Multi-dimensional Storage QoS Guarantees for an Object-Based Storage System*

Fei Mu, Jiwu Shu, Bigang Li, and Weimin Zheng

Department of Computer Science and Technology, Tsinghua University,
100084, Beijing, P.R. China
mufei@mails.tsinghua.edu.cn

Abstract. The Object-based storage is an emerging storage architecture that could easily fulfill multi-dimensional storage QoS requests. This paper focuses on providing QoS guarantees under Object storage infrastructure along the three most prevalent dimensions: capacity, bandwidth and latency through storage resource allocation and IO commands scheduling. Firstly we propose an algorithm on storage resource mapping derived from Toyoda algorithm, which achieves efficient resource utilization through consideration of the OSDs' serving ability. Secondly we propose an object commands scheduling mechanism and develop a prototype system based on the Lustre filesystem. Through adding timestamp to each object command and scheduling the command queue by final finish time, the system can efficiently fulfill the demands on latency from the front applications.

1 Introduction

As becoming more and more complicated, storage applications urgently demand Quality of Service (QoS) guarantees along multiple dimensions such as storage capacity, data bandwidth, respond latency, reliability, security, etc [1][2][3]. For example, a voice over IP service requires a short latency; a VOD service requires a large capacity and a high bandwidth while an E-mail service may require a relatively lower capacity and a lower bandwidth but a higher security. Among them, capacity, bandwidth and latency are commonly regarded as most prevalent.

Object Storage Device (OSD) [4][5] is an emerging storage technology in recent years. It offloads storage management from host operating systems to intelligent storage devices and provides an object-level storage interface in contrast to the conventional block-level storage interface. The OSD interface is focused on moving chosen low-level storage, space management, and security functions into storage devices to enable the creation of scalable self-managed, protected and heterogeneous shared storage for storage networks.

The special architecture of Object-based storage systems achieves convenience in providing storage QoS guarantees [6][7][8]. It has better scalability by separating the

* Supported by National Grand Fundamental Research 973 Program of China under Grant No. 2004CB318205.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part III, LNCS 3993, pp. 687 – 694, 2006.

control path from data path, which makes the storage system be able to integrate large quantities of devices to achieve a high bandwidth. Furthermore, the front applications access data from the object storage devices through an *object* level interface. Accordingly, we can use the concept of *class* to identify different storage applications and the multi-dimensional storage QoS requests can be expressed as *attributes* attached to each object. Hence, both the front clients and the end devices can be knowledgeable of what the front applications demand. In object-based storage systems, the end storage device has computational ability, which means that the object device can adjust its object command queue to fulfill the application request.

Yiping Lu [6] and Joel C.Wu [7] proposed QoS frameworks for OSD-based storage system, which showed that OSD-based storage system is an ideal platform to provide QoS guarantees. Kevin KleinOsowski provided suggestions for improving the OSD specification and its ability to communicate QoS requirements [8]. His work concentrated only on bandwidth guarantee. StoneHenge [9] is a multi-dimensional storage virtualization system, which is able to multiplex multiple virtual disks with a distinct bandwidth, capacity, and latency attributes. But this work was based on a conventional cluster environment using block-level storage devices.

The Lustre filesystem [10][11][12] runs today on many of the largest Linux clusters in the world. At the root of Lustre is the concept of object storage. It is a good platform for testing and validating OSD concepts such as storage QoS guarantees.

We propagate our research work in two steps to fulfill the QoS request along capacity, bandwidth and latency. Firstly we allocate the total storage resource for application classes according to their QoS requirement. But for an actual storage system the workload is quite complicated and only storage resource allocation can not assure the access latency. Thus we also propose a scheme on command scheduling and propagate a prototype system to provide latency guarantee based on Lustre filesystem.

2 Storage Resource Mapping Algorithm

This storage resource mapping algorithm concentrates on the dimensions of capacity and bandwidth because various disk performance requirements can be readily translated into bandwidth requirements. According to reference [13], we have

$$T_i \le \frac{\omega + L_{\max}}{B_i} + \frac{L_{\max}}{B} \tag{1}$$

In equation 1, T_i is latency of application class i, ω is the data size in the waiting queue, L_{max} is the maximum size of a single command, B is the bandwidth for a certain physical device, which could all be acquired by experiment and experience. So we can translate a request on latency into that on bandwidth. We adopt the bigger one during resource mapping.

