Design and Implementation of Cloud Storage Flow Access Control Based on Token Bucket Algorithm

Ai Mingzhen[1, a], Yu Wen [2,b]

[1]Beijing Key Laboratory of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunication, Beijing 100876, China

[2] Beijing Key Laboratory of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunication, Beijing 100876, China

[a] aimingzhen1992@163.com, [b] yuwen@bupt.edu.cn

Keywords: Cloud storage, Service interface, Flow control; Token bucket.

Abstract. With the popularity of cloud computing, cloud computing enterprises are launching cloud storage services, which are becoming more and more attention and use as an emerging network storage technology. In the application process of cloud storage, object storage service interface is growing rapidly, so as to ensure stability and reliability, the flow control of object storage service is required. Based on the token bucket algorithm, this paper put forward three kinds of flow access control strategy to cloud storage service interface, which can provide fine-grained flow access control for different user priorities and service interfaces.

1. Introduction

Cloud storage is a new concept that extends and evolves in the concept of cloud computing. It is a new type of network storage technology. Cloud storage is a system that brings together a large number of different types of storage devices in a network to work together through application software to jointly provide data storage and service access [1]. Users can connect to cloud storage at anytime, anywhere, via any connected device and conveniently access the data. The biggest feature of cloud storage is the storage as services, where users can upload their data to the cloud storage via API of the public cloud storage.

Object storage is cloud computing architecture of data storage that is usually used to store unstructured data with classified classification characteristics. [2] With the rapid growth of usage, the service interface pressure of object storage grows. Flow of different users and different interface are different. Therefore, object storage requires a fine-grained flow control based on user priorities and individual interfaces to prevent service overload, thus enhancing the reliability and robustness of services.

Based on object storage platform and the token bucket algorithm, this paper will study flow access control strategy of the cloud storage service interface, implementing fine-grained flow control for different user priority and service interface.

2. Overview of Cloud Storage

This paper is based on the object storage system using the bucket model. Object storage is to support massive user remote access and unlimited capacity expansion. Object storage supports streaming writes and reads. Object storage provides an API for developers to upload their own files to the remote. The local application interface for object storage is a RESTful API. In object storage, buckets are containers for storing objects. Each object is stored in a storage bucket. Objects are the basic entities that are stored. Objects are composed of object data and metadata. The metadata is a name - value pair that describe an object. The key is the unique identifier for the object in the bucket. Each object in the bucket can only have one key. The combination of bucket, key, and user identity "*uuid*" can uniquely identify an object.

For service interface of the object storage, the prevailing flow control technology is Nginx cluster, sliding window, bucket algorithm. With Nginx's own request limitation module ngx_http_req_module[3] and the flow restriction module ngx_stream_limit_conn_module[4],distributed network applications deployed on the Nginx cluster can control flow[5]. The Nginx cluster can't implement fine-grained flow control for interfaces. High time precision division of sliding window algorithm may waste system computing power. The bucket algorithm cannot handle the burst request.

3. Relevant technical specifications

Flow shaping is the active regulation of network flow rate [6]. The typical role of network shaping is to limit the flow and burst of a certain connection that flows out of a network, so that such network packets are sent out at a more uniform speed. Flow shaping is usually done using a buffer and a token bucket and leaky bucket algorithm. When network packets are sent too quickly, they are first cached in the buffer, and the buffered packets are sent evenly under the control of the token bucket. In the Internet era, many web services borrowed from the idea of flow shaping and used network flow to control the rate of Internet service.

Leaky bucket. The bucket algorithm provides a mechanism to control the flow rate of the port, smooth the flow of flow on the network, and achieve flow shaping, thus providing a stable flow for the network. The leaky bucket algorithm abstracts the flow control model into a bucket of fixed capacity, which water flow into at an uncertain rate. But water is flowing at a fixed rate of v. When the bucket is full, the excess water will overflow.

The design idea of the leaky bucket algorithm is shown in the figure below.

