Software-Defined Storage

Fenggang Wu (4874734) and Guobao Sun (4867911)

December 10, 2013

Abstract

Virtualization technique is one of the most promising trend of deploying storage systems which can make full use of the hardware and build manageable storage systems. However the complexity of too many layers leads to the difficulty to enforce end-to-end QoS policy. In this report, we proposed a software defined storage system which can meet this challenge and provide with better QoS provisioning.

1 Introduction

With the rapid development of storage systems, more and more storage systems are deployed and many literatures are focusing on designing superior systems. Among them, virtualization technique is one of the most promising trend of deploying storage systems. To virtualize servers, we can simply store each virtual machine (VM) in a file on the shared storage of physical servers. This mitigates some severe problems existing in large storage systems (e.g., systems that are used in data centers) like VM migration or duplication. As a result, virtualization is pervasively used today.

Although virtualization helps with building a more manageable system, the side effect of importing virtualization into systems is that there will be too many layers, which leads to increased end-to-end complexity. Here, the end-to-end process is the process starting from applications in VMs, to disks in storage systems. If we have a end-to-end policy, i.e., we want to make some restrictions on a particular end-to-end connection, it will be very hard to achieve due to the number of layers involved in this process.

Such complexity makes us believe that based on this, it is hard to put Quality of Service (QoS) requirements onto a specific end-to-end process. There are several different kinds of QoS requirements we may come up with, e.g., bandwidth requirement (bandwidth from VMs to storage disks), latency requirement (maximum latency tolerance range), or priority requirement.

All these deficiencies make it necessary to have a software-defined design of storage system, i.e., Software-Defined Storage (SDS) system. Although many companies are talking about this term [1–3], there is no clear definition about what SDS really should be, at least for now. According to Wikipedia, "Software-defined storage (SDS) is a marketing buzzword for promoting computer data storage technologies."

To try to define what SDS really is, a common way, yet also used by many companies, is to "steal" ideas from Software-Defined Networking (SDN) [8]. If applications have specific disk-related requirements, e.g. QoS requirements (bandwidth, latency, etc.), it is very difficult, or even impossible for existing storage systems to meet these needs. By introducing software-defined storage systems, what we want to achieve is to have a central controller which can have the ability to implement different policies on different kinds of traffic made by applications (in some sense we can also call it flows). This necessitates the introduce of SDS.

As we mentioned previously, if we try to define what will be contained in SDS, there are several possible features:

- Controllers are still needed. It needs to be able to interact with storage systems (or some layers of storage systems), and provide APIs to translate policies into different restrictions on the systems.

- A definition of flows. In SDN, different flows can be differentiated by host names, IP addresses, TCP or UDP connections. In SDS, we also need to make it clear about what we can use to differentiate among flows.
- Different actions should be conducted on different flows.

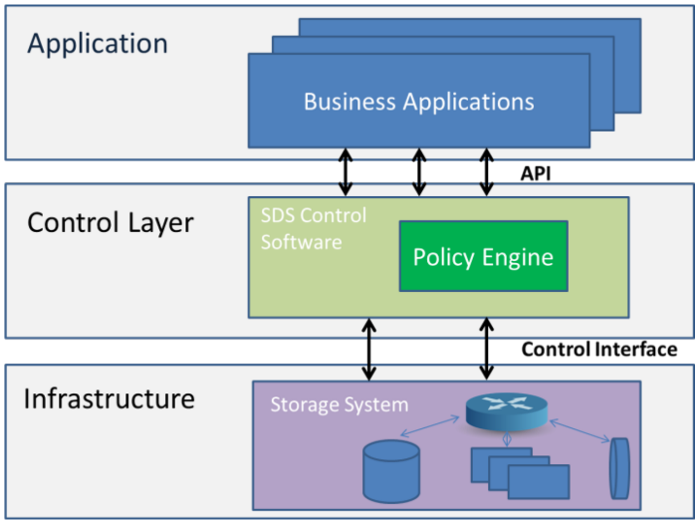

Figure 1: Possible Architecture of SDS

In Figure 1, we depicted a picture which is similar as SDN counterpart. This may be not detailed enough, and we will revise it later in later sections.