2.1 Problem Statement

Suppose there are N object devices in the storage system which provides M application classes. Each storage application has a demand on both capacity and bandwidth which is beyond the ability of a single object device. Commonly one application

should be evenly scattered into m_j parts till each part could be guaranteed by a single object device. m_j will be settle upon different policies according to different application classes. Finally there will be $K = m_1 + m_2 + \ldots + m_M$ sub-applications in the storage system, and our study focuses on how does one single object serve the scattered applications when $K>N$.

We use a two-dimension vector $\mathbf{S}=\{C, B\}$ ($C>0, B>0$) to express a sub-application, C indicates the capacity while B indicates the bandwidth. An object device could be expressed by this vector too. In special, we use **O**, **A**, **R** to represent the original resource vector**,** allocated resource vector and reserved resource vector. We get **O=A+R**. Then we assume that the at last object device i selects n_i sub-applications to serve and the result set is $\{\mathbf{S_{ij}}, 0 < j <= n_i\}$. Ideally there could be:

$$\begin{cases} \vec{A}i = \sum_{j=1}^{n_j} \vec{S}_{ij} = \vec{O}_i (i = 1,2,\cdots,N) \\ \sum_{i=1}^{N} n_i = \sum_{j=1}^{M} m_j = K \end{cases} \tag{2}$$

This is a multi-dimensional bin-packing problem and we propose a greedy algorithm derived from Toyoda algorithm [14].

We use **<A, O>** to present the included angle between vector **A** and vector **O**, which is assumed to be lied between $(0, \pi]$. There is:

$$I_i = \frac{\vec{A}_i \cdot \vec{O}_i}{\| \vec{A}_i \| \cdot \| \vec{O}_i \|} = \cos < \vec{A}_i, \vec{O}_i > \tag{3}$$

2.2 Storage Resource Mapping Principle

The resource mapping should adapt to the devices' serving ability. In virtualization work, the physical devices are always logically divided into storage slices for different virtual storage applications. The resource vector of a single slice has the same direction as that of the whole storage device. So we should minimize the included angle between the allocated resource vector and the original resource vector. Through equation 3, our goal is to maximize I_i. Nowadays many storage systems adopt replicas to enhance reliability and performance, so the resource utilization needn't be strictly maximized.

We concentrate on the **O**, **A**, **R** of a certain object device. We calculate I for every received sub-application. If $\|\mathbf{A+S}\| < \|\mathbf{R-S}\|$, then **<O, A+S>** is larger than **<R-S, O>**, we calculate I by equation 4; while if $\|\mathbf{A+S}\| > \|\mathbf{R-S}\|$, **<O, A+S>** is smaller than **<R-S, O>**, we calculate I by equation 5. Finally, the sub-application which has the largest I should be chosen to be mapped on the certain object device.

$$I = \frac{(\vec{R} - \vec{S}) \cdot \vec{O}}{\| \vec{R} - \vec{S} \| \cdot \| \vec{O} \|} \tag{4}$$

$$I = \frac{(\vec{A} + \vec{S}) \cdot \vec{O}}{\| \vec{A} + \vec{S} \| \cdot \| \vec{O} \|} \tag{5}$$

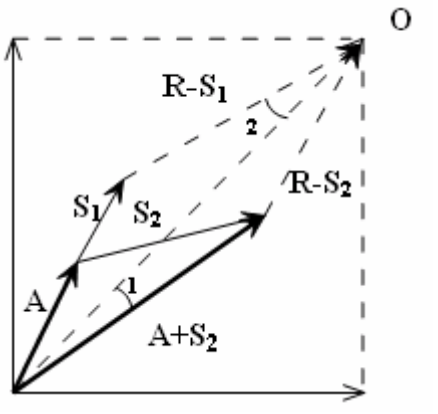

Fig. 1. $\|\mathbf{A+S_1}\| < \|\mathbf{R\text{-}S_1}\|$, then $\angle 2$ is smaller than $< \mathbf{A+S_1, O} >$; $\|\mathbf{A+S_2}\| > \|\mathbf{R\text{-}S_2}\|$, then $\angle 1$ is smaller than $< \mathbf{O, R\text{-}S_2} >$. $\angle 1$ is smaller than $\angle 2$, so at last we choose $\mathbf{S_2}$.

2.3 Algorithm Description

The single object device allocation algorithm could be described as below:

$\{\mathbf{S_j}\}$: the sub-application set.
$\{\mathbf{O_i}\}$: the object device set.
N: the number of object devices.
$\mathbf{O_i}$: the original resource vector of the object device *i*.
$\mathbf{A_i}$: the allocated resource vector of the object device *i*.
$\mathbf{R_i}$: the left resource vector of the object device *i*.

```
While {Sj} != φ or there exists a available object device
    For each available element in {Oi}
            For each elements in {Sj}
            {
                    If Ri couldn't guarantee S, continue;
                    For a certain S, calculate I by equation 4 and 5.
            }
            If all Sj can not be served by Oi, mark Oi as unavailable, continue.
            Select the S for object device i according to chapter2.2.
            Ai:=Ai+ S, Ri:=Ri- S, {Sj}={Sj}-{S}
    }
}
```

If the OSD mapping process ends when none OSD is available, new object devices should be added into the OSD storage system.

