Applied Mathematics & Information Sciences 5 (2) (2011), 53S-60S
– An International Journal

A Decentralized Resource Allocation Approach for Response-time Guarantees in Storage System

Liu Liu[1] , Lu Xu[2] and Dezhi Yang[3]
[1]Institute of Computing Technology, Chinese Academy of Science, Beijing 100190, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
National Engineering Laboratory for Disaster Backup and Recovery, Beijing 100876, China
[3]PetroChina Planning and Engineering Institute, Beijing 100083, China
Email Address: liuliu@nrchpc.ac.cn

Received June 18, 2010; Revised January 18, 2011

We present a decentralized resource allocation approach, DRG (Decentralized Response-time Guarantee), that can provide response time guarantees for multiple concurrent workloads sharing a back-end storage system in a distributed manner without assuming any support from the storage itself. This new approach uses several run-time statistics of both workloads and system as indicators of busty and load condition at the backend storage and accepts a control equation periodically to adjust the number of I/O requests which could be issued per workloads to meet the performance goals. Using a real I/O trace, we demonstrate that our approach can simultaneously meet the response-time requirements imposed by an SLO without requiring extensive knowledge of the underlying storage system.

Keywords: storage systems, resource allocation, Quality of Service, performance virtualization.

1 Introduction

I/O consolidation is a growing trend in production environments due to the increasing complexity in the management storage systems. A consequence of this trend is the need to serve multiple users and/or workloads simultaneously. It is imperative to ensure that these users could be insulated from each other in order to meet any service-level objective (SLO) especially in a distributed storage environment. Previous distributed proposals for performance virtualization basically provided only the throughput guarantees and may not always exploit the full bandwidth offered by the storage system.

With the advent of Storage Area Network (SAN) and Network Attached Storage (NAS) technology, more service providers and enterprises are increasingly mapping application workloads onto shared pools of storage resources. Consolidation of these backend storage resources brings ease of backup, flexibility in provisioning and centralized administration, and makes economic sense, but they can induce inter-workload interference from which arises performance degradation and lack of performance predictability. As a result, resource management mechanisms are required to enable performance isolation and to enforce Service-Level Objective (SLO). Such isolation is called performance virtualization since it gives the impression that the storage utility is fully devoted to each workload. One way to achieve this virtualization is an intrusive-approach implemented in internal resources such as disk schedulers or cache [1] [2] [3]. Because of scale and complexity these approaches are not very attractive from the practical point of view. We would like to use a nonintrusive approach which treats the storage utility as a black box throttling requests before dispatching them to storage. A centralized nonintrusive scheduler which has complete control over all requests can supply both throughput and response time virtualization although existing decentralized approaches still have the limitation of providing response time guarantees.

2 Related work

One way to achieve I/O performance virtualization is to perform resource provisioning/partitioning across workloads in a static manner [6][7], which is typically coarse-grained and unable to handle short-term transient conditions. Most of the online fine-grained centralized solutions can provide both throughput and response time guarantees, like Façade[8], SFQ(D)[4](using GPS[9]), Avatar[10], while distributed solutions, such as Triage[11], RW(D)[4], basically can only control throughput. Although PARDA[5] can control device latency, it still cannot control response time which is composed of not only device latency but also host-queuing delay. This article presents a decentralized resource allocation approach, DRG, which we believe is the first scheduler that can provide response time guarantees for open workload in a distributed storage system. Based on run-time workload characteristic and storage status, DRG periodically reallocate storage resource to meet the response time goals of concurrent workloads.