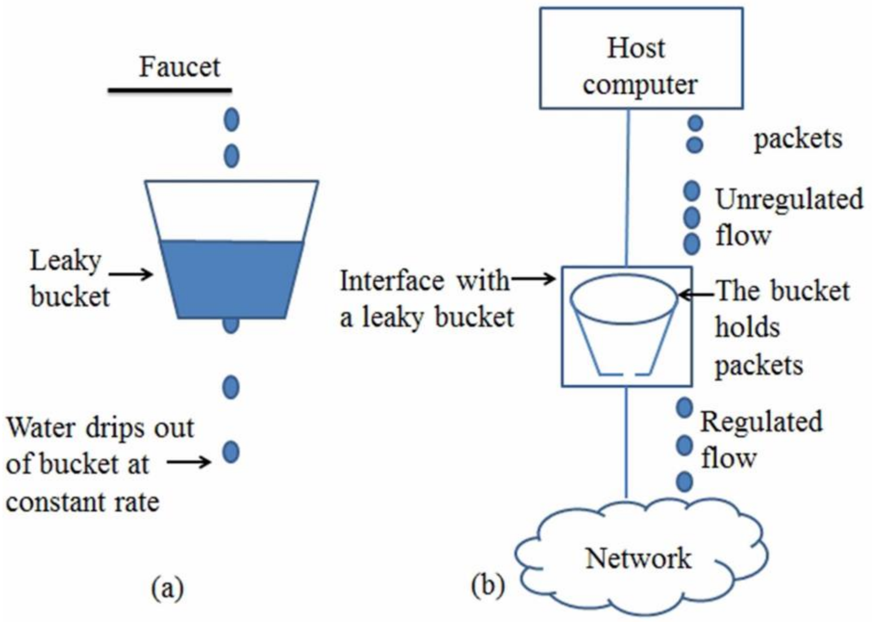

Fig.1 Schematic of Leaky Bucket Algorithm

Analogous to the flow control, the incoming packets will be placed in the data cache, and the cached size can not exceed the capacity of the cache. Once the cache is full, place it in a waiting queue, or discard it directly. Since the rate of the leaky bucket algorithm is fixed, the packets in cached will be forwarded to the network at a constant rate.

For the capacity of α, the outflow rate is v, when data request arrives:

(1) If the current capacity in the bucket is *current_capacity* $< \alpha$, then the flow is allowed to be put into the bucket and cached, waiting for forwarding or service;

(2) If the current capacity in the bucket is *current_capacity* $\geq \alpha$, then according to the setting, the flow will be placed in the waiting queue or discarded;

Because the leakage rate of the leaking barrel is a fixed parameter, even if there is no resource conflict in the network, the leaky bucket algorithm can not take the burst flow to the port rate. For many application scenarios, there is a requirement to allow burst transport, in addition to control the average transmission rate.At this time, the leaky bucket algorithm may not be suitable, and the bucket algorithm is more suitable.

Token bucket algorithm. The token bucket algorithm is a kind of network flow shaping algorithm. The token bucket algorithm is used to control the number of data sent to the network and to allow the sending of burst data. In the token bucket algorithm, there is a fixed capacity bucket with tokens in the bucket. The bucket began to be empty, and the token was filled with a fixed rate of v into the bucket until it reached the capacity of the bucket. The extra token will be discarded. Whenever a request comes,try to remove a token from the bucket. If there is no token, the request cannot pass.

The basic process of the token bucket algorithm is as follows:

(1) If the average transmission rate of user configuration is v,then a token is added to the bucket every $1/v$ second;

(2) Supposing that the bucket can hold β tokens at most. If token bucket is full, then this token will be discarded;

(3) When an n-byte packet arrives, n tokens are deleted from the token bucket and the packet is sent to the network;

(4) If there are fewer than n tokens in the token bucket, the token is not deleted and the packet is considered out of flow limits;

(5) The algorithm allows bursts of up to β bytes in length. From the long-term results, the packet rate is limited to a constant v. Packets outside the flow limit can be handled differently:

(1) Discard directly;

(2) Packets are placed on the waiting queue, transmitted when there are enough tokens in the token bucket;

(3) Packets are forwarded to the network, but special labeling is required. Discard these specially marked packets when the network is overloaded.

