2014 IEEE 6th International Conference on Cloud Computing Technology and Science

Two-level Storage QoS to Manage Performance for Multiple Tenants with Multiple Workloads

Shu Qin Ren, Shibin Cheng, Yu Zhang, En Sheng Lim, Khai Leong Yong
Data Storage Institute
Agency for Science, Technology and Research (A*STAR) Singapore
Email: Ren_Shuqin,Chen_Shibin,Zhang_Yu,Lim_En_Seng,Yong_Khai_Leong@dsi.a-star.edu.sg
Zengxiang Li
Institute of High Performance Computing
Agency for Science, Technology and Research (A*STAR) Singapore
Email: liz@ihpc.a-star.edu.sg

***Abstract*—With more applications moving to cloud, scalable storage systems, composed of a cluster of storage servers and gateways, are deployed as the back-end infrastructure to accommodate high-volume data. In such an environment, it is a challenge to provide predictable and controllable storage performance for multi-tenanted users with multiple applications, due to performance violation from misbehaving applications. In this paper, we propose a two-level QoS controller over scalable storage system. On the higher level, I/O throughput rented by each tenant is guaranteed and strictly limited by a CAP value. On the lower level, this rented service can be on-demand served among multiple applications under the same tenant. Thus our distributed controller not only shields performance violation from "noisy" tenants but also allows tenants to fully utilizing the rented I/O throughput. Furthermore, the QoS controller is implemented in an efficient manner, by reusing the communication channels among gateways and storage servers and piggybacking control signals on data communications. The experimental results have shown that the two-level QoS controller can guarantee I/O throughput at tenant level by controlling the CAP value while accelerating applications by on-demand serving at a very little computation cost.**

Nowadays, Big data applications has been widely used in every area of science, commerce, and society [1] [2]. The big data applications are usually time-consuming, requiring huge amount of compute, storage and network resource. Since cloud computing offers a way to increase IT capacity on the fly, more and more interest has been shown to execute big data applications on the Cloud [3]. In order to accommodate high-volume data, scalable storage systems are deployed as the back-end infrastructure in Cloud. Fig.1 illustrates a general scalable storage system. This scalable storage system, composes of multiple storage nodes as servers and gateways, providing data service for multiple tenants with multiple applications. However, a tenants data service quality could be disrupted by noisy neighbors who are misbehaving or aggressive tenants. Thus an efficient quality of service (QoS) control is highly demanded in such a multi-tenant Cloud environment to exclude performance interference among these tenants. The other concern is how the tenants can fully utilize their services for which they paid. If multiple applications belonged to same tenant can share the data service level, they will be accelerated and obtain best service utilization.

QoS has been well-studied for shared network resource. A variety of mechanisms have been used in QoS network protocols, such as differentiated services [4] [5], resource reservation protocol [6], integrated services [7], and so on. Recently QoS for storage system has been considered more important to manage performance in the Cloud environment. Similar to the differentiated services in network, differentiated storage can be referred in [8], [9]. But these solutions have not tackled the performance interferences from malicious users. Similar to resource reservation in network, multi-tier storage system [10] consisting of different types of hard disk drives as SSDs, SAS and SATA drives, was once adopted by storage vendors as a straightforward method to provide differentiated I/O performance. This solution neither tackles the performance interference among applications accessing same tier of storage, nor concerns about whether the tenant has fully utilize his/her paid service.

To cope with classes of data service in cloud environment, a good QoS design has to provide performance differentiation, service granularity and cost efficiency. We worked on differentiated QoS-aware distributed storage system in our previous work [11] to provide proportional I/O service for different tenants. The proportional QoS storage is enforced in a distributed manner by collaborating among multiple nodes, and is very efficient by piggybacking control signal over data path. The nearly real-time response results triggered us for quantitative QoS for a distributed storage system, which will be elaborated in this paper.

This paper mainly tackles performance interference and service utilization in a multi-tenant Cloud environment by enforcing a two-level QoS storage controller. Three contributions are from our work:

1) Distributed token buckets coordinate for global QoS controlling.
2) On-demand I/O service provides a tenant better service utilization by allowing his multiple applications share service.
3) A practical solution for multi-tenant cloud environment, with low overhead and delay.

