Workload Adaptive I/O Fairness Scheme for Modern Cloud Storage

KiSung Jin, SangMin Lee, HongYeon Kim, YoungKyun Kim
Department of High Performance Computing Research, SW contents Laboratory
Electronics and Telecommunications Research Institute
Daejeon, Korea
e-mail: {ksjin, sanglee, kimhy, kimyoung}@etri.re.kr

***Abstract*—Although many cloud services have introduced several algorithms for providing Quality of Service (QoS) satisfaction to users, the performance interference problem among services has not yet been solved. Because multiple heterogeneous services produce non-deterministic workloads in the real world, service providers often experience contradictory results in their quality prediction. To solve this phenomenon, we specifically focus on the storage layer among the entire cloud stack. In this paper, we propose a workload adaptive Input/Output (I/O) fairness scheme to guarantee balanced data access regardless of various service workloads. Furthermore, we validate our idea through performance evaluations and show that our algorithm can satisfy QoS requirements in the cloud service.**

Keywords-Cloud; Cloud Storage; I/O Fairness; QoS

I. INTRODUCTION

The cloud platform provides large pools of computations and storage resources to heterogeneous services on demand. An increasing number of services are already moving their workloads to cloud platforms. Many of commercial or open platforms, such as Azure [1], Amazon [2], Hadoop [3], Openstack [4] and Cloudstack [5] represent the beginning of a much larger trend. For example, Amazon's Elastic Compute Cloud (EC2) provides practically infinite resources to anyone willing to pay. Following this trend, Gartner estimated that the annual growth rate of cloud services will reach about 16% by 2018 [6].

However, most existing systems still provide weak performance isolation or simple fairness control techniques among multiple services. High-demand or misbehaving services can overload shared resources as well as can disrupt other well planned services. In particular, if one of services gives rise to massive I/O workloads unexpectedly, the remaining services will suffer from poor service quality due to bottlenecks. Such a performance violation implies that paying for quantity of resources does not necessarily mean the user will receive the desired QoS level. This is a key problem, which prevents more services to move to the cloud platform.

By carefully observing this performance violation problem in the cloud platform, we find that the key reason for the poor service quality is mainly due to the I/O interference on the shared storage. Many researches have been still trying to find their solutions in upper layers, such as the application, the server, and the network. However, this approach is regarded as an easy and simple way, but other serious problems still remain. For example, lots of developers can easily add some QoS optimizations to the application layer. In the start-up phase, these approaches may give service providers an illusion that they can control. However, as an increasing number of services are gradually producing data explosions and storage bottlenecks, they would realize that the cloud platform finally reaches a limit that they cannot control. Even if the server and the network layer provide us resource isolation methods with virtualization, they cannot avoid I/O interference problems on the storage.

In this paper, we propose a workload adaptive I/O fairness scheme that controls access fairness among all heterogeneous services sharing the cloud storage. Our algorithm continuously collects workload status from all services, and automatically controls each access ratio to guarantee the I/O fairness. It always guarantees balanced I/O performance regardless of various service workloads. Our model is meaningful in that the quality of service is guaranteed on storage level rather than upper layers, such as the application, the server, and the network. While traditional approaches on upper layers continuously require resource reconfiguration or service optimizations, our model can always assure the QoS among all services based on real I/O workloads.

This paper starts with an overview of QoS problems on the traditional cloud platform in Section 2. Then, we present our proposed model and algorithms to guarantee the quality of service for multiple heterogeneous services in Section 3. In Section 4, we show the superiority of our algorithms through performance evaluation and continue with conclusions in Section 5.

II. RELATED WORKS

Recently, there are lots of heterogeneous applications simultaneously working in the cloud platform. Under this environment, because all services try to use the system resources in the best-effort manner, inevitable performance violations can severely degrade the service quality. If one of services overuses the system resources, the remaining services may suffer from the relatively poor performance. Although many researches try to solve this problem, it is very hard to find complete solutions yet. To find the fundamental reason of this problem, we review current activities followed by commonly used cloud architecture.

