ScaleIO Storage Driver Based on CoprHD Southbound SDK
AN ABSTRACT OF THE PROJECT OF

Varun Rajgopal, for the degree of Master of Science in Computer Science presented on August 11, 2016.

Title: ScaleIO storage driver based on CoprHD Southbound SDK

Abstract approved:

______________________________________________________

Dr. Carlos Jensen

CoprHD is an open source software defined storage controller platform. It holds an inventory of all storage devices in the data center and understands their connectivity. It is an operating system for a storage cloud. It is designed with two key goals in mind [1]:

- 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.
- Extend a full set of essential provisioning operations to end-users.

CoprHD automates storage provisioning and orchestrates repetitive and complex tasks. It is open and extensible providing REST API support, Java, Ruby and Python SDKs. It supports varieties of platforms such as VMAX, SRDF, XtremIO, VNX,
RecoverPoint, Isilon, ScaleIO, VPLEX, NetApp, HDS, and IBM XIV. It easily plugs into any of the cloud stacks such as VMware, Microsoft and OpenStack [1].

CoprHD integrates with the Northbound API and the Southbound API. It plays a “service provider” role with a third party software system to do the storage provisioning operations. The target system is in the north of CoprHD. It plays a “Storage Manager” role with a storage device to do discovery/provisioning/metering/monitoring related operations. The target system is in the South of CoprHD.

The limitations of current southbound API approach work for internal device access layer, but is hardly suitable for third party device drivers. In this model, the storage driver is tightly coupled with the CoprHD infrastructure and requires the storage driver to know how to work with the CoprHD database, zookeeper, task completers, and workflow. It does not support device independent capability model.

The goal of this project is to implement a ScaleIO storage driver for the discovery of the ScaleIO storage device, its pools and ports using the new CoprHD Southbound API that is not bound to CoprHD persistent data model classes and does not depend on the CoprHD infrastructure services in its implementation.
ScaleIO Storage Driver Based on CoprHD Southbound SDK

by
Varun Rajgopal

A PROJECT

submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented August 11, 2016
Commencement June 2016
Master of Science project of Varun Rajgopal presented on August 11, 2016

APPROVED:

__________________________________________
Major Professor, representing Electrical and Computer Engineering

__________________________________________
Director of the School of Electrical Engineering and Computer Science

__________________________________________
Dean of the Graduate School

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

__________________________________________
Varun Rajgopal, Author
ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Carlos Jensen for his constant support and guidance. Working with him has helped me learn a lot about how to approach and solve problems. His course on HCI was very helpful, in analyzing and understanding the important aspects between people and computers.

I'd like to thank my committee member, Dr. Bella Bose, whose course on Error Correction Codes was very interesting. His course helped me to learn various error correct techniques that enable to reconstruct the original data when the data is corrupted. I'd like to thank my committee member, Dr. Rakesh Bobba for his constant support and guidance.

I'd like to thank Shayne Huddleston for giving me an opportunity to work on this project. I'd like to thank Brenda Langley, Nicole Thompson, Tina Batten, and the members of the EECS office who were very helpful to me during my Master’s program. I’d also like to thank Jason Davidson, John Mark Walker, and Evgeny Roytman from EMC Corporation and Anjaneya Chagam from Intel for their constant support and guidance.