This paper is organized as following, Section I presents the related works on QoS control mechanism and existing solutions for distributed storage. Section II describes our proposed two-level QoS controlling over distributed storage.

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.56

Section III evaluates the performance and feasibility of our proposed QoS control. Section IV concludes our work.

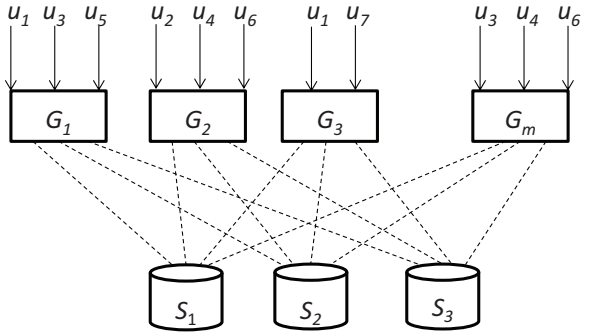


Fig. 1: Scale-out Storage System with Distributed File System

I. Related Work

Conventional QoS Control Mechanisms. Scheduling algorithm plays a critical role for service differentiation in a large-scale shared resource system. Plenty of prior works have been undergone. In a packet switching network, Round-Robin based algorithms [12], [13] and virtual time based algorithms [14] are proposed for hierarchical link sharing in packet network. Some analytical works on packet network scheduling [15], [16] discuss the fairness, deviation of delay and computation complexity of the algorithms. Similar scheduling was also applied for differentiated service quality in network storage system using adaptation of Start-time Fair Queuing (SFQ) [17].

QoS control for Distributed Storage System. A plenty of service quality control mechanisms are proposed for distributed storage systems either in centralized or distributed manner. The centralized QoS approaches, such as Pclock, Horizon, Fahrrad and Faade [18], [19], [20], [21], tickles the disk scheduling to satisfy the delay bound. However these approaches limit the system scalability, not applicable to the current big scale shared storage system. Other distributed QoS approaches [22], [23], [24], [9] provide quality of service of distributed storage system by employing multiple schedulers. But only [9] [9] is scheduling with global differentiated service while the other three are just provide local fairness without global information concerned. Thus the scheduling without global information can be easily violated by malicious users.

Similar to [9], we are working on the quality of service for a distributed storage system in a distributed scheduling manner. Differently we are targeting guaranteed service instead of differentiated service, concerning service utilization and global fairness.

II. Two-level Storage QoS to Manage Performance for Multiple Tenants with Multiple Workloads

A. Scheduling Issues Addressed for Distributed Storage System

A distributed storage system, as shown in Fig.1, is composed of multiple storage nodes as gateways and servers to accommodate the big volume of data, where tenants applications access I/Os through attached gateways. A tenant can run same application through single or multiple gateways and he also can run different applications either through single or multiple gateways. To provide a predictable I/O performance in such a multi-tenant cloud environment, three important issues are critical to address:

TABLE I: Notations Used in Two-level Storage QoS for Multi-tenant Environment

Symbol	Description
u_i	the i-th tenant
l_i	the i-th tenant's service level(MBps)
G_j	the j-th gateway
S_k	the k-th storage server node
D_{ij}	the i-th tenants I/O demands at gateway G_j
TB_{ij}	the token bucket for the i-th tenant at gateway G_j
bkj	the basket sent to gateway G_j from server S_k
$< u_i, G_j, D_{ij} >$	3-tuple to indicate the I/O demand from Ui at gateway Gj

Avoid a tenants performance interference from other tenants. It is managing performance interference to isolate the global service a tenant can obtain. In such a complex circumstances that a tenant may attach to multiple gateways, these gateways coordinate each other to control and ensure a tenants total I/O as the first level control. The details can be referred from section II-B.