 ISBN: 978-1-61208-570-8

TABLE I. COMPARISONS OF CURRENT QOS APPROACHES IN CLOUD LAYERS

	Application	Server	Network	Storage
Isolation Object	Service level Bandwidth	Computing Resource	Communication Resource	Physical Storage Device
Isolation Method	Service Optimization	Server Virtualization	Network Virtualization	Partitioning Storage Device
Avoiding I/O Interference	Difficult	Difficult	Difficult	Easy
Continuous Optimization	High	High	Middle	Middle
Technical Maturity	High	High	High	Low

In the SNIA [7], the cloud environment consists of 4 layers: the application, the server, the network, and the storage. We analyzed the role of each layer, as well as the techniques for ensuring QoS at each layer.

Application Layer provides a service logic to end users. Under the scalability feature of the cloud platform, a service provider can add a new service to the application layer at any time. Traditionally, many major cloud computing vendors, such as Amazon [2], Windows Azure [1], Google App Engine [8] provide "pay-per-use fixed pricing" or "pay for resources" model. While they guarantee the minimum rates of the user contract, they do not provide system wide fairness because they assume uniform load distributions across tenant partitions. Hadoop [3] supports resource management scheme for MapReduce framework running on Hadoop Distributed File System (HDFS). It is designed to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster. Choosy [9] provides Constraint Max-Min Fairness (CMMF) by generalizing previous max-min fairness scheme to handle hard task placement constraints. However, an unpredicted workload pattern caused by multiple heterogeneous applications often confuses the service provider. Even if the service provider can estimate the individual service workload, the interference among multiple services can degrade the overall service qualities.

Server Layer provides computing platforms to applications. In this layer, a recent trend is to use a server virtualization technique, which encapsulates workloads in virtual machines (VMs) and consolidates them on multicore servers. In order to maximize the resource utilization of shared resources, hardware extensions such as caches have been considered extensively in previous work. For example, hardware based control schemes have been proposed dynamically partition cache resources based upon utility metrics [10] or integrate novel data control policies to pseudo-partition caches [11]. Even though resources are sliced and allocated to different VMs, they are still shared and interfere with each other without constraints. The isolation across VMs provided by hypervisors rather amplifies the performance issues demonstrated by several works [12] [13] [14].

Network Layer provides communications between each layer. Recently, as the cloud service is emerging, the virtual private network (VPN) is becoming an important factor for service providers. VPN provides customers with predictable and secure network connections over a shared network. Because the network is essential to organize distributed computing, many optimization techniques have been introduced for a long time. Hose model [15] allows for greater flexibility since it permits traffic to and from an endpoint to be arbitrarily distributed to other endpoints. VMware provides the vNetwork Distributed Switch that combines all virtual switches into one logical centrally managed unit. As an open software cloud platform, OpenStack [4] and CloudStack [5] add the virtual network functionality into their software stack. However, the network virtualization still cannot solve the storage interference problem caused by multiple applications. Even if we configure the well planned network topology, it cannot avoid the storage bottleneck.

Storage Layer manages storing data produced by applications. The storage layer faces different challenges than sharing resources at upper layers. Rather than managing individual storage partitions, the storage layer wants to treat the entire storage system as a single black box. All of the applications share their data on the virtualized storage. While many studies provide differentiated service to workloads accessing a single storage array, their techniques are not suitable for cloud storage but rather a centralized one [16]. Pisces [17] provides per-tenant performance isolation and fairness in shared key-value storage. A server-side I/O coordination scheme is introduced in [18]. However, although some algorithms and models are trying to satisfy storage level QoS for a quite long time, it is relatively hard to find practical solutions comparing to other layer in the cloud.

In Table 1, we summarize the status of current approaches from the perspective of the global QoS control in the cloud service. According to our observations, even if most of cloud layers do their best to control the QoS in their own way, they still cannot avoid I/O interferences caused by multiple simultaneous workloads. Furthermore, they still have an unavoidable side effect, which requires continuous optimizations whenever the service scale or the I/O workload is changed.

III. WORKLOAD ADAPTIVE I/O FAIRNESS SCHEME

To overcome the problems described in Section 1 and Section 2, we propose a purely storage oriented QoS model called the Global QoS algorithm. It always guarantees balanced I/O performance regardless of service workloads. For example, let us suppose there are different types of services running on the storage. If one of the services instantaneously produces unpredicted bursty workloads, it will require to consume more storage bandwidth. Because resources are limited on given hardware configuration, the rest of the services will suffer from the lack of resources. This can degrade overall service qualities and can lead to the

 ISBN: 978-1-61208-570-8

unfairness problem among services. Our scheme automatically controls each access ratio to guarantee the I/O fairness.