Finally, I'd like to thank my parents, my family, my friends for their love and support.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Software Defined Storage</td>
<td>4</td>
</tr>
<tr>
<td>3 CoprHD</td>
<td>5</td>
</tr>
<tr>
<td>3.1 CoprHD cluster</td>
<td>7</td>
</tr>
<tr>
<td>3.2 CoprHD Functional Overview</td>
<td>9</td>
</tr>
<tr>
<td>3.3 CoprHD Virtual Data Center Resources</td>
<td>11</td>
</tr>
<tr>
<td>4 CoprHD Southbound SDK</td>
<td>15</td>
</tr>
<tr>
<td>4.1 Interface implementation</td>
<td>15</td>
</tr>
<tr>
<td>4.2 Device discovery</td>
<td>16</td>
</tr>
<tr>
<td>5 EMC ScaleIO storage device</td>
<td>18</td>
</tr>
<tr>
<td>5.1 Software components</td>
<td>18</td>
</tr>
<tr>
<td>5.2 Protection domains</td>
<td>20</td>
</tr>
<tr>
<td>6 Design and Approach</td>
<td>21</td>
</tr>
<tr>
<td>6.1 High level template for driver API</td>
<td>22</td>
</tr>
<tr>
<td>6.2 Workflow</td>
<td>23</td>
</tr>
<tr>
<td>6.3 Implementation</td>
<td>26</td>
</tr>
<tr>
<td>7 Conclusion</td>
<td>31</td>
</tr>
<tr>
<td>Bibliography</td>
<td>32</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. ScaleIO components map to CoprHD Components</td>
<td>20</td>
</tr>
<tr>
<td>3. High level template for driver API</td>
<td>23</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Top 10 challenges of data center operators [2]</td>
<td>1</td>
</tr>
<tr>
<td>2 SDS architecture [3]</td>
<td>4</td>
</tr>
<tr>
<td>3 CoprHD architecture [3]</td>
<td>5</td>
</tr>
<tr>
<td>4 CoprHD cluster [3]</td>
<td>8</td>
</tr>
<tr>
<td>6 CoprHD virtual data center resources [4]</td>
<td>11</td>
</tr>
<tr>
<td>7 Interface implementation with an example of ScaleIO storage driver</td>
<td>16</td>
</tr>
<tr>
<td>8 High level implementation of the design</td>
<td>21</td>
</tr>
<tr>
<td>9 Code flow</td>
<td>23</td>
</tr>
<tr>
<td>10 ScaleIO Storage System information in the CoprHD after discovery</td>
<td>28</td>
</tr>
<tr>
<td>11 New Storage Pools in CoprHD UI after ScaleIO Storage System discovery</td>
<td>29</td>
</tr>
<tr>
<td>12 New Storage Ports in CoprHD UI after ScaleIO Storage System discovery</td>
<td>30</td>
</tr>
</tbody>
</table>
1 Introduction

As enterprises consolidate into mega data centers and small-to-medium sized businesses move to cloud and hosting infrastructures, data centers are rapidly expanding to thousands of servers. A recent survey revealed that top 10 concerns and challenges of data center operators [2].

![Top 10 challenges of data center operators](image)

Figure 1. Top 10 challenges of data center operators [2]

Management, capacity planning, power availability, and costs are among the main challenges that operators of modern data centers face. Data storage is the origin of many of these challenges.

Using the software defined storage (SDS) approach to data storage we will be able to use programming techniques that control storage-related tasks that are decoupled from the physical storage hardware. SDS places the emphasis on storage-related services rather than the storage hardware. The goal of storage
defined storage is to provide administrators with flexible storage management capabilities through programming. A storage system can be used more efficiently and its administration can be simplified through automated policy based management. With the software-defined market evolving and many storage marketers are still trying to figure out how to associate their products in order to be considered as part of the SDS market.

EMC VIPR controller is one of the industry’s top “software-defined storage” controller in the market. It can control and manage variety of storage platforms, and it is multi-vendor, supporting not only EMC, but NetApp, Hitachi and many others. VIPR Controller has REST APIs, ability to integrate to OpenStack Cinder APIs, a pluggable backend, and is the only software stack that fulfills the hopes and dreams of a true SDS controller by not only providing heterogeneous storage management but also metering, a storage service catalog, resource pooling and more. CoprHD is the open source version of the VIPR controller.

With the release of CoprHD as open software product is expected to attract attention from 3-rd party storage vendors. It is anticipated that vendors will start integrating their arrays into CoprHD. This requires to provide clear way to integrate support for 3-rd party storage array drivers into CoprHD. In the past integration of new type of storage array were primarily statically coded into CoprHD. Typically, new array types bring their own flavor of data services, unique capabilities, their own device export constructs. This required additions and modifications to several software components within the CoprHD core — UI,
persistent classes, export orchestrators, placement matchers. In this model, addition of new arrays to CoprHD supported set of storage products is challenging for open source community. This is high-level description of limitations of current device access layer in CoprHD for open source storage array integration.

Therefore, this project focuses on implementing a storage driver for the ScaleIO storage device for the new Southbound SDK using a driver intermediate layer that acts as an interface between the storage device and CoprHD. The abstraction layer creates a generic platform to support different types of storage devices. In this design the storage driver does not have direct access to Cassandra, Zookeeper, task completers from API methods and intermediate layer provides a similar scope of storage operations as current CoprHD device access layer.