The principle of token bucket algorithm is shown below:

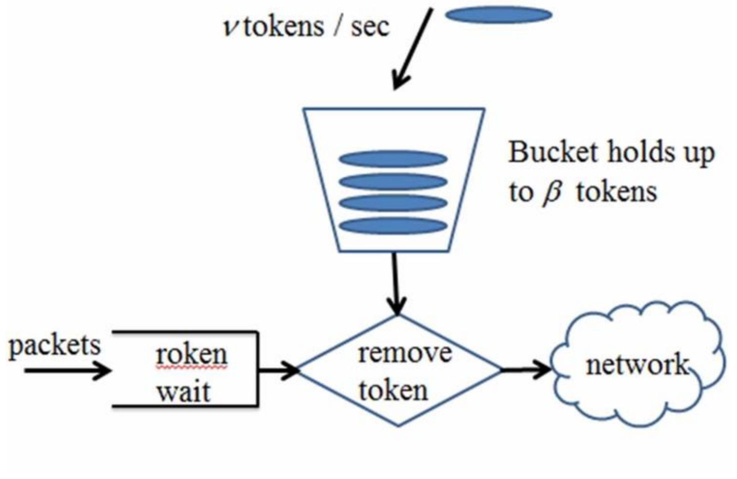

Fig.2 Schematic of Token Bucket Algorithm

Token bucket algorithms not only limit the average data transfer rate, but also allow some degree of burst transfer. In the token bucket algorithm, as long as there are tokens in the token bucket, burst data is allowed to be transmitted until the user-configured threshold is reached. Therefore, the algorithm is suitable for transmitting burst flow.

4. Flow Control Strategy Design of Service Interface

Demand Analysis. In order to flow control strategy to be more consistent with the production environment, we need in-depth analysis of various needs.

Functional requirements. The flow access control strategy should be transparent to the user. When the user normally accesses the object storage service, the flow control strategy is to control the flow, and the delay of flow control operation needs to be reduced as far as possible. When the object storage is in the face of abnormal flow, such as DDos attack and peak flow access, the flow control strategy can immediately restrict the growth of flow and ensure normal service.

Non-functional requirements. The current object cloud storage hopes to implement fine-grained service interface flow access control policies based on different users and interfaces. Before running flow control services, the service provider can configure the roaming access control strategy of the service interface itself.

Design of Flow Control Strategy. The main function of flow control strategy is to simulate the token bucket algorithm to determine whether current flow needs to be restricted. In this paper, three flow control strategies are designed. These three strategies are based on the token bucket algorithm. the basic idea is that each request requests a token to the flow control service. if the token exists, then the forwarding is allowed. If the token is empty, the request is denied.

A Service Strategy Based on User Priority. This strategy provides an algorithm similar to priority scheduling. The algorithm needs to compute priority for each user based on one or more attributes of the user. For example, different users can use multiple computing resources such as network bandwidth, memory, CPU, hard disk, and so on on the cloud computing platform. These resources are given different weights, and different amounts of the same resource are assigned to different score. The sum named *sum_weight* of the product of the user's resource score and weight is calculated. Assume that according to the range of *sum_weight*, the user's priority is divided into four types from high to low, namely $c4$, $c3$, $c2$, $c1$。 Different priority users correspond to four token bucket instances. The four token bucket instances set different token generation rates $v4$, $v3$, $v2$, $v1$,which satisfy the size relationship: $v4> v3> v2> v1$. Let the average current production environment QPS is vq, so that $vq < v1$. When the request arrives, the strategy determines whether the user can get the token based on the user's priority. If the token is obtained, processing the request is allowed. Otherwise, refuse the request. The presence or absence of the token is closely related to the token generated rate configured by the service provider according to priority of different users .