A. System Architecture

In this section, we suppose a cloud storage architecture composed of three types of nodes; a storage node, a controller node, and a client node, as shown in Figure 1. The storage node is responsible for storing and retrieving the data produced by all applications. In the cloud storage, lots of storage nodes are connected by a network and provide a single virtualized space to applications. The controller node manages all of nodes participating in the cloud storage as well as monitors overall resource status, such as the storage usage, the network bandwidth, and the resource health. The client node helps applications to access their data over the cloud storage communicating with the controller node and storage nodes. Based on this traditional storage architecture, we add new modules to it to guarantee a balanced QoS level to all heterogeneous services.

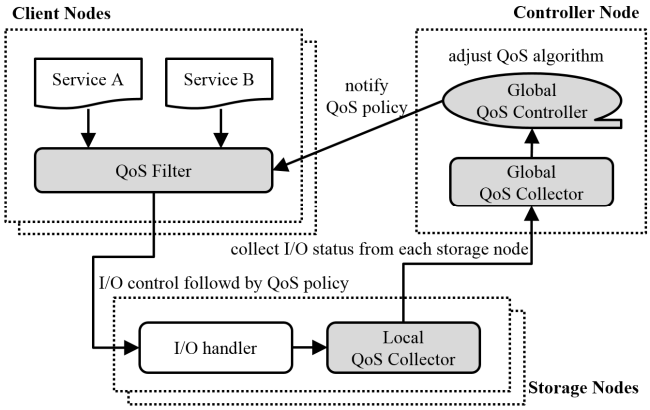

Figure 1. System Architecture

Local QoS Collector collects the local I/O history of all services in each node and forwards them to the controller node. The local I/O history can be collected by the I/O handler, which has a role of storing and retrieving an application data in the local storage media. Whenever the client node requests the data, the I/O handler notifies an access to information to the Local QoS Collector. There are two types of information in the local I/O history. One is the amount of access usage identified by each service and the other is each usage ratio of reading and writing within a service. All collected local I/O history is refreshed after notifying to the controller node in a specific period. One consideration point in notifying phase is how to decide the time interval to notify the local I/O history to the controller node. If the time interval is too short, it can give a burden to the system. On the contrary, if the time interval is too long, it becomes insensitive to workload changes so that a delayed QoS control is inevitable. However, because the time period does not undermine the basic functionality of our algorithm, we leave it to the tunable parameter.

Global QoS Collector manages a global I/O history collected from all storage nodes. That is, the global I/O history is the systemwide workload information which is merged with local I/O histories collected from storage nodes. After receiving the local I/O history from the local QoS collector, the global QoS collector classifies it by each service.

Global QoS Controller plays an important role to decide a policy for QoS control by using our algorithm. The global QoS controller analyzes system-wide workload information based on the global I/O history and decides a proper policy by using our Global QoS algorithm. For this, the Global QoS algorithm compares I/O workloads of each service to estimate the current I/O fairness level, such as the I/O skewness or the I/O starvation. After that, the Global QoS algorithm decides a policy for all services. If one of services is producing relatively bursty workload, a negative QoS policy is applied to that service. The negative QoS policy means that it tries to throttle overloaded workloads to an average access rate. Finally, determined policies are delivered to the client node to control the access behavior of the application.

QoS Filter is running on the client node and manages application access pattern based on policies decided by the global QoS controller. If one of the applications gets the negative policy, the QoS filter throttles the access speed of that application so as to keep all services to experience fair access quality.

B. Global QoS Algorithm

In this section, we explain the Global QoS algorithm to solve the I/O interference problem among multiple heterogeneous services in the cloud storage. The main principle of our algorithm is to control applications to use resources evenly within the uppermost system performance, which is dynamically renewed over time. We describe the algorithm in detail, as follows:

$$IO_{MAX} = MAX\left(IO_{MAX}(current), \sum_{s=1}^{s=n} HIST_{AVG}(T)\right) \quad (1)$$