Better service utilization. It is to provide on-demand service sharing among a tenants applications to accelerate their executions. An adaptive service control at tenant level is proposed as the second level control, as detailed in section II-B

Minimizing performance overhead introduced by QoS control. We propose a mutualistic piggyback algorithm to let tenants help each other piggyback necessary control information on data path, as detailed in section II-C and II-D

Notations of this article are listed as Table I.

B. Two Level QoS Control – Adaptive Rate Limiters using Distributed Token Buckets

The token bucket algorithm is widely used in network environment to determine traffic transmission timing for complying with the bandwidth and burst. To determine the service level of our token bucket, we use the measurements from throughput instead of bandwidth. Our token bucket will generate the token dynamically and will affect the I/O service rate together with the incoming traffic; with a valid token and an incoming traffic, a specific throughput can be limited. To provide a global and fair service for multiple nodes such as cloud environment, we enforce a CAP value with distributed token bucket algorithm as the first level control. To provide adaptive service utilization, we introduce an adaptive rate limiter to share tokens among these token buckets as the second level control.

Distributed Token Bucket. A token bucket has two parameters: average rate to generate token at a fixed interval and burst size as the maximum tokens or bucket capacity. We set throughput as a performance measurement to provide different service level by setting different token average rate. We set maximum throughput by setting bucket capacity.

Each tenant has a distributed token bucket on the gateway where it goes through. When the tenant are served across multiple gateways, the distributed token buckets which works individually work together to behave as a global token bucket for the tenant. A tenants global service level thus is ensured by these distributed token buckets. The procedures of a distributed token bucket algorithm are diagrammed as Fig.2 As shown in Fig.2, a tenant u_1 has two different applications, app_1 and app_2 attached two gateways, and his service is controlled by setting the token average rate of the corresponding token buckets TB_{11} and TB_{12}. These average token rates need to be set to allow total throughput from the gateways. In case that app_1 had more I/O requests than app_2, if app_1 could borrow the spared service from app_2 , app_1 would get accelerated and tenant u_1 would get better service utilization. Thus adaptive rate limiter is needed to work with the distributed token bucket.

Adaptive Rate Limiters. Assume the i-th tenant u_i subscribed service rate as l_i and he attached k gateways $G_1, G_2, , G_k$ and his I/O requests at the gateway G_j is D_{ij}. If each of these k gateways knew the others I/O requests, it could calculate the total requests of tenant u_i by adding up as equation 1. Thus the gateway G_j can schedule its local service proportionally according to its local requests share as equation 2. Such an adaptive and distributed rate limiters not only ensure global service at tenant level, but also provide better service usage for tenant. But these adaptive rate limiters only work when the gateways know others I/O requests on other gateways. How can the system allow gateways collect the demand of requests on others in an efficient yet low latency approach? Section II-C gives a lightweight and collective method to pass over these information among gateways.

$$D_i = \sum_{j=1}^{k} D_{ij}, u_i \rightarrow G_j \tag{1}$$

$$S_{ik} = \frac{D_{ik}}{D_i} * l_i. \tag{2}$$

C. Mutualistic Piggyback for Real Time QoS Control on Distributed Storage System

In order to adaptively schedule a tenants service cross multiple attached gateways, the I/O demands have to be transmitted among these gateways. In order to schedule different I/Os according to their subscribed service levels, control signals are labelled at gateways to identify the different I/Os and indicate the incoming demands, detonated by a 3-tuple $< u_i, G_k, D_{ik} >$. Logically these labels flow along signaling path while I/Os go along data path, as illustrated in Fig.3. But it would bring in an extra communication channel and complexity to collect labels through the signaling path. Here we proposed a lightweight labels distribution method by making use of existing data path between gateways and storage nodes. We make use of I/O requests to piggyback labels from a gateway to a storage server and I/O replies to piggyback and forward the labels to other gateways. Upon receiving the labels from other gateways, a gateway aggregates these labels and calculates global I/O demands. Thus an eco-signal distribution system is formed without extra channel and components.