This document is organized as follows: Section 2 provides an overview of the software-defined storage architecture; Section 3 gives an overview of the CoprHD architecture, cluster, services, its functional overview and resources. Section 4 details about the current Southbound SDK that we are using. Section 5 discusses about the EMC ScaleIO storage device for which we are going to implement a storage driver. Section 6 discusses the design and approach to implement a ScaleIO storage driver using a new Southbound SDK. Section 7 concludes by stating the contributions of this project.
2 Software Defined Storage

Software defined storage (SDS) brings “cloud” benefits to storage, including auto-provisioning, self-service, and single pane of glass for management. A key enabler of the SDS architecture is an SDS controller for a single pane of management [3]. The SDS controller provides visibility and control of all storage resources in the data center and acts as a communication interface between the applications, orchestrator, and storage systems. It also allocates storage resources to meet SLAs.

Figure 2. SDS architecture [3]
3 CoprHD

CoprHD is an “open source” SDS controller that discovers, pools and automates the management of a heterogeneous storage ecosystem [3]. Following are some of the benefits:

- It integrates with traditional, cloud, cloud native computing stacks.
- Self-service provisioning via REST APIs and catalogs.
- It provides end to end storage automation. It includes intelligent resource selection and placement, local and remote protection, SAN zoning, host attach, migration and tech refresh.
- Discovers heterogeneous storage systems. It may include traditional, scale-out, SAN/IP networking, host config, across one or more DCs.

![CoprHD Architecture](image)

Figure 3. CoprHD architecture [1]

CoprHD addresses these goals in a vendor-agnostic and extensible way,
supporting a variety of storage devices from several vendors, without resorting to a "least common denominator" approach in regard to features offered by these devices. It itself does not provide storage. It holds an inventory of all storage devices in the data center and understands their connectivity [1]. It allows the storage administrator to group these resources into either:

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

End-users use this as 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, it 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. 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 [1].

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.

- It is a multi-user, multi-tenant system. As such, it enforces access control (authentication, authorization, and separation).
- 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.
- 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.

### 3.1 CoprHD Cluster

To provide high availability (HA), CoprHD is a cluster comprising 3 or 5 identical nodes, operating in an active-active mode. CoprHD itself is a cloud application, and the nodes do not share any resources and do not rely on any hardware support to implement HA [3]. In fact, in a typical deployment, the nodes are simply virtual machines.
Each node runs the same collection of services. In general, the CoprHD software is divided between services along functional boundaries. Here is the list of services:

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbsvc</td>
<td>Comprises a distributed column-based database used as a persistence layer for all provisioning data (Cassandra backed)</td>
</tr>
<tr>
<td>coordinatorsvc</td>
<td>Offers distributed shared memory used for cluster coordination.</td>
</tr>
<tr>
<td>controllersvc</td>
<td>Performs asynchronous operations on storage devices and comprises all of the device specific code.</td>
</tr>
<tr>
<td>apisvc</td>
<td>Provides all public APIs for storage provisioning and storage management</td>
</tr>
<tr>
<td>authsvc</td>
<td>Provides the authentication APIs</td>
</tr>
<tr>
<td>portalsvc</td>
<td>Implements the CoprHD web-based GUI</td>
</tr>
<tr>
<td>sasvc</td>
<td>Offers automation services on top of the provisioning APIs</td>
</tr>
</tbody>
</table>
syssvc | Provides various system management and monitoring interfaces, access to internal logs, etc.

Table 1. CoprHD services [3]

### 3.2 CoprHD Functional Overview

The CoprHD Storage Controller provides abstraction of physical data centers to a Virtual Data Center.

![CoprHD functional overview](image)

Figure 5. CoprHD functional overview [4]

*Virtual Data Center* is a CoprHD abstraction that manages a virtually defined data center, consisting of resources from one or more physical data centers. Each Virtual Data Center is managed by one CoprHD cluster (set of nodes) [4].
**Virtual Storage Array** is an abstraction consisting of resources from one or more Storage Systems (ports and physical pools), SAN Networks, and Host Resources (host initiators).