In this report, we will analyze what is needed to form a software-defined storage system, and what features the existing solutions have. Moreover, we identify the deficiencies lying in existing solutions, and try to propose a feasible solution to solve it.

The remaining part of the report is organized as follows: In Section 2 we present our motivation and the problem we would like to address. In Section 3 we discuss existing literatures that are related to this topic. In Section 4 we discuss our proposed design. Our report will conclude with our conclusion and future works in Section 5.

2 Motivation and Problem Statement

2.1 Background

Storage virtualization can improve hardware utilization and provides great flexibility on building applications on top of low cost commodity hardware . Such virtualization can reduce the cost and ease the management workload[4].

At the same time, QoS provisioning has be extensively studied in that QoS is crucial for many application running on top of the storage systems. QoS can be seen as a end-to-end issue [7][5]. However, too much layers along the data flow path leads to great complexity , making it difficult to guarantee the QoS service.

2.2 Limitation of current storage solutions

Current storage have the following limitations:

- Too many layers impede policy enforcement.

  Current virtualized storage systems exhibit many layers. For example, a simple `write` operation to the virtual disk will be translated to a block IO in the hypervisor. Then this data block will be transmitted in the storage network as packets. Finally, the written block will be delivered to the target storage server, ending up with "real" write operation to the disk[10]. Therefore end-to-end policy is quite hard to be enforced given such complexity.

- Losing control of individual block data makes it difficult to guarantee Quality of Service.

  As described above, the data block will traverse different layers until finally reaching the storage target. Once the operation is invoked by the VM application, such application will have nothing to do for controlling. Data bandwidth cannot be guaranteed. The route of the data is out of control either.

- No admission control, no resource reservation.

  From the system point of view, the overall storage system have no idea on how much of the resource has been consumed, or how much remaining can be allocated. When new tenant wants to join in this storage system, the system have no admission control to ensure the QoS of the current served tenants.

2.3 Problem Statement and Design Issues

In this subsection, we will first briefly describe the storage system this report is focusing on, then we will clarify several terms by giving the definitions. Finally we will give the statement of this Software defined storage problem.

In this report, we are focusing on small compute-purposed clusters with a group of compute servers connected with a group of storage servers (Figure 2 [9]).

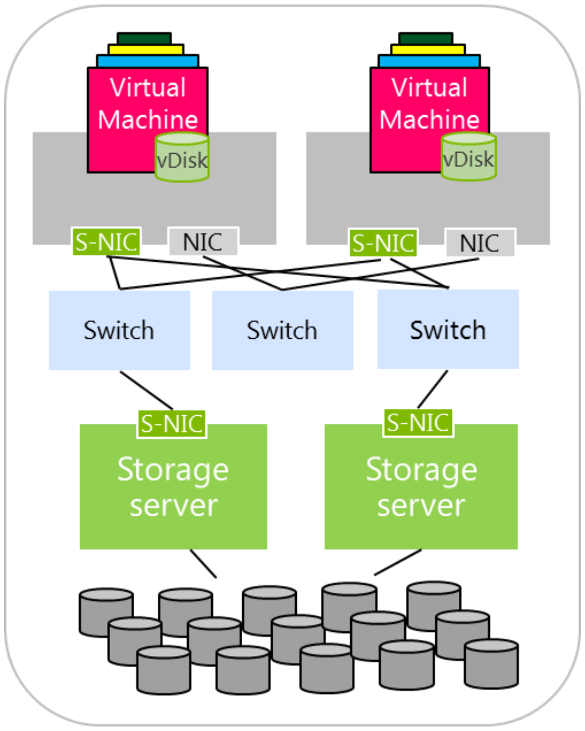

Figure 2: Computer cluster with a group of compute and storage servers

On each of the compute server, several Virtual machines (VM) is running. Each tenant will have several VMs allocated in the system.

Quality of Service (QoS): The QoS we focus here consists of bandwidth and priority.

Different tenants, VMs and files may have different bandwidth and priority requirements. The they can have minimum bandwidth limit. Also, each of them can have different priority. The tenants get their bandwidth and priority enforced when the VM is set up. The request will be rejected if there is not enough resources.