To enhance the control efficiency, two observations are addressed here: 1) a tenants labeled demands are not necessary to be broadcasted to all the gateways from a storage server, but only necessary to the gateways he attached; 2) when a tenants I/O replies from a storage server to a gateway, it can not only piggybacking his own labeled demands but also other tenants labeled demand.

To intelligently choose whose signals and where to carry, we design a user-gateway connection map to assist the mutualistic piggyback, as shown in Fig.3. In the connection map table, each row of the map table is a gateway vector G_i to indicate the users connected to the i-th gateway and each column is a user vector U_j to indicate the gateways the j-th user attached to. This map is only updated when new users joined or existing users quited the service and it is relatively stable. Storage servers use a basket to contain the labels to a gateway, the m-th basket b_{km} on storage server S_k is used to collect all the labels piggybacked to gateway G_m. And whenever there is an I/O reply back to gateway G_m, the whole basket of labels will be piggybacked.

In our system, we use a counter to update the i-th tenants I/O demands on the j-th gateway, denoted as D_{ij}. And whenever his I/O request going to a storage server S_k, it piggybacks the 3-tuple label $< u_i, G_j, D_{ij} >$. Upon receiving the I/O requests and the piggybacked label, storage server will check the user-gateway connection map to find out what else gateways user u_i is attached, for example G_m, G_n, and it will open the corresponding baskets b_m, b_n and makes a copy of the label put it into each of these baskets. Whenever there is I/O reply from storage server Sk to gateway G_m, the whole basket b_m will be piggybacked to gateway. Upon receiving the labels from different storage servers, a gateway aggregates and obtains the total I/O demands and schedule its local I/Os with token bucket algorithm accordingly as equation 2. One thing to note is that a tenants global service is guaranteed only on the condition that his group of token buckets to enforce the local services at the same point. Thus a synchronous schedule point has to be defined across multiple gateways. We propose a synchronous mechanism using relative time stamps in section II-D.

D. Synchronous Scheduling using Relative Time Stamps

As the above formulated data limiting, I/O requests could be dynamically scheduled with the adaptive token buckets if these schedulers became effective at the same point for stable service level. Without data limiting, some users may experience different global services from their subscription, either higher or lower than promised. As the arrival time of I/O requests and replies are not predicable, we have set a reference time stamp to synchronize the time to make schedule effective. As system time on storage servers can be synchronized and we set epochs at a fixed interval. The beginning point of next epoch is the reference time to activate new schedule. I/O requests are sent to storage servers for I/O replies. When I/O replies back to clients, the time offset from service point to the begging point of next epoch is tagged as timestamp T_{off} and attached to the I/O reply. This time offset is tagged at tenant level, after obtaining time stamps attached, clients choose the first received tag to decide the next schedule effective point for this tenant U_m by adding his first T_{off} to current time. Thus, the new schedules of multiple token buckets can take effective at the same point by making use of these relative time stamps,