The algorithm steps of this strategy are as follows:

(1) Gets the user's *uuid* from the request data and determine whether the current flow control strategy is already open. If the strategy service is not turned on, the current request does not need to intercept based on the user priority service policy to check the next blocking policy.

(2) If the intercept strategy has been turned on, according to the user's *uuid*, this algorithm get the user priority named *c* attached to the current request. According to the user's priority, this algorithm obtains the corresponding token bucket algorithm instance named *bucket_instance*。

(3) Call The function of token bucket *bucket_instance* to get the token. If the current request gets the token, the request is allowed to be processed.

(4) If the current request gets a token failure, then the service request is rejected.

A service policy based on interface priority. The service policy based on interface priority limits the access frequency of users to a service interface in the time window named *windows_time*. When the user accesses the interface, the constant *access_times*, the maximum number of accesses in the current time window, decides whether to reject the service.

This policy sets a priority C for each interface based on the processing time and system consumption of different interfaces. This policy takes the eight interfaces in table 1 as an example and sets three priority levels. Different priority interfaces, which are given different allowable access times *access_times* in a time window. There is a variable *expire_time* in this policy that represents the expiration time of a time wondow. Local interface-class map data structure used to store the interface priority information configured by the service provider. When the user accesses the service interface of this policy, the priority of the service interface is obtained, and the service instance is obtained according to that priority.

Tab.1 Interface Priority

Name of Interface	Priority	Name of Interface	Priority
get_bucket()	1	get_object()	1
list_bucket()	0	list_object()	0
delete_bucket()	1	delete_object()	2
Create_bucket()	2	put_object()	2

For example, when *windows_time* = 50ms, *access_times* = 100, the first 100 requests pass through the 50ms time window, and the 101st request is rejected.

The algorithm steps of this strategy are as follows:

(1) When the user request arrives, the flow control service checks the policy switch and the interface information to match. If not, the flow control service executes the next policy. If the match is matched, the policy retrieves the priority information φ for that interface from the local map.

(2) According to the priority, get the current policy's *windows_time* and *access_times*, *expire_time*.

(3) If *expire_time* = 0, this is the first visit, allowing the visit, while setting *expire_time* = *windows_time*+*Unix Time()*, updating the *remain_times* = *access_times*. The function *UnixTime()* is used to get the current time.

(4) If *expire_time* ≠0, and *expire_time* < *UnixTime()*, indicating that the last window cycle is over, setting *expire_time* = 0, and executing step 3.

(5) When *expire_time* ≥ *UnixTime ()*, if the *remain_times* < 1 ,indicating that the number of this visit is insufficient, the request is rejected. If the *remain_times* ≥ 1, the value of the *remain_times* is reduced once within the allowable range, allowing this request.

A Service Strategy with Burst Time. This strategy is based on a token bucket algorithm that resolves the degradation of the token bucket algorithm's processing rate to a constant rate ν when a large number of requests come in for a long time. During a time period of the strategy, the time is divided into a burst time named *burst_time* and a normal service time named *normal_time*. These two periods correspond to their respective maximum number of visits *burst_times* and *normal_times*,and satisfy *burst_time* < *normal_time*， *burst_times* > *normal_times*. When the user first requests, the strategy allows the request, and set the current time as the start time of the cycle. If the subsequent request is within burst time, the request will try to obtain a token of burst time.Otherwise,the request tries to get the token of normal service time. If the token is given, the request is passed, otherwise the number of access times is exceeded and the service is denied.

For example, setting *burst_time* = 100ms, *normal_time* = 900ms, then the service request arrives, if the time is within 100ms on the current cycle, then the request of former *burst_times* times is normal service, otherwise the request is rejected. If the request arrives at 100ms to 1000ms, then the request of the former *normal_times* times is allowed, otherwise the service is rejected. The algorithm then goes into a loop of two time periods.