We define the IO_{MAX} as the uppermost system performance in current hardware configuration. The service provider does not need to calculate the allowable performance value in their system, because our algorithm automatically updates the up-to-date IO_{MAX} by using real application workloads in (1). Under this equation, even if a new node is added to the cloud storage, our algorithm can update the IO_{MAX} for a new system configuration. For this, we use the $HIST_{AVG}$, which is the average access workload of each service for a period of T. The sum of all service's $HIST_{AVG}$ implies the total average usage of the cloud storage. Then, we compare the $HIST_{AVG}(T)$ with the current IO_{MAX}. If the current IO_{MAX} is less than the $HIST_{AVG}(T)$, we change the new IO_{MAX} with the calculated $HIST_{AVG}(T)$.

$$S_A = \left(\frac{IO_{MAX}}{Number\ of\ Services}\right) \quad (2)$$

The S_A(Allocation for a Service) means an allocated I/O quota for each service. The S_A can be calculated by dividing

the IO_{MAX} to the number of services, as in (2). In our algorithm, every QoS policy is affected by the S_A value.

$$S_C = \left\{ \left(\frac{S_U - S_A}{S_A} \right) \times T \right\} \times \mathcal{M} \quad (3)$$

In (3), the S_U(Used value of a Service) means the amount used by the service in time T. The S_C(Control value for a Service) is used for controlling the service, which uses the storage resource excessively. Comparing the S_U with S_A, we decide a policy to perform the QoS control. If the S_U is less than the S_A, we provide a best-effort policy to allow full data access to that service. On the other hand, if the service usage exceeds the S_A, we consider that the QoS control is required to that service. In that case, we provide a negative policy to that service so that the global service fairness is guaranteed. For example, let us suppose that our algorithm allows services to use the cloud storage at a speed of 100MBps. If a service uses storage resources at a speed of 120MBps, we control the service to access data 20MBps slower. In addition, the $\mathcal{M}$ is used as a moderator to avoid fluctuation of the I/O performance caused by too sensitive access control. Using the $\mathcal{M}$, a smoother quality control is possible.

Procedure I : Calculating QoS Parameters from Real Workloads

```
1:  Variables: current IO_MAX, HIST_AVG(T) for each service
2:  Location: the Controller Node
3:
4:  Procedure WORKLOAD_CALCULATOR
5:    /* get sum of each service workloads */
6:    for each service 1 ~ N
7:      add HIST_AVG(T) ← "average I/O usage for each service"
8:    end loop
9:
10:   /* get average workload value for all services */
11:   HIST_AVG(T) ← ( HIST_AVG(T) / N )
12:
13:   /* get IO_MAX from real workloads */
14:   IO_MAX ← MAX(IO_MAX(current), HIST_AVG(T))
15:
16:   /* get S_A which is an allocated quota for a service */
17:   S_A ← IO_MAX / N
18: End Procedure
```

Figure 2. Caculate QoS Prameters

Procedure II : Determine QoS Policy

```
1:  Variables: S_A, S_U, T
2:  Location: the Controller Node
3:
4:  Procedure POLICY_GENERATOR
5:    /* get S_C to determine the policy for this service */
6:    S_C ← (( S_U - S_A ) / S_A) x T
7:
8:    /* adjust S_C by using Moderator Constant */
9:    S_C ← S_C x M
10:
11:   /* get I/O Policy for this service */
12:   if S_U < S_A then
13:     SET "best-effort" policy
14:   else if S_U < S_A then
15:     SET "negative" policy
16:   end if
17: End Procedure
```

Figure 3. Determine the QoS Policy

Procedure III : Throttling Each Service's I/O Activities

```
1:  Variables: I/O Policy, S_C
2:  Location: the Client Node
3:
4:  Procedure WORKLOAD_CONTROLLER
5:    /* throttle each I/O by using the determined policy */
6:    if Policy = "best-effort" then
7:      return
8:    else if Policy = "negative" then
9:      delay I/O request by using S_C value
10:   end if
11: End Procedure
12:
13: Procedure every read() and write()
14:   /* control I/O action for this service */
15:   call WORKLOAD_CONTROLLER
16:
17:   /* do actual I/O process */
18:   call 'read()' or 'write()' to access the data
19: End Procedure
```

Figure 4. Control the Each Service's QoS

Because our algorithm is designed to be suitable to a general cloud storage architecture, it can be easily adapted to most current storage platforms without disturbing their own functionality. For this, we represent our algorithms by using pseudo codes in Figures 2 to 4.