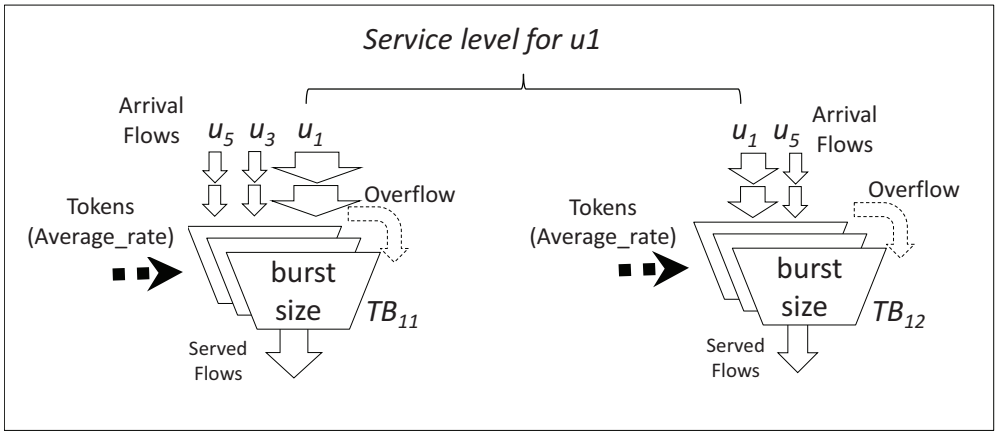
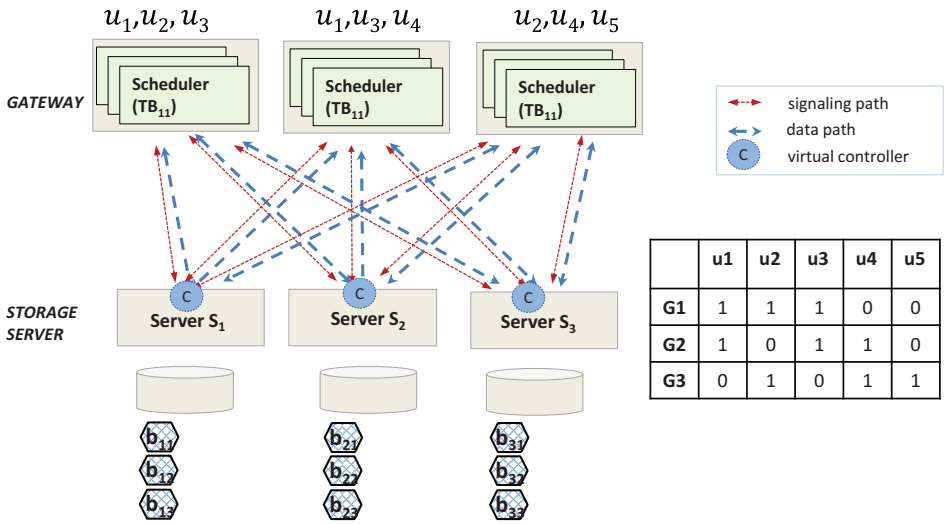


Fig. 2: Distributed Token Bucket to Provide global service at user level



	u1	u2	u3	u4	u5
G1	1	1	1	0	0
G2	1	0	1	1	0
G3	0	1	0	1	1

Fig. 3: Low Latency and Distributed Scheduling with Piggyback

which ensures a fair and persistent quality of service for each tenant.

III. Experiment Performance and Feasibility Evaluation

A. Experiment Setup

We evaluate the proposed Two-level QoS Storage Management with multiple tenants and multiple workloads. The distributed storage system is built over GlusterFS [25], for its scale-out architecture and good I/O performance. As proposed scheme, QoS control signals are piggybacked on the data I/O. Thus we observed the I/O flow in GlusterFS and add control modules on gateways (Gluster clients) and storage servers (Gluster servers). The modules on gateways and servers function as Table II.

We built a platform with two gateways (Gluster clients) and two servers, which are connected through a switch. Both the servers are configured with Dual Core AMD Opteron(tm) Processor 270 1.995 GHz, 1024 KB cache, 3 GB RAM and 1 Gbps NIC. All the gateways are equipped with Intel(R) Core(TM)2 Duo CPU E6550 2.33 GHz, 4096 KB cache, 2 GB RAM and 1 Gbps NIC. All the equipment in the System are running Ubuntu server 12.04 LTS, Linux kernel 3.2.0. We use IOzone [26] to generate workload, high volume of I/O requests are generated by increasing number of IOzone processes. Each IOzone process access a 1GB file cyclically.