The algorithm steps of this strategy are as follows:

(1)The algorithm checks if the strategy switch is on when a user request arrives. If not, skip the following steps. If enabled, get *burst_time*, *normal_time*, *burst_times*, *normal_times*.

(2)If *start_time* = 0, indicating the first request, the strategy will set *start_time* = *Unix_time()*, *burst_remain_times* = *burst_times*, *normal_remain_times* = *normal_times*,and allow the request。

(3)If *start_time* ≠ 0, calculate *gap_time* = UnixTime () - *start_time*.

(4)If *gap_time* ≤ *burst_time*, it indicates that the current request is within the burst time and tries to acquire the burst time token *burst_remain_times*. If *burst_remain_times* ≥ 1, the request is allowed to decrement the value of *burst_remain_times* by one. Otherwise, it indicates that the number of tokens in the burst time has run out, rejecting the request.

(5) If *gap_time* > *burst_time* and *gap_time* < *burst_time* + *normal_time*, it indicates that the current request is during the normal service time . the current request tries to obtain the normal service token *normal_remain_times*. If *normal_remain_times* ≥ 1, it indicates that the request is currently permitted within the allowed number of accesses, and decrements the value of *burst_remain_times* by one. Otherwise, it indicates that the number of tokens in normal service time has run out, and the request is rejected.

(6) If *gap_time* > *burst_time* + *normsal_time*, the previous cycle has ended.Setting *start_time* = 0 and going to the second step to continue.

5. Architecture Design

Flow control services architecture is as follows. Object storage is mainly divided into three parts: flow control module, Nginx load balancing module, object storage service node OSS Node. Nginx is used for load balancing; the flow control module intercepts requests according to the setting of three kinds of strategies and provides flow control service by way of function call.

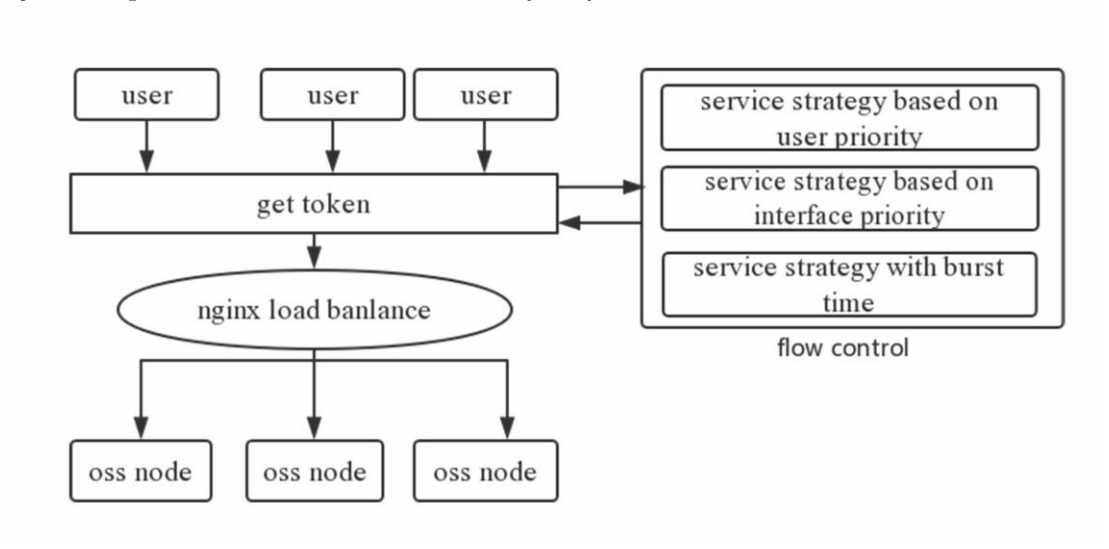
user
user
user
get token
nginx load banlance
oss node
oss node
oss node
service strategy based on user priority
service strategy based on interface priority
service strategy with burst time
flow control

