbound SDK. Then select either the volume to be clone or consistency group to be cloned. Next specify the name of the newly cloned volume and number of clones needed. Finally press order to clone the parent volume.

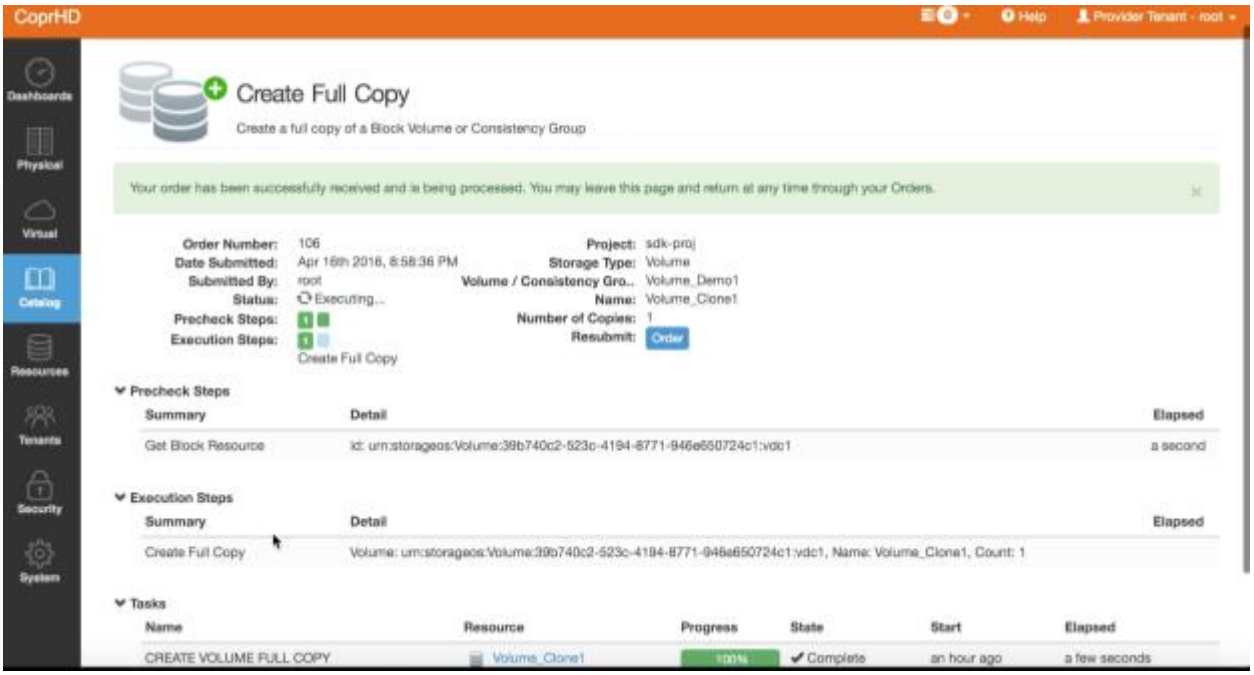

Figure 9: Create Clone Result

Figure 9 shows outcome of Create clones. It shows order status and other order detail. The final outcome is Successful, Partially Failed or Failed. Summary shows the Volume number and volume to be cloned.

