Quotas

by maoling

0. Why is quota necessary?

It is possible for clients to send/fetch very high volumes of data and thus monopolize server resources, cause network saturation and even DOS attack to server. Having quotas protects against these issues and is all the more important in large multi-tenant clusters where a small set of badly behaved clients can degrade user experience for the well behaved ones. Quota provides an isolation effect on users applications, eliminate the noisy of neighbor, avoid users to deploy multiple sets of zk cluster for every application in most general scenarios. Of course, because in the same JVM instance, this isolation provided is not as hard as the hardware/physics isolation.

1. Space Quota

We now can use quota feature to limit the the total children count or bytes under a znode. we can called this Quota type: `Space Quota`. (Not repeat it again here).

2. Throttle Quota

2.0 brief introduction

We also require another Quota type: `Throttle Quota`(or throughput Quota) which is able to limit the QPS of requests, throttle request peak/pressure and smooth request traffic. Throttle Quota can also have protective effects on the server as `RequestThrottler` did, and more flexible as it can be tuning during the runtime

2.1 manual

- Throttle type: it can be expressed as READ, WRITE, or the default type(read + write).
- Timeframes: it can be expressed in the following units: sec, min, hour, day. Actually, for simplification, it's enough to only support sec for most user cases.
- Request sizes: it can be expressed in the following units: B (bytes), K (kilobytes), M (megabytes), G (gigabytes), T(terabytes), P (petabytes). For example: 1048576 bytes/sec means: allow to request 1MB data every second.
- Request number: it is a number. For example: 1000 reqs/sec represents: allow one thousand requests every second.
- How to measure the request rate?
    - Most systems take a `Sliding Window algorithm` approach. Its base idea is: the sliding window is that instead of making discrete jumps as the fixed window does, it smoothly slides with passing time. For example, split the one second into ten small time window (every time window is 100ms, we call it a slot or block), then judge which slot the current request located and calculate current QPS. more detail in [6]

```
AtomicLong[] countPerSlot = new AtomicLong();
countPerSlot[index].incrementAndGet();
count.incrementAndGet();
index = (index + 1) % slot;
scheduledExecutorService.scheduleAtFixedRate(slidingWindowRateLimiter, 100, 100,
TimeUnit.MILLISECONDS);// slot = 10;
```

2.2 Scope

Background: Suppose that we have 26 downstream businesses(from business-A to business-Z). If latency spike happens and traffic surges, how can we detect which business is responsible for this and control this emergency situation rapidly? To meet this demand, we need to do some works to identity the client. The scope of Throttle Quota can be implemented from these two alternative dimensions: `User` or `Client.id`

2.2.1 User:

ZooKeeper doesn't have a thorough user management component. For simplification, we can take advantage of the `username:password` of ACL digest auth.

```
String auth = userName + ":" +password;
zk.addAuthInfo("digest", auth.getBytes());
```

2.2.2 Client.id:

`client.id` is a logical grouping of clients with a meaningful name chosen by the client application.
`client.id` are defined in the application using the `client.id` property. A client ID identifies an application making a request.

2.2.2.1 How to generate the client.id?

- If using a unauthorized client, the client id is hard-coded by a "client.id" property in the `ZKClientConfig`.
- If using an authorized client, we can obtain the secured identity from client certificate as the client.id.
  For a Kerberos' example:

```
Client {
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  principal="businessA/cluster2-host1@HADOOP.COM";
};
```

Parse the keyword: "businessA" as the client.id. Of course, an authorized client can also use the hard-coded way as the unauthorized client did. Some discussions in [5].

2.2.3 Hard/Soft Throttle Quota

- For the hard Throttle Quota, the server will reject requests and reply the client a `QuotaExceededException` by force.
- However, for the soft Throttle Quota, we should not do nothing, just log the warnings as count or bytes space quota did. At this case, when server detects this quota violation, the server slows down a client exceeding its quota. It computes the amount of delay needed to bring a guilty client under it's quota and delays the response for that time. This approach keeps the quota violation transparent to clients. This also keeps them from having to implement any special backoff and retry behavior which can get tricky. In fact, bad client behavior (retry without backoff) can exacerbate the very problem quotas are trying to solve. Of course at this case, users need to increase the value of `zookeeper.request.timeout` to avoid too much client's timeout

3 Some existing designs

3.1

In the original implementation of ZOOKEEPER-1383, it counted the throughput from `ServerCnxn#packetReceived` based on a simple time-window approach. However it did not distinguish between read and write requests, and it's not easy and appropriate to do it at that place: `ServerCnxn`. It used the `ServerCnxn#resetStats` to split/divide time windows, it's not good for controlling the traffic more precisely in a fine-grained manner.

```
A Simplified request workflow:
request ---> ServerCnxn ---> RequestThrottler ---> PrepRequestProcessor --->.....---> FinalRequestProcessor
```

3.2

In the FB's design[1], they combine with `RequestThrottler` based on the `Rolling-window rate throttling` approach. When request arrives at the `RequestThrottler`, asks for the `RequestOracle` to update/check Quotas and make a throttling decision, then pipeline requests in the RequestProcessors. more details in [1],[2]

Reference:

- [1] https://www.facebook.com/zkmeetup/videos/2984998221531756/
- [2] https://engineering.fb.com/open-source/zookeeper-meetup/
- [3] https://kafka.apache.org/documentation/#design_quotas
- [4] https://kafka.apache.org/documentation/#quotas
- [5] https://issues.apache.org/jira/browse/ZOOKEEPER-3467
- [6] https://tech.domain.com.au/2017/11/protect-your-api-resources-with-rate-limiting/