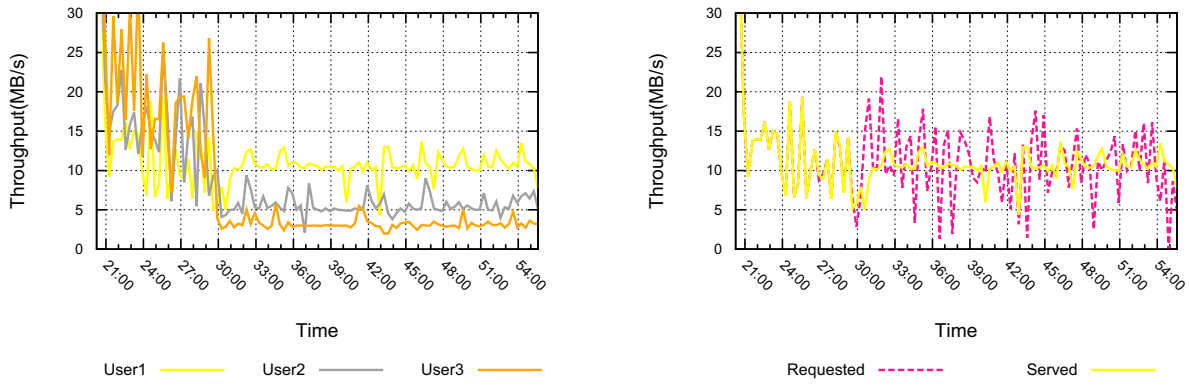
B. Experiment on Distributed QoS Enforcement

We setup the system with 3 tenants ($User_1, User_2 and User_3$) and each tenant is attached to both gateways and has I/O requests through them. The QoS services subscribed by 3 users are 12MBps, 5MBps and 3MBps respectively, their services are marked as such colors as gold, grey and orange respectively.

Fig. 4(a) gives the proposed Two-level QoS enforcement over distributed storage system. Three tenants, $User_1$, $User_2$

TABLE II: QoS Control Modules developed in GlusterFS

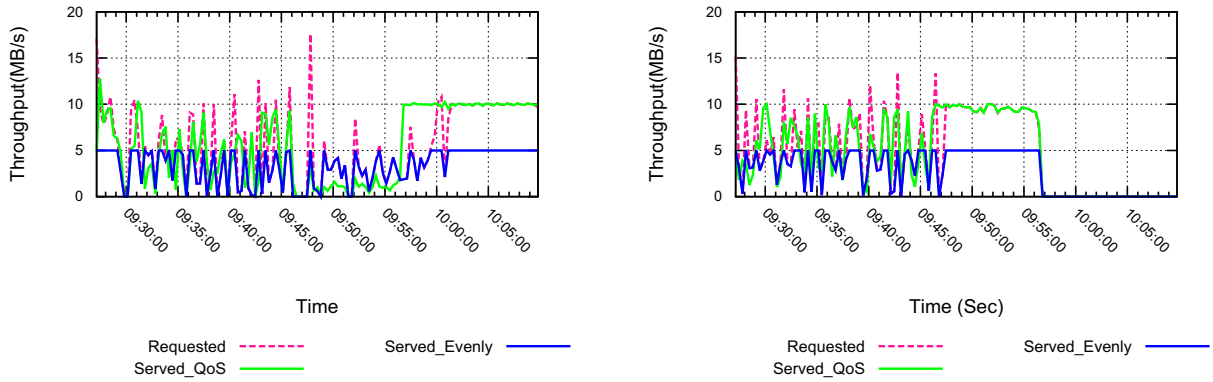
Module Name	Module Functions	Node Name
I/O demand labelling	Label the I/O demands for each individual tenant	Gateway
Shoot basket	Copy and put the labels piggybacked on I/O requests from gateways into the corresponding baskets	Server
Aggregate labels	Aggregate the received labels from servers;Calculate the total I/O demands for each tenant;Scheduling the current I/O accordingly	Gateway
Fetch basket & label	Add time stamp in the label and piggyback the new label on I/O reply to the gateway nodes	Server



(a) Throughput of Different Users

(b) Requested and Served Throughput of One User

Fig. 4: Guaranteed Global Throughput at Tenant Level



(a) Gateway1

(b) Gateway2

Fig. 5: Served Service at Each Gateway for Individual Tenant

and $User_3$ are scheduled accordingly to their service subscribed. Although $User_3$, the low service level user (3MBps), is more aggressive to send requests and gain better service level before scheduling, his benefit is immediately controlled after our QoS scheduling is enforced at 30:00. And the control further reduces his I/O requests coming and provides a stable level of service.