2.4 Simulation Results

We generate *N* random object device vectors {O} , N^2 random sub-application vectors {S} and run the mapping the OSD mapping algorithm for 500 times to get an average.

Table 1 indicates that our algorithm achieve better adaptation to physical device's serving ability than Toyoda Weighted algorithm when achieving a good resource utilization.

Table 1. Simulation results1

	N	5	10	50	100
<A,O> (degree)	The algorithm presented here	5.516	3.633	1.580	1.174
	Toyoda Weighted algorithm	5.737	3.713	1.812	1.293
resource utilization (when resource is used up)		74.65%	83.90%	94.17%	96.30%

3 Command Scheduling Scheme

In this chapter we focus on command scheduling to provide access latency guarantee for a front application under Lustre filesystem.

3.1 Latency Guarantee System Architecture

The latency guarantee system is designed and implemented under the Lustre filesystem, which is consisted of three parts: Clients, Metadata Servers (MDS) and Object Storage Targets (OST). Lustre clients run the Lustre filesystem and interact with OSTs for file data I/O and with MDS for namespace operations.

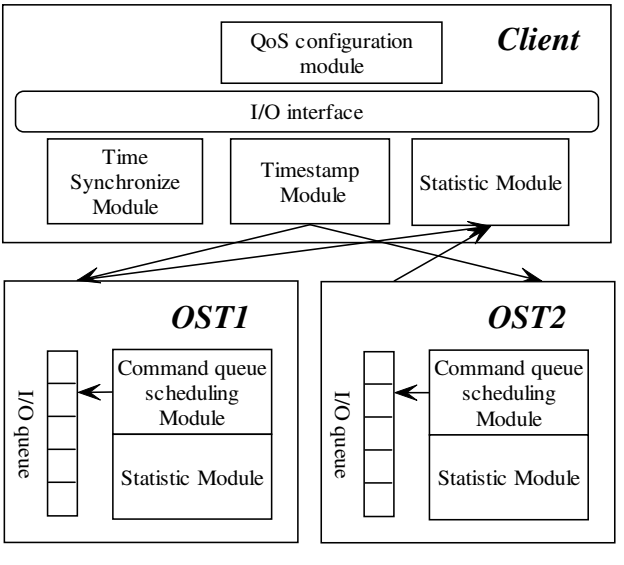

Fig. 2. The architecture of latency guarantee system

As shown in figure 2, the latency guarantee system is made up of four main components. The timestamp module receives the front object commands, recording their arrival time. It also responds for retrieving latency attributes from the objects according to various kinds of operations and adding which to the object command request. The command queue scheduling module adjusts the command queue through the information of the object command provided by the timestamp module. The statistic module will concentrate on the actual latency of the certain application and interact with the QoS configuration module. The QoS configuration module can set important parameter of the system such as common latency, coefficient for a given latency request, etc. It could indicate if the system resource is sufficient, and if the latency attributes specified by users are appropriate.

3.2 Key Technologies

Extended object commands. Between Lustre clients and the object storage targets the basic communication unit *struct ptlrpc_request* contains Lustre request message and Lustre reply message, which are all of the type *struct lustre_msg*. It is the most fundamental unit of the Lustre network protocol. This data structure is logically associated with the *object command*. In our implementation, the latency information is integrated into the *lustre_msg* while the arrival time of a certain command is integrated into the *ptlrpc_request*.

Command queue scheduling. In current Lustre system, the new arrived command request is directly added to the tail of the command queue. For a certain command request, its finish time should not be later than T^F:

$$T^F = T_{arrival} + T_{latency} \tag{6}$$

Because $T_{arrival}$ and $T_{latency}$ could be acquired precisely, we use T^F to adjust the command queue. For the commands without QoS demand, we assume their $T_{latency}$ to be the longest $T_{latency}$ required in the whole storage system. This value could be modified according the statistic information. We calculate T^F when receiving a new command, then search along the command queue from the tail to find the first command which has a smaller T^F and insert the new coming command behind it. During this process, if we meet a command having the same IO object ID with the new arrival command, then just insert the new command right behind it. Hence, the commands in the queue are nearly sorted by their T^F. The sequence of the commands which come from the same application or access the same destination has been maintained too.