**Virtual Storage Pool** represents a pool of storage with particular characteristics that can be provisioned in order to implement a given *Class of Service* [4]. A Virtual Pool can be considered a type of filter; selecting no attributes allows a virtual pool to contain all storage pools available in the system. As attributes are added, they filter the storage pools to those matching those attributes. Virtual Pools can be configured to have Disaster Recovery (DR) or High Availability (HA) between Virtual Arrays using product technologies like RecoverPoint, VMAX SRDF.
3.3 CoprHD Virtual Data Center Resources

The white boxes are physical resources that are contained in the VDC. These include *Storage Systems*, which are comprised of *Storage Ports* and *Storage Pools*, *Networks*, and *Hosts* [4]. In many cases all these physical resources can be discovered by CoprHD. The discovery and configuration of the physical VDC Resources is generally the responsibility of the System Admin.

A *Tenant* is the entity that is authenticated and authorized to do certain actions in the VDC. Users belong to tenants. Users may also be given various tenant-specific roles, such as Tenant Administrator, Project Administrator, or Tenant Approver [4]. The roles determine what actions a User can take for that tenant. Users may also have VDC-wide roles such as System Administrator,
Security Administrator, etc.

One or more Projects may be contained within a Tenant. A project holds the various Storage Resources that are created, such as Volumes, Snapshots, File Systems, etc.

Virtual Arrays contain resources from one or more Storage Systems. They are configured by a System Admin. The relationship between the Virtual Array and Storage System components is complicated, and depends somewhat on how the Virtual Array is configured:

Storage Systems are associated with Virtual Arrays indirectly by the association of Storage Ports or Storage Pools within that Storage System being associated with the Virtual Array [4].

Storage Ports are associated with the Virtual Array either implicitly (by being in a Network that is explicitly associated with the Virtual Array) or explicitly (by being manually assigned to the Virtual Array), A given Storage Port is either implicitly or explicitly assigned to all the Virtual Arrays it is associated with. It cannot be implicitly assigned to some Virtual Arrays while being explicitly assigned to others. If the System Administrators take no action to explicitly assign a Storage Port to a Virtual Array, it will automatically be implicitly assigned to all the Virtual Arrays that the Network containing the Storage Port is assigned to. However once the Storage Port is explicitly assigned to a Virtual Array, any implicit assignments it had are destroyed.
Storage Pools are similarly implicitly or explicitly assigned to Virtual Arrays. They receive an implicit assignment to the Virtual Array if any Storage Port on the Storage System is assigned to the Virtual Array. Alternately, Storage Pools may be explicitly assigned to Virtual Arrays in which case their implicit assignments are destroyed. Storage Pools are also implicitly or explicitly assigned to Virtual Pools [4]. They are implicitly assigned if their characteristics match all the relevant characteristics in the Virtual Pool. The Virtual Array may specify the explicit assignment of Storage Pools, in which case the Storage Pool characteristics must still match the Virtual Pool characteristics, and additionally, the Storage Pool must be explicitly identified in the Virtual Pools list of assigned Storage Pools.

- Virtual Pools contain match criteria for Storage Systems and Storage Pools that represent the characteristics Storage Resources (Volumes, File Systems) will have if they are created using the Virtual Pool. Examples of the various match criteria are given below. Virtual Pools do not contain Storage Pools, but are implicitly matched using the criteria or explicitly matched using both the criteria and a list of selected Storage Pools.

Storage Resources are provisioned using a Virtual Array and a Virtual Pool so as to provide storage space of the desired criteria (given by the Virtual Pool) and location (given by the Virtual Array). Each Storage Resource also belongs to a Project that contains the various Storage Resources associated with a project or application. Storage Resources may be of various types (such as Block, File, or Object). After a Virtual Pool change request is successfully processed, the Storage Resource will then be associated with a new Virtual Pool and the association with
the original Virtual Pool is severed.

*Hosts* use Storage Resources to store data for some application. A Host may be part of a *Cluster* but is not required to be.

*Export Groups* are a CoprHD structure that provides access for one or more Hosts to one or more Block Storage Resources.

* SMBFileShares and FileExportRules* are structures that provide IP access from one or more Hosts to one or more File Storage Resources.
4 CoprHD Southbound SDK

CoprHD plays a “Storage Manager” role with a storage device to do discovery/provisioning/metering/monitoring related operations. The target system is in the South of CoprHD.

CoprHD has software layer which encapsulates access to physical devices through their APIs. This layer provides internal interfaces to CoprHD core for discovery, provisioning, metering and monitoring.