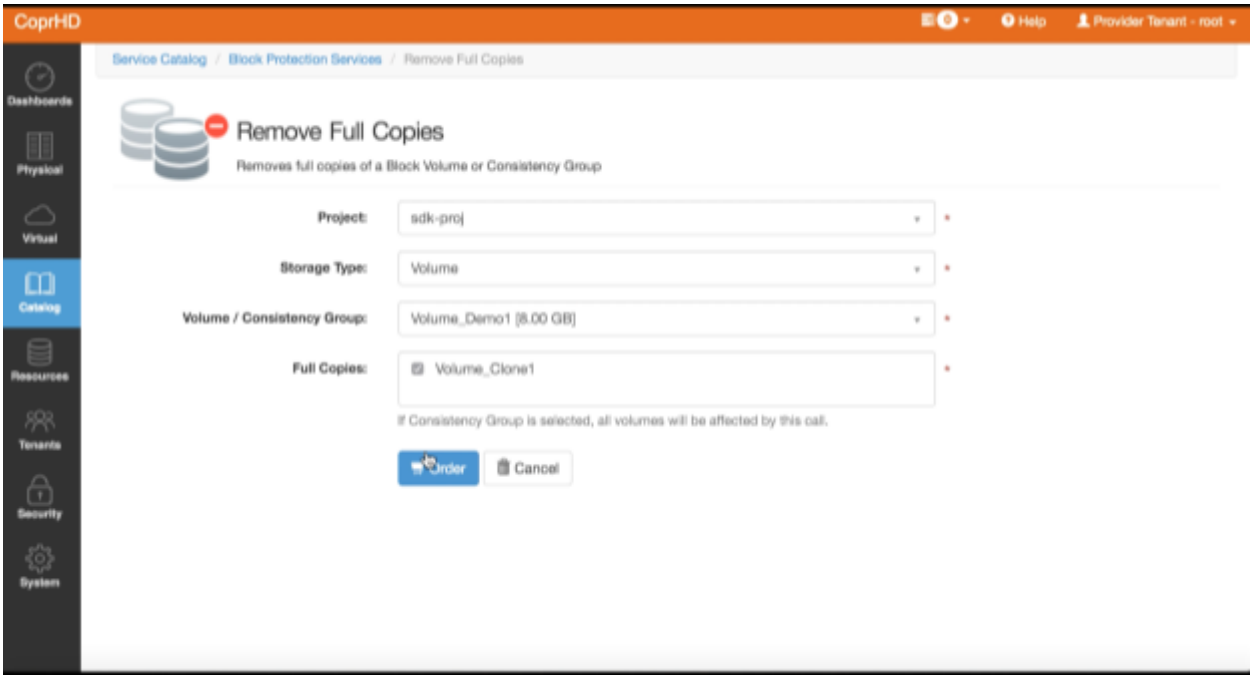

Figure 10: Delete Clone

Figure 10 shows Delete Clone Operation. Like Volume Create operation, in Volume Deletion, select the consistency group or volume and the name of the clone/clones to be deleted.

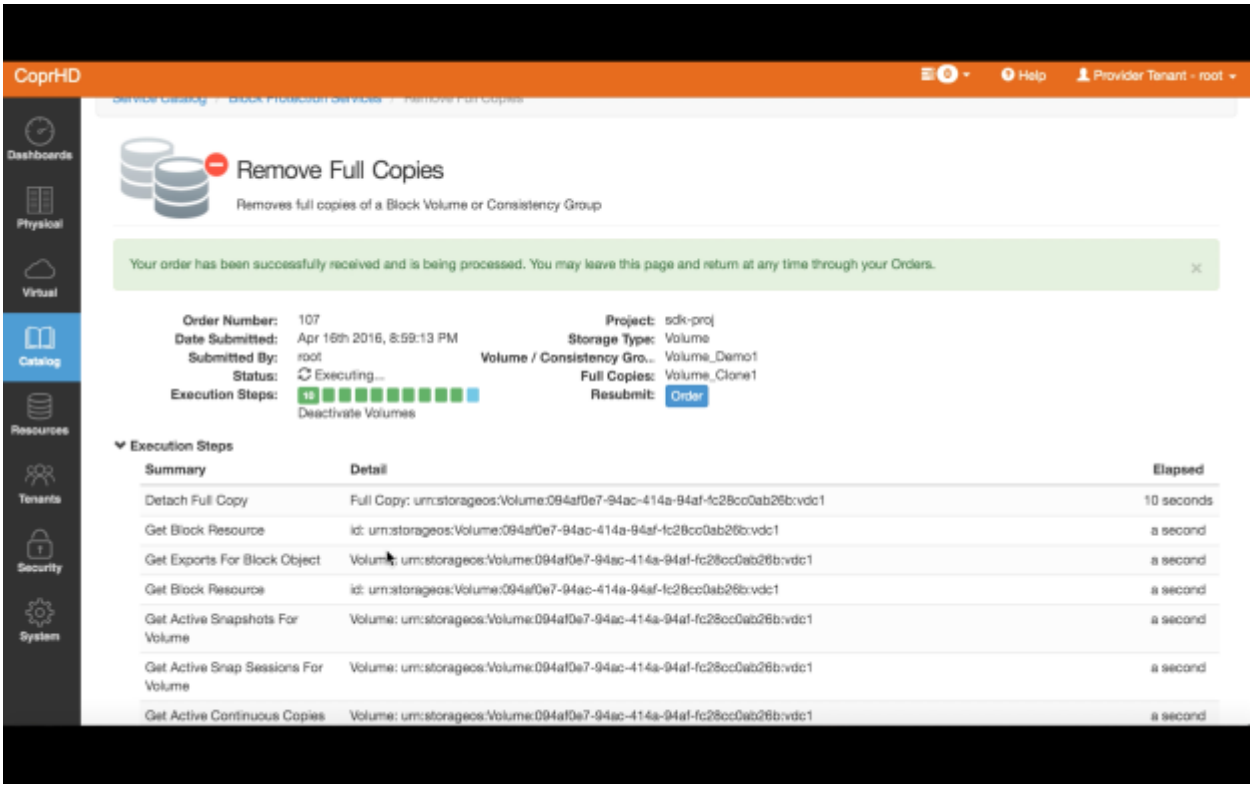

Figure 11: Delete Clone Result

Figure 11 show result of Delete Clone Operation, it shows order number and the details of process. It also shows execution process for removal of clones. Outcome can be either the volume is deleted or partially deleted or failed.