C. Global Weighted-QoS Algorithm

The Global QoS algorithm considers that all services have equal right to access the storage resource. It can successfully distribute the overall storage bandwidth to all services evenly. However, in the real cloud world, there are various service requirements depending on different conditions, such as the service scale, the number of users or the data access pattern. While one may require a small amount of storage bandwidth, the other may want unlimited data access. To reflect this factor to the cloud storage, we provide another scheme called the Global Weighted-QoS algorithm.

$$SW_A = (IO_{MAX} \times W) \quad (4)$$

We define SW_A(Weighted Allocation for a Service) as an allocated I/O quota derived from applying the weight parameter to the IO_{MAX}. Although the weight parameter can be the storage usage ratio, the hard limited value of the I/O usage or any values influencing the storage performance, we regard the weight parameter as a storage usage ratio(%) to simplify the description. Under the weight parameter concept, each service can use the storage resource within the SW_A. The SW_A for a service can be calculated by (4).

$$SW_C = \left\{ \left(\frac{S_U - SW_A}{SW_A} \right) \times T \right\} \times \mathcal{M} \quad (5)$$

In (5), the SW_C is used for controlling the service, which excessively uses the storage resource compared to SW_A. The algorithm flow is very similar to the global QoS algorithm. If the SW_C is greater than the SW_A in time T, we control the access speed of that service.

IV. PERFORMANCE EVALUATIONS

In this section, we discuss the result of performance evaluations to verify our algorithms. Our simulation codes are added to the MAHA-FS [19], which was developed by ETRI in Korea. The MAHA-FS is a large scale distributed cloud storage for supporting high scalability, high reliability and a scaled up performance. MAHA-FS isolates each service by allocating independent volumes to store user data. Each volume provides the replication scheme to guarantee an available service. Currently, the MAHA-FS has been used in lots of real services, such as internet portals, content delivery network (CDN) services, and broadcasting companies. As a representative reference, we have UPlusBox [20], which is the biggest cloud service in Korea. UplusBox have been using 10PB of storage in single silo constructed by the MAHA-FS. As the accumulated storage capacity for all silos reaches about 60PB, MAHA-FS has been recognized as a reliable as well as a practical system.

A. Experimental Setup

Our evaluation environment is shown in Figure 5. This testbed is used for developing and testing of the MAHA-FS. There are 7 racks, each of which has 36-38 nodes respectively and a total of 256 nodes are connected within the cluster. Each node consists of the same hardware specifications: 2.5GHz X3320 CPU, 2GB of memory and 512GB hard disk. All machines run on Linux kernel 2.6.32 and are connected through a gigabit ethernet network. For a network configuration, the Extreme X650-24T runs as a core switch, and it is connected with 7 Extreme X350-24T edge switches.

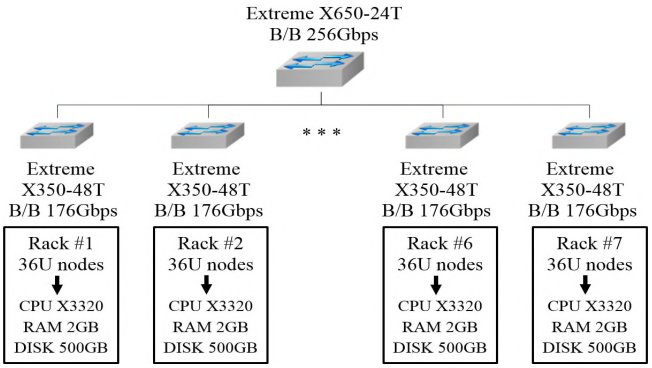

Figure 5. Evaluation Environment

B. Global QoS Control

To verify the Global QoS algorithm to guarantee the fairness for multiple heterogeneous services, we simulate our algorithm by using general Web service workloads. There are 4 services running simultaneously, and all services upload or download random files in the same manner. The size of each file is determined randomly in the range from 600MB to 1.4GB. To generate an imbalance of storage consumptions, we set each service to have different number of users. While the smallest service A has 100 users, the biggest service D has 400 users. Next, we observe before and after the performance transition under applying our algorithm.