4.1 Interface Implementation

Implementation of device layer API is tightly bound to Cassandra and Zookeeper, and in some cases needs access to instances of core CoprHD services for internal communication. Typically, implementation of these interfaces uses parameters, which reference persistent classes, to read necessary data from CoprHD database, builds request to physical device, executes this request and calls task completer to update database with physical properties of provisioned devices and to call CoprHD core services to notify about task completion.

The major issue with this approach is that interfaces and their implementation require knowledge of internal details about CoprHD persistent layer and CoprHD core services.
4.2 Device Discovery

Currently CoprHD does not have generic API and generic model to discover and to process data services and capabilities provided by storage devices in device independent way. Most of the capability information is statically coded in the persistent classes and across internal CoprHD services and CoprHD UI. The issue here is that unique capabilities of new storage arrays should be manually coded into CoprHD database, core services and UI.

On the device access level, the API has three parts: Discovery and Metering, Monitoring, Provisioning.

**Discovery API:**
- Scan provider to get list of storage systems.
- Discover storage system properties (serial number, IP address, firmware
version, etc.), discover storage pools, discover storage ports, and discover brownfield resources.

- Populate all data in Cassandra/ZK which CoprHD needs for provisioning operations.
- Update CoprHD task with operation status.

**Metering API:**

- Collect statistic information about storage resources: allocated/provisioned capacity of volumes and file systems, bandwidthIn, bandwidthOut for volumes and file systems, snapshot capacity, etc.
- Update CoprHD metering task with operation status.

**Provisioning API:**

- Provision storage resources on physical storage system.
- Populate physical properties for provisioned resources in Cassandra.
- Update CoprHD task with operation status.

**Monitoring API:**

- Collect events and alerts for storage arrays.
- Populate Cassandra timeseries Event table.
- Update ZK monitoring queue and update CoprHD task with operation status.
5 EMC ScaleIO Storage device

ScaleIO is a software-only solution of EMC Corporation that uses existing server or hosts’ local disks and LAN to realize a virtual SAN that has all the benefits of external storage— but at a fraction of the cost and the complexity [2]. Elastic Cloud Storage turns existing local internal storage into internal shared block storage that is comparable to or better than the more expensive external shared block storage. The lightweight ScaleIO software components are installed in the application hosts and inter-communicate via a standard LAN to handle the application I/O requests sent to ScaleIO block volumes. An extremely efficient decentralized block I/O combined with a distributed, sliced volume layout results in a massively parallel I/O system that can scale to hundreds and thousands of nodes.

ScaleIO enables administrators to add or remove nodes and capacity “on the fly”. The Software immediately responds to the changes, rebalancing the storage distribution and achieving a layout that optimally suits the new configuration. As ScaleIO is hardware agnostic, the software works efficiently with various types of storage, including magnetic disks, solid-state disks, and PCIe flash cards, networks, and hosts. It can be easily installed in an existing infrastructure as well as in greenfield configurations.

5.1 Software Components

- The ScaleIO Data Client (SDC) is a lightweight device driver situated in each host whose applications or file system requires access to the ScaleIO virtual SAN block devices [2]. The SDC exposes block devices
representing the ScaleIO volumes that are currently mapped to that host.

- The ScaleIO Data Server (SDS) is a lightweight software component that is situated in each host contributes the local storage to the central ScaleIO virtual SAN.

- Meta Data Manager (MDM) configures and monitors the ScaleIO system [2]. Minimum number of MDM in a redundant cluster is three (primary, secondary and tie breaker).

ScaleIO implements a pure block storage layout. Its entire architecture and data path are optimized for block storage access needs. For example, when an application submits a read I/O request to its SDC, the SDC instantly deduces which SDS is responsible for the specified volume address and then interacts directly with the relevant SDS. The SDS reads the data and returns the result to the SDC. The SDC provides the read data to the application.

This flow is very simple, consuming as few resources as necessary. The data moves over the network exactly once, and maximum of only one I/O request is sent to the SDS storage. The write I/O flow is similarly simple and efficient. Unlike some block storage systems that run on top of a file system or object storage that runs on top of a local file system, ScaleIO offers optimal I/O efficiency.
5.2 Protection domains

A set of SDSs are grouped into multiple protection domains. Protection domains are further grouped into a large ScaleIO storage pool. ScaleIO volumes are assigned to specific protection domains. Protection domains are useful for mitigating the risk of a dual point of failure in a two-copy scheme or triple point of failure in a three-copy scheme.