6.3.3 Task Completed(Overview):

- Planning and Designing ScaleIO driver for South-Bound SDK for Clone and Export Functions
- Setting up personalized developers environment for OSU and ScaleIO cluster. Setting up involved IDE setup, CoprHD Setup, and ScaleIO cluster setup.

- Implementing Volume protection function Clone, and task such as Create Clone, Detach Clone, Delete Clone, Create Consistency Group Clone, Delete Consistency Group Clone.
- Performing Unit testing on the ScaleIO driver for Clone

7. Achievements:

CoprHD South bound SDK development for ScaleIO has become a template to write heterogeneous driver for CoprHD for the community. Since the development of ScaleIO driver was concurrent with SDK development, which resulted in stable SDK, due to frequent feedbacks provided by Oregon State University team. We also contributed towards providing various documentation to setup developers environment, process to write a driver, process to perform Unit and integration testing.

Also OSU team got an opportunity to present their part of work in two conferences.

CoprHD Summit: 2nd Developers meetup held at Oregon State University, January 2016. This conference provided an opportunity for OSU team host a large open source community, share new technology and present and review the project.

Vault Conference: "Vault" is a storage conference organized in April 2016, organized by The RedHat Inc. Team OSU presented their fully featured project, along with that represented CoprHD community by sharing their experience and promoting CoprHD.

8. Conclusion:

The project is comprehensive process of related study for Software Defined storage, CoprHD, defining development environment, design, implement and test for volume protection operation clone. This project development provided a great insight and experience in Software Defined Storage(SDS), Software Defined Network, IT Infrastructure and Datacenter operation,

Virtualization technology such as VMware and OpenStack, with exposure to Industry Standard tools and ethics.

The project implemented a ScaleIO storage driver for the volume protection operation Clone with its sub-functionalities using the new CoprHD Southbound SDK. The result of the project was API that is not bound to CoprHD persistent data model classes and does not depend on the CoprHD infrastructure services in its implementation. In the new South Bound SDK, third party storage drivers can easily integrate to CoprHD to advertise capability metadata and capability instances of their storage resources and can also provide capability to integrate device specific orchestrations into CoprHD. This enables developer to write a driver for a third party storage, without understanding underlying layers and relation between Cassandra, zookeeper, task completers from API methods.

This is a big advantage in an Open-Source community, whereas not only development process is accelerated but also encourages the developers to write driver without much efforts. Concurrent development of Driver and SDK and feedback from OSU helped the community to develop a stable and reliable SDK. Finally, driver developed by OSU became a template to develop drivers for future storage system in CoprHD open-source community.

Bibliography

[1] George Crump, Senior Analyst, Storage-Switzerland, Thursday, February 23, 2012 “Best Practices For Managing A Multi-Vendor Storage Environment”, http://www.storage-switzerland.com/Articles/Entries/2012/2/23_Best_Practices_For_Managing_A_Multi-Vendor_Storage_Environment.html

[2] Anjaneya Chagam, Urayoan Irizarry, U. “Introduction to CoprHD: An Open Source Software Defined Storage Controller”. Online: http://www.snia.org/sites/default/files/SDC15_presentations/sds/ChagamIntroduction_CoprHD_SDSController-r4-revision.pdf

[3] Salvatore DeSimone, Vice President & Chief Technology Officer, Emerging Technologies, “Defining Software-Defined Storage” Online: http://www.emc.com/microsites/sds/articles/software-defined-storage-definition.htm

[4] Bill Kleyman, March 5,2014, “Software Defined Storage: What does it really means?” Online: http://www.datacenterknowledge.com/archives/2014/03/05/software-defined-storage-really-mean/

[5] Olaf Manczak, Bill Elliot, Irrizary, “A Short Guide to CoprHD Architecture”, Online: https://coprhd.atlassian.net/wiki/display/COP/A+Short+Guide+to+the+CoprHD+Architecture

[6] Evgeny Roytman, “Integration of 3-rd party drivers to CoprHD”, First CoprHD Summit at Boston, MA , July 2015

[7] EMC2 Corp, “EMC SCALEIO: Software Defined, Scale-Out SAN”, Online: https://www.emc.com/collateral/data-sheet/h12713-emc-scaleio.pdf