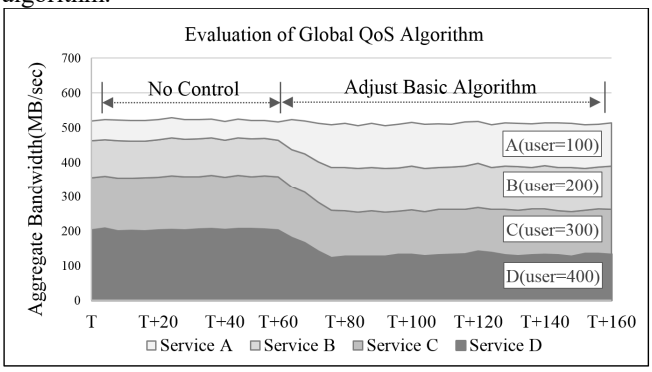

Figure 6. Fairness Result of the Global QoS Algorithm

In Figure 6, the X-axis is a time increased by second and the Y-axis means an occupied bandwidth for each services. Before T+60 without any control, we can see obviously unfair access results depending on the service scale. While service D uses the storage at the speed of 200MB/sec, service A only shows 30MB/sec. However, after applying our algorithm in T+60, each service is changed to have equal resource usage. Our algorithm analyzes a workload status, and lets all services to use a storage resource in fair way. The range from T+60 to T+80 is the intermediate period of adjustment. Our algorithm throttles the bandwidth of excessive services, such as service C and service D. And then, we can see that a sustained fair bandwidth is guaranteed for all services continuously.

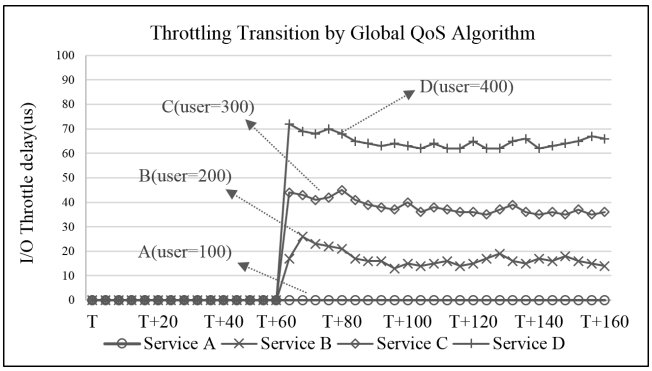

Figure 7. Throttling on the Global QoS Algorithm

Figure 7 shows the I/O throttle transition in the same simulation. Similar to Figure 3, we can see that access to excessive services is controlled after T+60. Especially, the I/O throttle value is increasing in proportion to the service scale. In case of the biggest service D, about 70us of throttle

delay is assigned for every access. Another notable point is the degree of performance fluctuation caused by too sensitive throttling action. However, as can be seen in Figure 4, our algorithm guarantees a sustained QoS control over the whole period.

C. Global Weighted-QoS Control

In this section, we discuss the evaluation result of the Global Weighed-QoS algorithm. Unlike in Section B, we set all services to have the same workload. The whole simulation is performed in three stages; the first is the stage in which services are working without our algorithm, the second is the stage in which our algorithm is applied with a different weight value, and the third stage is the stage in which our algorithm is applied with the same weight value. To satisfy different workload demands, we set the weight value of { 5%, 15%, 30%, 50% } at the second stage and { 25%, 25%, 25%, 25% } at the third stage, respectively. Figure 8 shows the result of our Global Weighted-QoS algorithm. In first stage, all of services share the storage resource in fair way because all services run with the same workload. However, after applying our global weighted-QoS algorithm in T+60, we can see that the QoS control is successfully working.

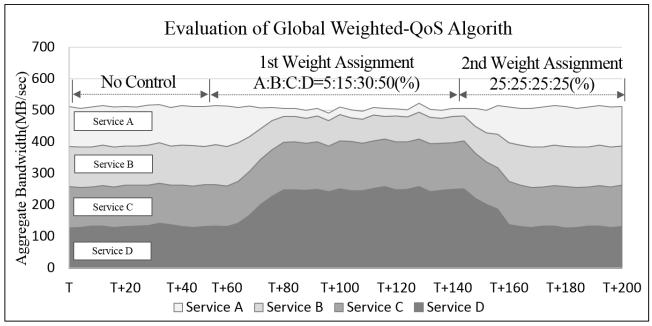

Figure 8. Throttling on the Global QoS Algorithm

Our algorithm analyzes the workload status of all services and automatically controls the QoS level of each service depending on the weight value. Finally, in T+140, all services share the resources in a fair way after applying the same weight value. Because the same weight value means that all services are controlled in a fair way, the result of the third stage is the same as the result of the first stage.