3 Decentralized Framework

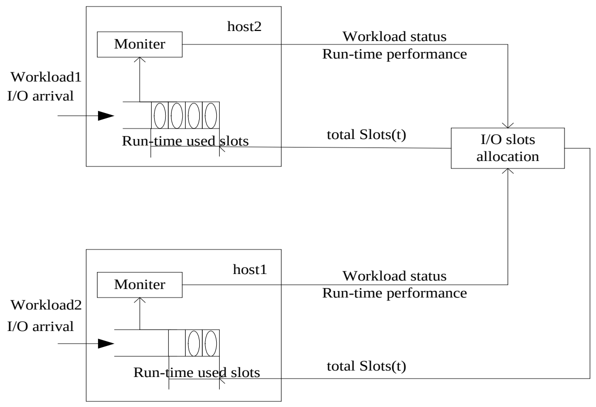


Figure 2.1: The architecture of decentralized resource allocation framework

Our decentralized framework is composed of 2 major components: Monitors which collect run-time workload characteristics and performance on distributed hosts and the

I/O slots allocation module invoked periodically dynamically to reallocate resources.

4 Controlling Principles

The basic rule to meet the response time is controlling the separate two parts of the response time: host-queuing delay and device latency. The former is influenced by dynamic behavior of the workload while the latter relies on specific storage status. We conceptually partition the array queue among workloads just like PARDA does. Furthermore we implement the I/O slots allocation model which adapts the device queue length per workloads according to dynamic resource requirement which can be concluded from the run-time infomation collected by monitors.

Table 4.1: Symbols used and their descriptions

$\lambda_i(t)$	average arrival rate in period t of workload i
$q_i^0(t)$	length of queuing requests at host of workload i at the end of period t
$que\_d_i(t)$	the queuing time to be spend on workload i during last adaptation time window to time period t
$d_i^0(t)$	the queuing time already spent on $q_i^0(t)$
$us_i(t)$	average used slots of workload i in period t
$us(t)$	system-wide average used slots of workload i in period t
$ts_i(t)$	total slots of workload i in period t
$l_i(t)$	average device-level latency of workload i in period t
$r_i(t)$	average response-time of workload i in period t
ℓ	system-wide device latency threshold
n	number of concurrent workloads

4.1 Controlling Device Latency

The used-slots computation uses a control mechanism shown to exhibit stable behavior for the FAST TCP flow-controlling algorithm. For latency estimation each host maintains an exponentially-weighted moving average of I/O latency denoted by $l_i^{'}(t)$ at time t to smooth short-term variations. Then we can use the FAST-TCP equation to maintain the latency of the device stable at ℓ.

$$l_i^{'}(t) = (1-\alpha)\times l_i(t) + \alpha \times l_i(t-1) \tag{4.1}$$

$$us(t) = (1-r)\times us(t-1) + r \times \frac{\ell \times n}{\sum l_i^{'}(t)} us(t-1) \tag{4.2}$$

According to Eq. (4.2) we can get the threshold of system-wide used slots in time window t. All following experiments used the smoothing parameters { α = 0.002 and r =

0.8} just like PARDA did.

4.2 Controlling Queuing Delay

Using one-step prediction, we assume $\lambda_i(t)$ would be $\lambda_i(t-1)$. Let μ_i denote the estimated service rate in the next window. We assume that the arrival and service rate are constant, $q_i(t)$, and average queue length in the next adaptation windows is given by:

$$q_i(t) = q_i^{\,0}(t-1) + (\mu_i - \lambda_i(t-1)) * t \tag{4.3}$$

$$avg\_q_i(t) = \frac{1}{w}\int_0^w q_i(t)dt \tag{4.4}$$

In a simpler situation, when considering value of w is 1 second, then avg_q_i is the average of $q_i^{\,0}(t)$and $q_i^{\,0}(t-1)$. We assume that the number of requests arriving during any given time window at the storage utility approximates the number of requests that leave it. Based on this assumption, we apply Little's law:

$$que\_d_i(t) = \frac{avg\_q_i(t)}{\lambda_i(t)} \tag{4.5}$$

However, this has not considered the queuing time $d_i^0(t-1)$ already spend on $q_i^{\,0}(t-1)$ and so we adjust this equation below：

$$que\_d_i(t) = \frac{avg\_q_i(t)}{\lambda_i(t)} + \frac{d_i^0(t-1)}{w} \tag{4.6}$$