Fig.3 Diagram of System Architecture

In the steps of the flow control service, the service provider first configures the switch status and the related parameter values of the three interception strategies in the flow control service. Each request sent to the server needs to determine whether the current request is allowed according to the three kinds of flow control strategies. If the request is allowed to pass, then enter the Nginx load balancing module, according to the load balancing results, the request is assigned to an object storage service node to handle the corresponding request. Three flow control strategies are carried out in turn, after a request is verified by an interception strategy, the subsequent interception strategy will no longer intercept the request for verification.

6. Experiment

In order to verify the effectiveness of flow control strategy, this paper designs a cloud storage prototype system in the experimental environment. Three flow control strategies will be tested on the prototype system.

Experimental Development Environment. The software and hardware development environment of the flow control control service is as follows.

Tab.2 Development Environment

Environment	Name	Environment	Name
operating system	Ubuntu16.04	development tools	Pycharm and git
CPU	1.9G AMD A4-3305M	Python version	2.6
memory	4G	database	MySQL 5.5

Experiment Procedure. In the object storage prototype system, in order to verify three kinds of intercepting strategies, eight interfaces in Table 1 were designed and implemented. Buckets and objects are stored in the database。There are four priority users in the prototype system. When a user requests an interface, the user's priority information will be attached. Using multithreading technology, the three flow control strategies were tested separately in the experiment. The experiment continue to visit the interface, and gradually increase the frequency of requests, record server QPS. The result is shown in the figure.

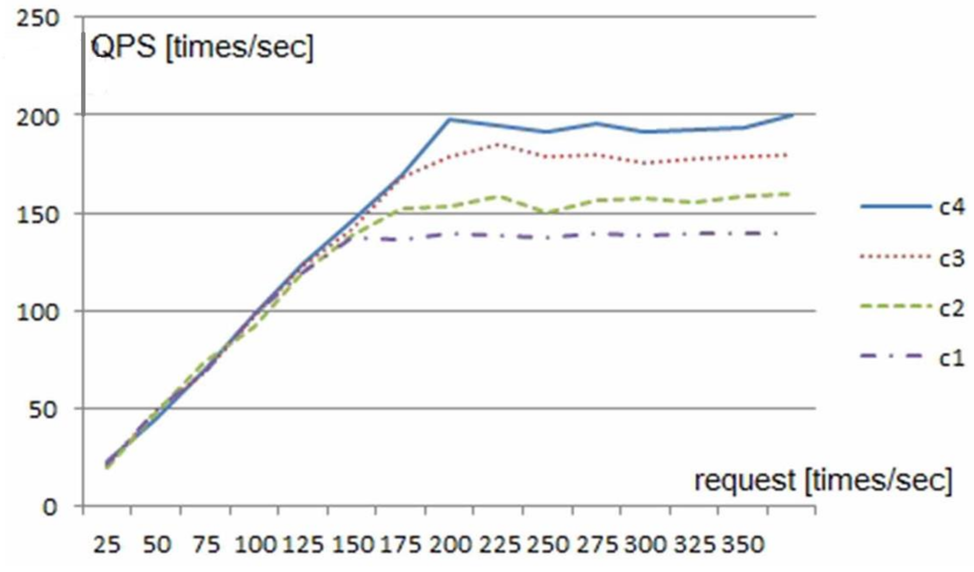

Fig.4 Test Result of Service Strategy Based on User Priority

When testing the service strategy based on user priority, the experiment sets *v4* = 200, *v3* = 180, *v2* = 160, *v1* = 140. As can be seen from the figure, as the access rate increases, the QPS of the object storage service gradually increases, and the number of user requests to receive the token is limited by the number of interception strategies. The result of service strategy based on interface priority is similar to this figure.