End-to-end policy: refers to the QoS policy along the data path between the VM applications and the storage target.

In this report we only consider the data flow between the VM and the storage target. The VM application and storage server disk are the two end of the data path. Plenty of policies can be enforced along this path, including bandwidth, latency, jitter, security, priority, etc. With respect to the QoS in we've discussed before, we consider the polices that are only related to the bandwidth and the priority.

The key problem here is: how to design a storage system that can help us better enforce the end-to-end QoS policy?

To design such a software-defined storage system, the following issues must be addressed. This system should be able to:

- fulfill the tenants' QoS requirements
- provide end-to-end policy enforcement.
- differentiate different kinds of data flows and deal with them repectively.
- have global view of the storage resources so as to reserve the storage resource and provide admission control.

3 Related Works

In this section we present some existing literatures that are related with our topic. Since there is no many works directly targeting at software-defined storage issues, we will also discuss some literatures around the topic.

3.1 Software-defined data center

Besides software-defined storage and storage-defined networking, there are also some people keeping talking about software-defined data centers (SDDC). Although not all papers have the name "software-defined data center" in their titles, they do accomplish something that is similar to (or actually is) it.

In [6], the authors try to build an abstract layer to which allows tenants to expose their requirements. Later, in [10], E. Thereska *et al.* move a big step ahead, and try to build a software-defined storage systems to meet the needs proposed by software-defined data centers. Although the main part of this paper is talking about the storage architecture in data centers, it does give us information that SDS is an important part of SDCC. It seems to us that a typical software-defined data center consists of three parts: software-defined computing, software-defined storage, and software-defined networking.

3.2 Software-defined networks

Although many companies are promoting their different so-called SDS products [1–3], they are all typically based on a same existing concept – Software-Defined Networking (SDN).

To help with explaining SDS, we will need to first briefly talk about SDN.

As shown in Figure 3.2, in SDN there is a controller, by using which we can alter the treatment policies for different flows without doing hardware changes or configurations. There are several important features about SDN:

- Controller needs to have a way to control lower level devices (e.g., routers, switches), and some APIs for upper level to make it possible to translate certain rules (or we may call it policies) to machine-readable commands.
- Some rules to categorize different kinds of connections in different ways, i.e. "Flows".
- Define several possible ways of handling different flows, e.g., limiting the bandwidth for a certain flow, dropping a flow, changing the flow's routing.

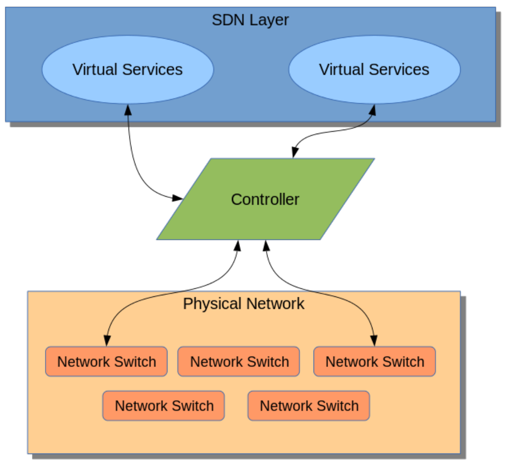

Figure 3: Overview of SDN (from Wikipedia)

OpenFlow [8] is a well known and pervasive used implementation of SDN. From that on, many literatures are focusing on this issue, and SDN eventually becomes a commonly recognized research issue. Since this is not our main point, and it is enough to explain the SDN structure with this paper, we will omit other literatures here.

4 Proposed Design

In our proposed design, we will build a central controller, which can provide a control panel for the tenants to setup their VM and even the files in the VMs. This central controller will in turn talk to two key part of the whole data path, hence enforce the QoS policy of bandwidth and priority on a per tenant, per VM or per file basis.

Firstly, we will describe the hardware architecture of the proposed system. Then we will describe the policy format by which the tenants talk to the central controller. The next will be the two key IO components that we want to modify to add control on. Finally, we will introduce the concept of IO queue, which characterize the flow and enforce the corresponding action.