4.3 Controlling Response Time

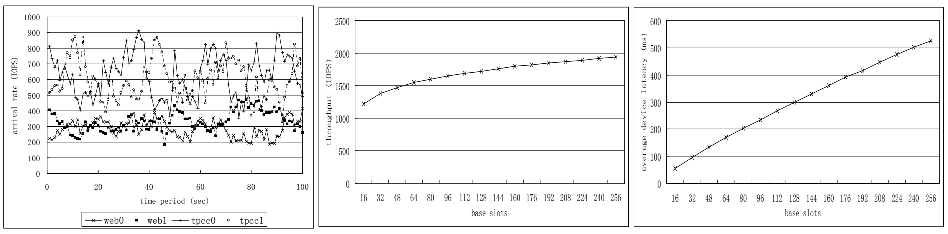
$$r_i^{'}(t) = que\_d_i(t) + \ell \tag{4.7}$$

After getting the relationship of device latency and queuing delay with needed storage resource, we can using the Lagrange multiplier method to get $us_i(t)$ which reflects resource allocation aiming to maximize the system-wide utility evaluation function. How to determine a good utility function is out of the scope of this article. Here we simply used the violated number as our evaluation metric.

5 Performance Evaluation

We evaluate our approach via a simulation-based analysis that is driven by 4 concurrent real workloads: Web0 and Web1 [13], tpcc0 and tpcc1 [14]. We use Disksim 4.0 [15] as the storage system simulator with a RAID 0 with 8 Seagate Cheetah9LP disks.

5.1 Storage performance capacity

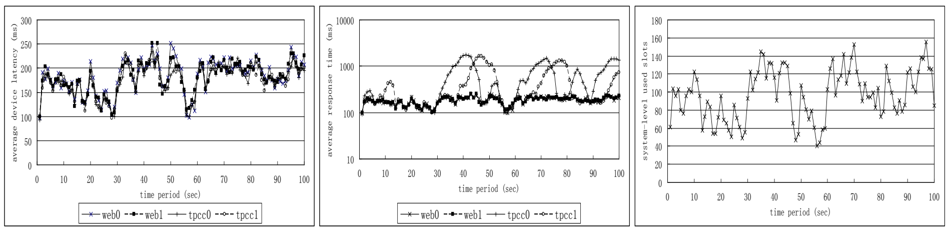


(a) dynamic arrival rate (b)throughput vs. slots (c) device latency vs. slots
Figure 5.1: characteristic of 4 workloads and back-end storage system

Unlike other sharing resources, storage performance capacity is variable according to workload characteristic and storage inner-character (organization, scheduling algorithm, cache etc). Figure 5.1 presents the characteristics of workloads and storage in different concurrent degrees. “Base slots” means max I/O slots for each workload. Obviously higher concurrent degree can bring higher throughput in the cost of larger device latency. So DRG can choose the maximal system-wide used-slots threshold-based smallest response time according to Figure 5.1(b)(c) which can be inferred using run-time information. For example, when the small response time is 200, the system-wide used slots need to be fewer than 320(80*4).

5.2 Device Latency vs. Response Time

In this Section we demonstrate the relationship between “used slots” and performance. We set each slot’s threshold to 128 and use RW(D) schema. The latency and response time over the duration of the experiment are presented in Figures 5.2(a) and 5.2(b). The used slots over the duration of the experiment are presented in Figure 5.2(c). Comparing Figure 5.2(a) and Figure 5.2(c), we can see that host-queuing delay is more influenced by the dynamic behavior of the related workload. Meanwhile the device latency is more likely to be influenced by used slots. So we can maintain it at a stable value by controlling the used slots which also represents specific storage load and capacity statistic.



(a) device latency (b)response time (c) used slots
Figure 5.2: performance and System-wide used slots under RW(D)

5.3 RW(D) vs. DRG

We firstly consider the most common sharing environment without any I/O controlling scheme. There is one main observation from Figure 5.3: because there is no queuing time but only the same device latency, the response times of all workloads are exactly the same.