3.3 Testing Results

In our testing environment, the client, MDS and OST are located in the same server. We run two applications with high workload. We give a shorter latency guarantee to application 1, while a longer one to application 2. Our statistics module counted that the latency guarantee ratio for application 1 and application 2 are 95.6% and 99.1%.

Experiment results show that the quantity of command requests is approximately linear to the total recommended data buffer size, so we use IOPS to evaluate the effect

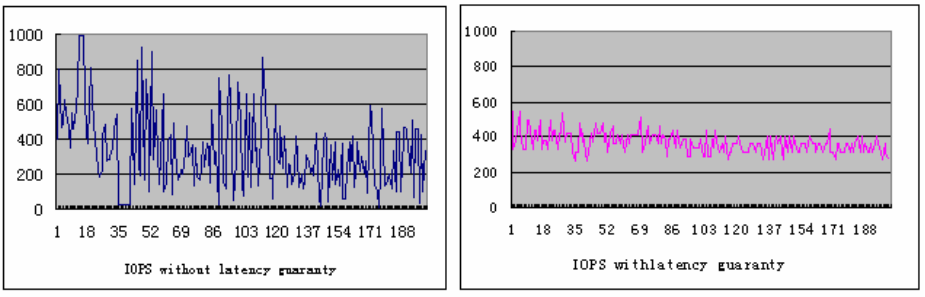

Fig. 3. Testing results of the latency guaranty system. The x axis presents the seconds and the y axis presents the IOPS.

of latency guarantee system. We focus on application 1, its IOPS without latency guarantee system and that with latency guarantee system are showed as Fig 3.

4 Conclusion and Future Work

This paper aims at providing multi-dimensional storage QoS guarantees for an Object Storage System. We focus on the three most prevalent dimensions: capacity, bandwidth and latency. We propose a storage resource allocation algorithm based on the principle of adapting to physical devices' serving ability when achieving efficient storage resource utilization. Considering the complicated application workload, we also propose a command scheduling scheme under Lustre filesystem. Testing result from the prototype system shows that the command scheduling scheme is impactful.

We will go on to study the load balancing between object devices. Command scheduling within the object devices is also challengeable. We will also concern about the emerging storage QoS dimension such as reliability, security, etc.

References

1. C. R. Lumb, A. Mrchant, G. A. Alvarez: Façade: virtual storage devices with performance guarantees, In Conference on File and Storage Technology (FAST 03), 2003
2. Z. Dimitrijevic and R. Rangaswami: Quality of service support for real-time storage systems, in Proc. of Intl. IPSI-2003 Conference, 2003.
3. M. de Miguel, J. Ruiz, and M. Garcia. QoS-Aware Component Frameworks. In Tenth IEEE International Workshop on Quality of Service, pages 161-169, May 2002.
4. M. Mesnier, G. R. Ganger, E. Riedel: Object-Based Storage. IEEE Communications Magazine, Vol. 41, No.8, pp 84-90, August 2003
5. R. O. Webster. Information Technology - SCSI Object-Based Storage Device Commands (OSD). February 2004. Rev 9.
6. Yingping Lu,David H.C. Du, Tom Ruwart: QoS Provisioning Framework for an OSD-based Storage System. NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05), 2005

7. Joel Wu and Scott A. Brandt, "QoS Support in Object-based Storage Devices," International Workshop on Storage Network Architecture and Parallel I/O (SNAPI 05), held in conjunction with the International Conference on Parallel Architectures and Compilation Techniques (PACT 2005), Saint Louis, Missouri, September 17–21, 2005
8. Kevin KleinOsowski, Tom Ruwart, David J. Lilja:Communicating Quality of Service Requirements to an Object-Based Storage Device. NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05), 2005
9. Lan Huang, Gang Peng, and Tzi-cker Chiueh. Multi-dimensional storage virtualization. SIGMETRICS Perform. Eval. Rev., 32(1), 2004, 14~24
10. P. Schwan. Lustre: Building a file system for 1000-node clusters. In Proceedings of the 2003 Linux Symposium, July 2003
11. Lustre project, http://www.lustre.org
12. P. J. Braam. The Lustre storage architecture, 2002
13. A. K. Parekh, R. G. Gallagher. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. IEEE/ACM Transactions on Networking, 2(2), 1994, 137~150
14. Y. Toyoda: A simplified algorithm for obtaining approximate solutions to zero-one programming problems. Management Science, 21(12):1417–1427, Aug. (1975).