4.1 Hardware Architecture

In Section 2, we have discussed about the system as a compute cluster consisting of a group of computer servers and storage servers. Here, in order to fulfill the QoS policy enforcement, a central controller is deployed so as to accept the tenants' request then translate the policy into specific controlling command, which will be in turn sent to the two key components (Figure 4.1 [10]).

Deploying a centralized controller has many benefits:

- A centralized controller will have a global view of all the storage resources. Therefore it can determine the remaining resources to make a decision on whether to allocate new VMs.
- A central controller can make better policy consistency. One policy may be deployed in different part of the system. A centralized control is better for coordinating all the controlling command to ensure each part of the storage system can work coherently.
- By this central controller, we can separate the control out of the data path. Transitional storage systems do not have a separate controlling component. All the control is distributed in the who system by different and heterogeneous parts, making the data transmission out of control. So by this kind of separate controller, we can detach the control from the data.

We will explore to the policy format and two components in detail in the following subsections.

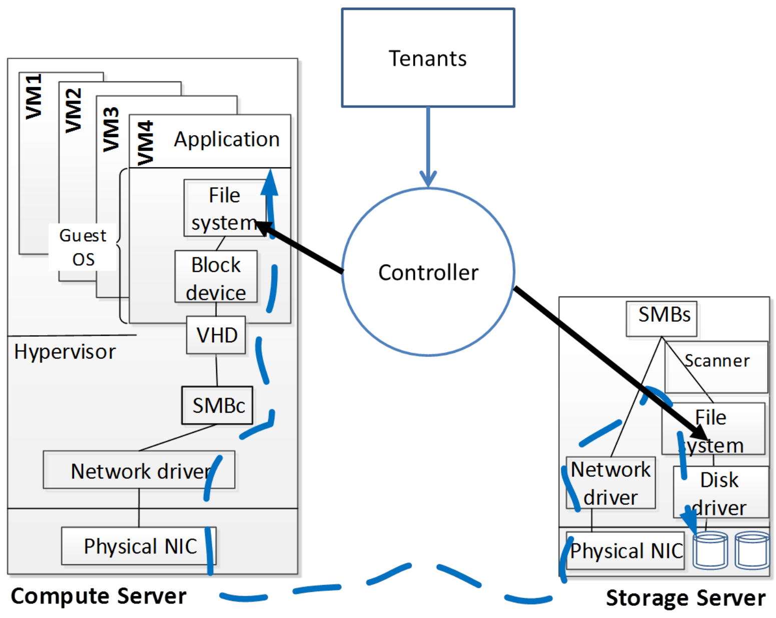

Figure 4: Hardware Architecture

Table 1: Protocol Format – Flow ID

Field	Note
Tenant	May run multiple VMs
VM	Has performance requirements
File	Requirements overrules VM requirements
Operation	read write append update

4.2 Policy format

That's talk about policy here in detail. The policy may include the identification of particular data flows and the corresponding action with them.

We want to differentiate data flows and treat them with different policy. Here several questions follows:

- How to define a data flow?
- How to identify a data flow, especially in different layers of the storage system?
- How to define a policy based on the data flow?

Here, similar to [10], we define the data flow as the data blocks that share the same fields in the this 4-element tuple:

$$< Tenant, VM, File, Operation >$$

.

The fields in the tuple can be wildcard. The description of each field can be found in table 1. We refer to this 4-element tuple as *Flow ID*, i.e. it can define different flows. Then as a result, we can associate corresponding actions with each of the flows.

To define each of the action and treat each of the flow differently, we also define the property of each of flow. Here a 3-element tuple is defined as:

$$< Throughtput, Priority, Property >$$

We call this 3-element tuple here as the *Flow Profile*, which stores the properties of this flow. Such property can be in turn translated to controlling commands to the storage system components. So

Table 2: Protocol Format – Flow Profile

Field	Note
bandwidth	requested bandwidth guarantee
Priority	Log, high; data, low; video playing apps. OK for slightly loss
Property	Read-intensive, write-intensive, update-intensive, parallel access, etc.

here, we can define one policy as a Flow ID with its associated Flow Property.

$$< Tenant, VM, File, Operation > \rightarrow < Throughtput, Priority, Property >$$

4.3 Control stages

A data flow will go through different components of the storage system until it finally reach the storage target (disks). The data have different "header" in each of the components. According to [10], we call each of the phase in the data transmission a *stage*.