To observe the on-demand scheduling at user level, the total I/O demands and service of $User_1$ are dedicated in Fig. 4(b), where we can see the I/Os are scheduled accordingly to its demands, but controlled to neither over utilize the system I/O bandwidth nor beyond its service subscribed.

C. Adaptive QoS Service Sharing for Tenants Better Service Utilization

This experiment is to verify the proposed scheme can adaptively adjust the service according to the demand to allow tenants best use of service subscribed. And we also observe the benefit from this adaptive scheduling by comparing with the existing distributed scheduling in Ceph [8], which uses equally proportional service rate at each storage node; service will be evenly served at each gateway.

We have 2 tenants of same service level with 10MBps. One tenant $User_3$ has 2 applications (App_1, App_2) attached 2 different gateways (GW_1 and GW_2) and the other tenant

$User_4$ only attach to one gateway (GW_2). Since $User_4$ only attaches to one gateway, the schedule and performance are same for both equally proportional scheduling and adaptive scheduling. We study the scheduling and performance benefits of $User_3$ with his two applications, as shown in Fig.5(a), Fig.5(b) and Fig.6. These two applications have different I/O traffic, with dynamic I/O requests, as shown in Fig.5(a) and 5(b) pink dash lines; their actual served I/O throughput are denoted as the green solid line for our adaptive scheduling and blue solid line for Ceph's evenly distributed scheduling. Obviously the two applications have very different I/O access patterns, app_1 has big spared bandwidth with small I/O demands sometime, its spared bandwidth can be shared with app_2 using out adaptive scheduling; but this spared bandwidth can not be shared with app_2using the equally proportional scheduling. Thus, tenant could gain better service utilization using our adaptive scheduling.

To measure service utilization at tenant level, we add up actual distributed service (throughput in our system) for each tenant. Fig.6 gives the total service the tenant $User_3$ gained with our adaptive scheduling and the evenly distributed schedules. The Green line shows the total service throughput of $User_3$ using our adaptive scheduling and the red line is the total throughput using evenly served scheduling. Its obvious to see that our adaptive scheduling can provide a tenant better service utilization, thus same batch of tasks could be processed in shorter period.

IV. Conclusion

This paper proposes two-level distributed storage QoS to manage I/O performance for multiple tenants with multiple workloads. The first level is to address performance interference by introducing a fair global guaranteed I/O services to limit service within service range. The second level is to address better service utilization by sharing spared service among a tenants application.

In terms of system deployment, some optimizations have been exercised to reduce overhead introduced from the QoS controlling. First we use I/O data path to piggyback control signal without introducing extra communication channel. Secondly we use relative timestamp to synchronize multiple nodes to activate schedule at the same time in a loose manner. And the experiment results show that our distributed and on-demand QoS mechanism is suitable for big-scale multi-tenant performance management for its scale-out architecture and efficient implementation. Large-scale test bed with storage nodes will be accessed by thousands of tenants for scalability verification in our future work.

Acknowledgment

The authors would like to thank Singapore Agency for Science, Technology and Research (A*STAR) for funding this work under future data center technology thematic strategic research program by with grant number 112 172 0015.). We also would thank student Qiu Li Li from Nanyang Technological University who did excellent work to assist with the experiment tests.

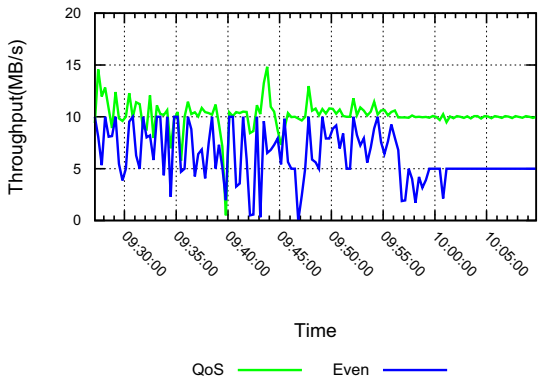


Fig. 6: Total Served Service at Multiple Gateways for Individual Tenant

References

[1] H. Chen, R. H. L. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact," *MIS Quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012.