The table below summarizes how ScaleIO components map to CoprHD components during discovery.

<table>
<thead>
<tr>
<th>ScaleIO Component</th>
<th>CoprHD component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary MDM</td>
<td>Storage Provider</td>
</tr>
<tr>
<td>Protection Domain</td>
<td>Storage System</td>
</tr>
<tr>
<td>Storage Pool</td>
<td>Storage Pool</td>
</tr>
<tr>
<td>SDS</td>
<td>Storage Port</td>
</tr>
<tr>
<td>SDC</td>
<td>Host</td>
</tr>
</tbody>
</table>

Table 2. ScaleIO components map to CoprHD Components
6 Design and Approach

This section details about the implementation of a ScaleIO driver based on new South bound SDK.

![Diagram of the design implementation]

Figure 8. High level implementation of the design

In this solution the driver is not bound to CoprHD persistent data model classes and does not depend on the CoprHD infrastructure services in its implementation. The driver intermediate layer acts as an interface between the driver and CoprHD. It does not have direct access to Cassandra, Zookeeper, task completers from API methods and provides similar scope of storage operations as current CoprHD device access layer. Support for 3rd party drivers to advertise capability metadata and capability instances of their storage resources can be achieved easily and also provides capability to integrate device specific
orchestrations into CoprHD.

The device driver SDK has two parts; object model and driver API.

The object model has infrastructure independent set of domain classes which define properties of storage objects.

- CapabilityDefinition.java
- CapabilityInstance.java
- StorageSystem.java
- StoragePool.java
- StoragePort.java
- StorageHosts.java

6.1 High level template for driver API

DiscoveryDriver.java

```java
public interface DiscoveryDriver {
    // Discovery
    // Get all storage systems
    public List<StorageSystems> getStorageSystems();

    // Discover storage systems and their capabilities
    public DriverTask discoverStorageSystem(List<StorageSystem> storageSystems);

    // Discover storage pools and their capabilities
    public DriverTask discoverStoragePools(StorageSystem storageSystem,
        List<StoragePool> storagePools);
    public DriverTask discoverStoragePorts (StorageSystem storageSystem,
        List<StoragePorts> storagePorts);
    public DriverTask discoverStorageHostComponents(StorageSystem storageSystem,
        List<StorageHostComponent> embeddedStorageHostComponents);
```
6.2 Workflow

The Discovery subsystem / framework in the controllersvc service of CoprHD is responsible for discovering existing information in Storage Providers, Storage Systems, Network Systems, and Compute Systems.

Storage Providers control multiple storage systems. Some examples of storage providers are SMI-S Provider (for VMAX, VNX, etc.), Hitachi HiCommand Device Manager (for HDS arrays) and the Cinder node for cinder-managed arrays.

For Storage Systems things like the Storage Ports and Storage Pools with their capabilities are discovered so they can be used in the Virtual Arrays and Virtual Pools. If we want to ingest Volumes and their Export Relationships, we may also discover UnManagedVolumes and UnManagedExportMasks for example. The picture below shows the high level flow of the code:
Figure 9. Code flow

The flow through this code is as follows:

- The StorageProviderService method scanStorageProviders invokes the
  SMISProviderService.scanSMISProviders method which eventually calls
  BlockControllerImpl.scanStorageProviders. ScanStorageProviders creates
  a DataCollectionScanJob and calls Controllersvc.enqueueDataCollectionJob. This
  will put a scan job on the queue. The purpose of a scan job is to scan for Storage
  Systems accessible to a Storage Provider (the provider may have access to more
  than one system.)

- BlockDeviceController.addStorageSystem will typically be called for new
Storage Systems that are discovered. This method calls ControllerServiceImpl.scheduleDiscoverJobs to put a discovery job for each new system on the queue. The purpose of a discovery job is to discover the detailed information that is needed from a Storage System.

DataCollectionMeteringJobs are queued in a similar fashion. These are used to periodically collect statistics information such as volume and port usage information. The output of the metering information is used by the MeteringService as well as to do metrics based Storage Port allocation. The queues are implemented in the Coordinator Service and are global across all CoprHD nodes. After initial invocation, jobs are periodically added to the queue to rescan providers, rediscover Storage Systems, and collect metering statistics periodically. DataCollectionJobScheduler is responsible to periodically put data collection jobs to the queues according to predefined time intervals. There are limits to prevent execution of a given job too frequently. Jobs can also be queued as a result of a UI or API request to rescan or rediscover a system.