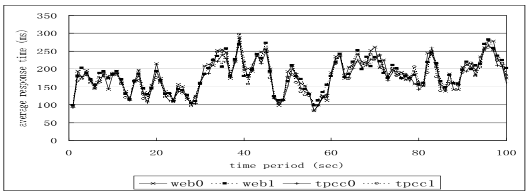
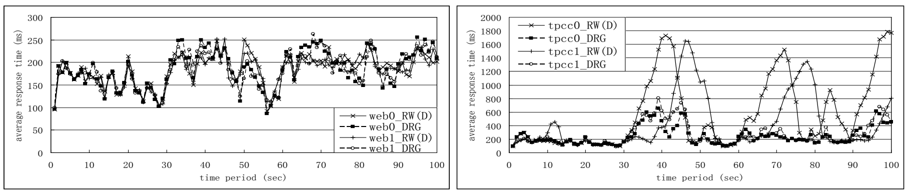


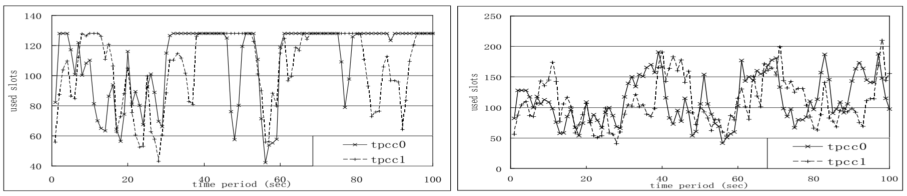
Figure 5.3: Response time of 4 concurrent workloads without any I/O controlling scheme

We use two different SLOs: SLO_A<250, 250, 800, 800> and SLO_B<400, 400, 250, 250 >. For example, SLO_A means web workloads need a response time less than 250ms. Once larger than 250ms, we state that a violation happened. The response time and used slots with SLO_A are presented in Figure 5.4 and Figure 5.5 while the response time and used slots with SLO_B are presented in Figure 5.6 and Figure 5.7.



(a) Response time of 2 web workloads (b) Response time of 2 tpcc workloads

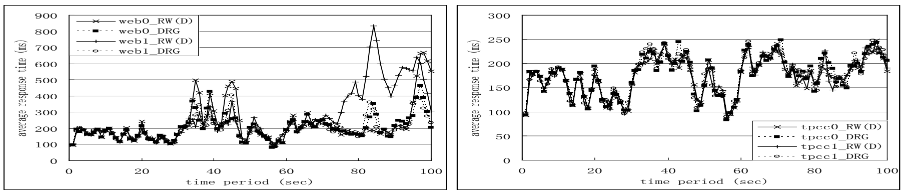
Figure 5.4: Response time of 4 concurrent workloads come with SLO_A



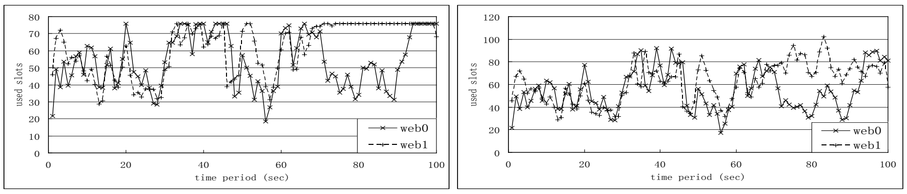
(a) Used slots under RW(D) (b) Used slots under DRG

Figure 5.5: The slots used by tpcc0 and tpcc1 workloads come with SLO_A

We observe that web workloads meet SLO_A. The violation rate of tpcc workloads under DGR is obviously much better than RW(D): 28% for tpcc0 and 20% for tpcc1 under RW(D) while only 1 violation for tpcc1 and 0 violation for tpcc0 under DRG.



(a) Response time of 2 web workloads (b) Response time of 2 tpcc workloads
Figure 5.6: Response time of 4 concurrent workloads come with SLO_B



(a) Used slots under RW(D) (b) Used slots under DRG
Figure 5.7: The slots used by web0 and web1 workloads comes with SLO_B

Just as the behavior with SLO_A, DRG can meet the response time need of tpcc workloads. DRG provides better violation ratios than RW(D) for web workloads: 5% for web0 and 14% for web1 under RW(D) while no violation happen to web workloads under DRG. The reason behind the better response-time guarantees is relying on better usage of system-wide resources which we can see from how slots were used. Just as Figure 5.5 and 5.7 show to us, under RW(D), while the queuing delay of one workload increased heavily because of lacking usable slots, the other cannot fulfill its slots, and vice versa, but under DRG we can see better usage situation of slots.