V. CONCLUSIONS

In this paper, we propose a workload adaptive I/O fairness scheme, which supports the global fairness among multiple heterogeneous services. The Global QoS algorithm guarantees balanced I/O performance regardless of service workloads. Our model has two contribution factors. The first is that the quality of service is guaranteed on storage level by using real workloads rather than higher layers, such as the application, the server, and the network. While traditional approaches require continuous optimizations for the cloud platform, our model controls the QoS in itself. The second contributing factor is that our idea has been designed to suit a general cloud storage architecture, and it can be easily adapted to many current storage platforms without disturbing their own functionality.

ACKNOWLEDGMENT

This work was supported by the ICT R&D program of MSIP/IITP, [R7117-16-0232, Development of extreme I/O storage technology for 32Gbps data services]

REFERENCES

[1] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, J. Haridas, "Windows Azure Storage: a highly available cloud storage service with strong consistency," Proceedings of the 23th ACM Symposium on Operating Systems Principles, pp. 143-157, Oct. 2011.

[2] Amazon Elastic Compute. [Online]. Available from: http://aws.amazon.com, Feb. 2017.

[3] K. Shvachko, H. Kuang, S. Radia, R. Chansler, R, "The hadoop distributed file system," IEEE 26th symposium on Mass storage systems and technologies (MSST), pp. 1-10, May. 2010.

[4] Openstack Networking. [Online]. Available from: http://wiki.openstack.org/Quantum. Feb. 2017.

[5] Apache Cloudstack. [Online]. Available from: http://www.cloudstack.org. Feb. 2017.

[6] Gartner 261942, Forecast Analysis: Public Cloud Services,

[7] SNIA, The Storage Networking Industry Association, [Online]. Available from: http://www.snia.org, Feb. 2017.

[8] Google App Engine, [Online]. Available from: https://appengine.google.com, Feb. 2017.

[9] A. Ghodsi, M. Zaharia, S. Shenker, I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," Proceedings of the 8th ACM European Conference on Computer Systems, pp. 365-378, April. 2013.

[10] M. Qureshi and Y. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," Proceedings of the 39th Annual IEEE/ACM International Symposium, pp. 423-432, Dec. 2006.

[11] Y. Xie and G. Loh, "PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches," In ACM SIGARCH Computer Architecture News, pp. 174-183, June. 2009.

[12] R. Nathuji, A. Kansal, A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," In Proceedings of the 5th ACM European conference on Computer systems, pp. 237-250, April. 2010.

[13] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," IEEE 3rd International Conference on Cloud Computing, pp. 51-58, July. 2010.

[14] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, C. Pu, C, "An analysis of performance interference effects in virtual environments,", In Performance Analysis of Systems & Software. ISPASS 2007. IEEE International Symposium, pp. 200-209, April. 2007.

[15] A. Kumar, R. Rastogi, A. Silberschatz, B. Yener, "Algorithms for provisioning virtual private networks in the hose model," IEEE/ACM Transactions on Networking, pp. 565-578, 2002.

[16] M. Wachs, M. Abd-El-Malek, E. Thereska, G. Ganger, "Argon: Performance Insulation for Shared Storage Servers," in FAST, pp. 5-5, Feb. 2007.

[17] D. Shue, M. Freedman, A. Shaikh, "Performance Isolation and Fairness for Multi-Tenant Cloud Storage," In OSDI, pp. 349-362, Oct. 2012.

 ISBN: 978-1-61208-570-8

[18] H. Song, Y. Yin, X. Sun, R. Thakur, S. Lang, "Server-side I/O coordination for parallel file systems," In High Performance Computing, Networking, Storage and Analysis(SC). International Conference on IEEE, pp. 1-11, Nov. 2011.

[19] Y. Kim, D. Kim, H. Kim, Y. Kim, W. Choi, "MAHA-FS: A distributed file system for high performance metadata processing and random IO," KIPS Transactions on Software and Data Engineering, pp. 91-96, 2013.

[20] UPlusBox: the Biggest Cloud Service in Korea, [Online]. Available from: http://uplusbox.co.kr, Feb. 2017.

 ISBN: 978-1-61208-570-8