We may want to identify some of the stage to put the control on it so as to enforce the QoS policies. Then the selected stages should be modified to expose several interfaces which the controller can talk to and have control on.

We select two stages to put control: the filesystem of both the VMs and the storage servers. We select them for the following reasons:

- We choose the storage server file system in that it have direct connection with the disk controller.

  Since the storage server is the last stop to access the hardware, it can directly monitor and control the hardware.

- We choose the VM file system because this is the level of stage which can recognized the VM files.

  If one policy is to be enforced on a per file basis, it should be deployed here. Otherwise, the operation to the virtual disk on the compute server will be translated to a IO operation to the remote VM image file, where we have no idea about the original file in the VM, hence have no way to apply control on it.

4.4 IO Queue

On the two stages described above, we propose to modify the file system, adding queuing mechanism in it. All incoming data will be assigned to a queue according to the Flow ID. All the data blocks having the same Flow ID will be assigned to the same queue.

Here we can have different operations on different queues. The two stages will always firstly process the queues having higher priority until it becomes empty. By doing this we can enforce the priority scheme.

Also we want to control the data rate of each of the queue. If some of the queue have more data to transmit than it requested, the stage will give back-pressure on the upstream stage. In this way we can reserve the bandwidth for other data flows, preventing it from being abused by aggressive flows.

At the storage server, we can also make data allocation decisions according to the property of different data flows. For example, for those read-intensive data, we can try to allocate them in SSD if there is any. For another example, if some data are expected to be parallel accessed, the file system will consider distribute the data to different physical location.

5 Conclusion and Future Works

Recently years have witnessed a significant trend of vitualization of the storage systems. Such virtualization provides us with high hardware utilization, low cost as well as the ease of management and platform flexibility. However, there is no free lunch. Such additional layers of abstraction and the complexity of the current storage system make it difficult to enforce satisfactory end-to-end QoS policies. We design a software defined storage system which can meet such challenges. In our proposed system, a central controller will be responsible to accept the tenants' requirement and then talk to two key stages to fulfill the control of the data flow. The two stages reside in the VM and the storage server file systems and can enforce the QoS policies such as bandwidth and priority.

Future work may includes implementing this proposed design into prototype and extending this design to other large scale storage systems.

References

[1] Emc vipr. `http://www.emc.com/data-center-management/vipr/index.htm`.

[2] The fundamentals of software-defined storage (coraid). `http://san.coraid.com/rs/coraid/images/SB-Coraid_SoftwareDefinedStorage.pdf`.

[3] Vmware virtual san. `http://www.vmware.com/products/virtual-san/`.

[4] Agrrawal, A., Shankar, R., Akarsh, S., and Madan, P. File system aware storage virtualization management. In *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on* (2012), IEEE, pp. 1–11.

[5] Aurrecoechea, C., Campbell, A. T., and Hauw, L. A survey of qos architectures. *Multimedia systems 6*, 3 (1998), 138–151.

[6] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. I. Towards predictable datacenter networks. In *SIGCOMM* (2011), vol. 11, pp. 242–253.

[7] Lu, Y., Du, D. H.-C., and Ruwart, T. Qos provisioning framework for an osd-based storage system. In *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on* (2005), IEEE, pp. 28–35.

[8] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[9] Thereska, E., Ballani, H., O'Shea, G., Karagiannis, T., Rowstron, A., Talpey, T., Black, R., and Zhu, T. Ioflow: a software-defined storage architecture, `http://research.microsoft.com/en-us/um/people/etheres/research/thereska_ioflow_sosp13.pptx`.

[10] Thereska, E., Ballani, H., O'Shea, G., Karagiannis, T., Rowstron, A., Talpey, T., Black, R., and Zhu, T. Ioflow: a software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 182–196.