6 Conclusion

This article presents a decentralized resource allocation approach DRG which can provide response-time guarantees for concurrent workloads in a distributed storage system. Our solution decouples the control of device latency and host-queuing delay, making it more flexible and efficient to use the shared storage resources. DRG firstly uses the FAST TCP flow control algorithm to maintain the device latency at a predictable stable value, and then uses run-time statistics of both workloads and system collected by distributed hosts as parameters in Little’s law-based equation to find the relationship between queuing delay and resource needs. Evaluation of DRG shows that it is able to provide response-time guarantees to the concurrent workloads sharing the same back-end storage array. As future work we are trying to integrate throughput guarantee to provide a complete I/O resource allocation framework.

Acknowledgements: This work is supported in part by the National High-Tech Research and Development Plan of China under grants No. 2009AA01A403, China Information Security Special (NDRC) Fund for Disaster Backup and Recovery Standard

Architecture, and China Information Security Special Fund (NDRC) for Disaster Backup and Recovery Product Industrialization.

References

[1] P. J. Shenoy and H. M. Vin. Cello: a disk scheduling framework for next generation operating systems. *In Proc. ACM SIGMETRICS'98/Performance '98*, pages 44－55. ACM Press, 1998

[2] Huang,L., Peng, G. and Chiueh. Multi-Dimensional storage virtualization. *In Proc. ACM SIGMETRICS'04/Performance '04*, June, 2004.

[3] Goyal, P., Jadav,D., Modha,D. S. and Tewari,R. CacheCOW: QoS for storage system caches. *In Proc. IWQoS*, 2003.

[4] Jin, W. and Chase, J. Interposed proportional sharing for a storage service utility. *In Proc. ACM SIGMETRICS'04/Performance '04*, June, 2004.

[5] A. Gulati, I. Ahmad and C. Waldspurger. Parda: Proportional allocation of resources in distributed storage access. *In USENIX FAST*,2009.

[6] G. A. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *In ACM Transactions on Computer Systems*, 19(4):483–518, November 2001.

[7] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal and A. Veitch. Hippodrome: running circles around storage administrators. *In USENIX FAST*,2002

[8] Lumb, C., Merchant, A. Facade: Virtual storage devices with performance guarantees. *In USENIX FAST*,2003.

[9] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services network- the single node case. *In Infocom*,1992

[10] Zhang, J., Sivasubramaniam, A., Wang and Q., Riska, A. Storage performance virtualization via throughput and latency control. *In Proc. MASCOTS*, 2005

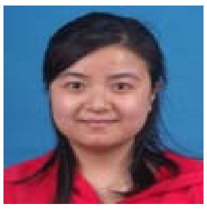
[11] Karlsson, M., Karamanolis, C. and Zhu, X. Triage: Performance isolation and differentiation for storage systems. *In Proc. of the 9th IWQoS*, 2004.

[12] Jin, C., Wei, D. and Low, S. FAST TCP: Motivation, Architecture, Algorithms, Performance. *In IEEE/ACM TON* , Volume 14 Issue 6, Dec, 2006

[13] Websearch traces. http://traces.cs.umass.edu/index.php/Storage/Storage, 2006

[14] TPC-C traces, http://tesla.hpl.hp.com/opensource/, 1994

[15] Ganger, G. and Worthington, B. *The DiskSim Simulation Environment Version 2.0 Reference Manual*. http://www.pdl.cmu.edu/DiskSim/, 2006



Liu Liu received the MS degree in Computer Science from Xi’an Jiaotong University (XJTU) in 2005 and is currently a PhD candidate in the ICT/CAS. Her research interests are in the areas of Parallel I/O and Storage Architectures, and Performance Virtualization.