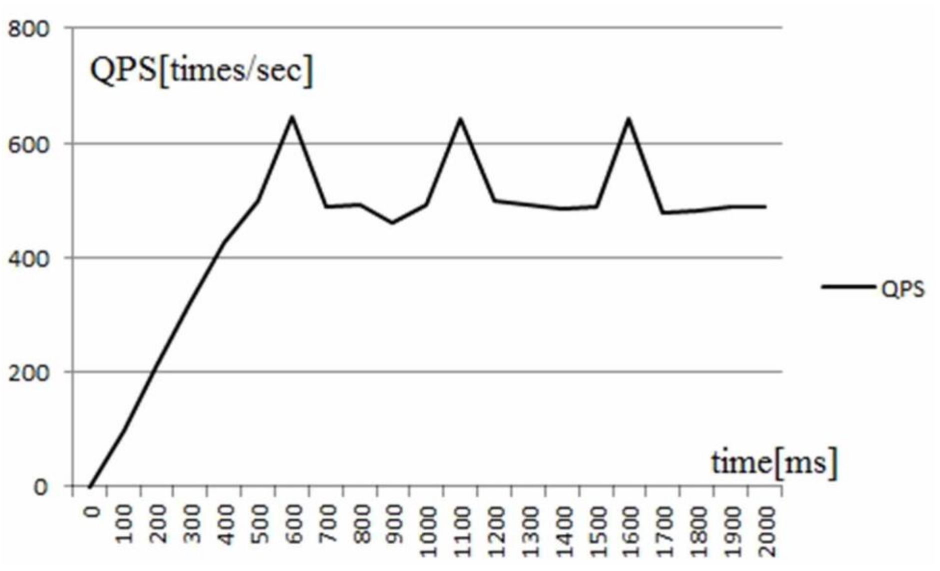

Fig.5 Test Results of Service Strategy with Burst Times

When testing service strategy with burst time,the experiment sets *burst_time* = 100ms, *normal_time* = 400ms, *burst_times* = 650, *normal_times* = 500. As requests increase, the QPS of service interfaces fluctuate dramatically during the burst, which helps to squeeze system performance.

Experiments show that the three flow control strategies can effectively control the flow and the object storage service can get effective overload protection.

7. Conclusion

In summary, the object storage due to mass storage, unlimited expansion, remote login, low price and other characteristics, has been widely used by enterprises and developers. Due to the huge amount of users and storage, the object storage has to face the problem of flow access control of the service interface. This paper designs three kinds of flow control services based on token bucket algorithm from the aspects of research background, research status, requirement analysis. This article establishes the object storage function prototype and verifies the validity of three flow access control strategies.

Acknowledgment

This work was supported by the National Natural Science Foundation of China(Grant No. 11272066 and No. 11472049).

References

[1] Shi Qiang, Zhao Peng-yuan. Analysis of Critical Technologies on Cloud Storage Security. Journal of The Hebei Academy of Sciences,2011,28(03):66-69.

[2] YANG Teng-Fei, SHEN Pei-Song, TIAN Xue, FENG Rong-Quan. Access Control Mechanism for Classified and Graded Object Storage in Cloud Computing. Journal of Software, 2017, 28(09): 2334-2353.

[3] Xie W, Li Y, Lu C, et al. Optimizing the resource-updating period behavior of HTTP cache servers for better scalability of live HTTP streaming systems. Broadband Multimedia Systems and Broadcasting (BMSB), 2012 IEEE International Symposium on. Seoul:IEEE, 2012:1-6.

[4] Liu Zhenyu. Website Load Balancing with Nginx. China Management Informationization,2012(16):96.

[5] Zhong Tianhui. Design and Implementation of Flow Control Service Based on Token Bucket Algorithm. Dalian university of technology,2016.

[6] Chakrabarti Deepayan, Vee Erik. Traffic Shaping to Optimize Ad Delivery. ACM Transactions on Economics and Computation, 2015, 3:131一148.