When it is time to process an item off the queue, the DataCollectionJobConsumer is invoked to process an item from any of the three queues. The DataCollectionJobConsumer creates an AccessProfile that contains all the parameters necessary to initiate communication with the system. These include things like the IP address, port, protocol, and credentials to be used.

This information is passed through the DataCollectionJobInvoker, whose job is to initiate the operation. Each Storage System type implements device specific
code in a subclass of the ExternalCommunicationInterface. This interface contains the scan, discover, and collectStatisticsInformation methods that perform the actual work, as well as additional methods for injecting CoprHD specific context into the device specific information. This context includes handles with which to access the database (DbClient), coordinatorsvc (CoordinatorClient), ControllerLockingService, and the TaskCompleter.

For this reason, the implementation of the ExternalCommunicationInterface is tightly coupled with these CoprHD internals. It will serve as an adapter for discovery requests from controller to 3rd party discovery drivers. It will expose the same interface to controller as other communication interface classes. 3rd party discovery drivers will be driven by ExternalCommunicationInterface and developers will register discovery drivers with CoprHD. It will route discovery requests from controller to 3rd party drivers based on the storage system type in the request. ExternalCommunicationInterface will process discovery data from 3rd party drivers and it will handle all communication with infrastructure required to execute the requests (Cassandra updates, interaction with ZooKeeper and task completer among these tasks).

ScaleIOCommunicationInterface is a specific implementation example of ExternalCommunicationInterface.

In the CoprHD UI, the discovery of ScaleIO system is discovered as a “scaleiosystem”. In order to discover the ScaleIO system, we need:

- The IP address of the primary MDM.
➢ The root credentials of the virtual machine (VM) or physical host where the MDM is running.

➢ The port number of the storage system.

6.3 Implementation

When CoprHD is pointed to the primary MDM:

1. If the ScaleIO system is stand-alone, a Storage Provider is created which maps to the primary MDM.

2. The Protection Domains are discovered and one Storage System is created in CoprHD for each ScaleIO Protection Domain.

3. Storage Pools are created in CoprHD: one Storage Pool in a CoprHD Storage System for each discovered Storage Pool that is part of the ScaleIO Protection Domain mapped to the Storage System.

4. Storage Ports are created in CoprHD: one Storage Port in a CoprHD Storage System for each discovered SDS that is part of the ScaleIO Protection Domain mapped to the Storage System. The name of the Storage Port maps to the name of the SDS ID.

5. CoprHD automatically creates hosts and host initiators: one host for each SDC. If the hosts were previously discovered, CoprHD adds the initiators to the already existing hosts.

After the initial discovery finishes, the following information for the Storage System displays in the CoprHD UI:
In this case, there is only one Protection Domain and therefore CoprHD has discovered one Storage System. A new Storage System with its Storage Pools and Ports is visible in **Physical Assets > Storage Systems** in the UI.

After the Storage System has been discovered, we can access its Storage Pools and Ports. The following information for the Pools and Ports are displayed in the UI:
In this case we have one Storage Pool with resource types as Thin, Thick. Drive types are of FC, SATA. It also displays the total and free amount of capacity available in GB.
In this example, we have three Storage Ports that are available in the Storage System of type ScaleIO.

It is important to note that any configuration changes to the ScaleIO system (i.e. add/remove SDS, SDC, etc.) will be discovered by CoprHD in the next polling cycle, which by default is set to 3,600 seconds. If there has been a change in the ScaleIO system that we want to immediately see in CoprHD, then we need to rediscover the ScaleIO Storage Provider manually.
7 Conclusion

This work implemented a ScaleIO storage driver for the discovery of the ScaleIO storage device, its pools and its ports using the new CoprHD Southbound API that is not bound to CoprHD persistent data model classes and does not depend on the CoprHD infrastructure services in its implementation.

3rd 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 model does not allow the driver to directly access the Cassandra, zookeeper, task completers from API methods and provides similar scope of storage operations as current CoprHD device access layer.

This helps the open source community to develop more drivers for more storage systems and easily plugin to the CoprHD using the new Southbound API.

Future work would be to integrate CoprHD with Container ecosystem and create plug-ins for Docker libstorage, Mesos, Kubernetes and improve intelligent placement algorithms for SLO-based placement and resource selection.
Bibliography