[2] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health Information Science and Systems*, vol. 2, no. 3, 2014.

[3] A. Divyakant, D. Sudipto, and E. A. Amr, "Big data and cloud computing: Current state and future opportunities," in *Procs of International Conference on Extending Database Technology (ICDT '11)*, 2011, pp. 530–533.

[4] C. DiffServ. Differentiated services (diffserv). Online. [Online]. Available: www.cisco.com/en/US/technologies/tk543/tk766/technologies_white_paper09186a00800a3e2f.html

[5] Cisco. Differentiated services (diffserv). Online. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfdfsrv.html

[6] C. RSVP. Resource reservation protocol (rsvp). Online. [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/resource-reservation-protocol-rsvp/index.html

[7] RFC. Integrated services (intserv). Online. [Online]. Available: http://tools.ietf.org/html/rfc2998

[8] S. A. Weil, S. A. Brandt, E. L. Miller, and D. D. E. Long, "Ceph: A scalable, high-performance distributed file system," in *Procs of Symposium on Operating systems design and implementation (OSDI?6)*, 2006, pp. 307–320.

[9] Y. Wang and A. Merchant, "Proportion-share scheduling for distributed storage system," in *Procs of Conference on File and Storage Technologies (FAST'07)*, 2007.

[10] A. Elnably, H. Wang, A. Gulati, and P. Varman, "Efficient qos for multi-tiered storage systems," in *Procs of Conference on Hot Topics in Storage and File Systems(HotStorage'12)*, 2012, pp. 6–6.

[11] S. C. B. T. E. L. K. Y. Y Zhang, SQ Ren, "Differcloudstor: Differentiated quality of service for cloud storage," in *Procs of Asia-Pacific magnetic Recording Conference (APMRC'12)*, 2012, pp. 1–9.

[12] E. L. Hahne, "Round-robin scheduling for max-min fairness in data networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 1024–1039, 1991.

[13] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Procs of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'95)*, 1995, pp. 231–242.

[14] P. Goyal, H. M. Vin, and H. Chen, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," in *Procs of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'96)*, 1996, pp. 157–168.

[15] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[16] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, 1989.

[17] W. Jin, J. S. Chase, and J. Kaur, "Interposed proportional sharing for a storage service utility," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 37–48, 2004.

[18] A. Gulati, A. Merchant, and P. J. Varman, "pclock: An arrival curve based approach for qos guarantees in shared storage systems," in *Procs of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'07)*, 2007, pp. 13–24.

[19] A. Povzner, D. Sawyer, and S. Brandt, "Horizon: Efficient deadline-driven disk i/o management for distributed storage systems," in *Procs of Symposium on High Performance Distributed Computing (HPDC'10)*, 2010, pp. 1–12.

[20] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn, "Efficient guaranteed disk request scheduling with fahrrad," in *Procs of European Conference on Computer Systems (Eurosys'08)*, 2008, pp. 13–25.

[21] C. R. Lumb, A. Merchant, and G. A. Alvarez, "Façade: Virtual storage devices with performance guarantees," in *Procs of Conference on File and Storage Technologies (FAST'03)*, 2003, pp. 131–144.

[22] J. C. Wu and S. A. Brandt, "Providing quality of service support in object-based file system," in *Procs of Conference on Mass Storage Systems and Technologies (MSST'07)*, 2007, pp. 157–170.

[23] A. Gulati, I. Ahmad, and C. A. Waldspurger, "Parda: Proportional allocation of resources for distributed storage access," in *Procs of Conference on File and Storage Technologies (FAST'09)*, 2009, pp. 85–98.

[24] L. Huang, G. Peng, and T.-c. Chiueh, "Multi-dimensional storage virtualization," in *Procs of Conference on Measurement and Modeling of Computer Systems(SIGMETRICS'04)*, 2004, pp. 14–24.

[25] Gluster. Glusterfs. Online. [Online]. Available: http://www.gluster.org/

[26] Iozone. Iozone. Online. [Online]. Available: http://www.iozone.org/