Manageable Call Queue

Problem

Now, the feature HBASE-11355 seperates the single Call Queue into read call queue and write call queue, and the feature HBASE-11724 splits the read call queue into get call queue and scan call queue. Administor can assign fixed number of handlers for the specified queue through the conf file hbase-site.xml . It's helpful in some outages , to avoid all read or all write requests ran out of handler threads.

however, there are still serveral problems:

1. workloads in the same request type queue(such as write queue) may influence each other as before, consider the following scenario:

(1) both client-1 and client-2 send write requests(all requests will pass to the same write-queue), the client-1 write the large objects(100KB record) , and client-2 write the small objects (1KB record). Test shows the client -1 will ran out of all handler threads of the write-queue, and decrease the client-2 throughput

(2) both client-3 and client-4 send get requests(all requests will pass to the same get-queue), the client-3 search all data from lots of hfiles(all search key are equally popular), read latency is high. the client-4 do not require any I/O resources(say, data is cached). Sometimes, the client-3 will accupy all handler threads of the get-queue, and increase the read latency of client-4

2. administor can only adjust the ipc queues/handlers with some static tuning options ,such as property ‘hbase.ipc.server.callqueue.handler.factor’, can't increate/decrease the handler number for the specified queue at runtime.

Solution

The Manageable Call Queue aims to provide a unified way to build:

1. Administor can create queue with specified number handler using shell command
2. Administor can increase/decrease the handler number for the specified queue using shell command
3. Administor can assign the same type request load to one queue, for example,

administor can assign all the large object write requests to one queue, while assign all the small object write requests to another queue.

The procedure below describes the operation of the feature.

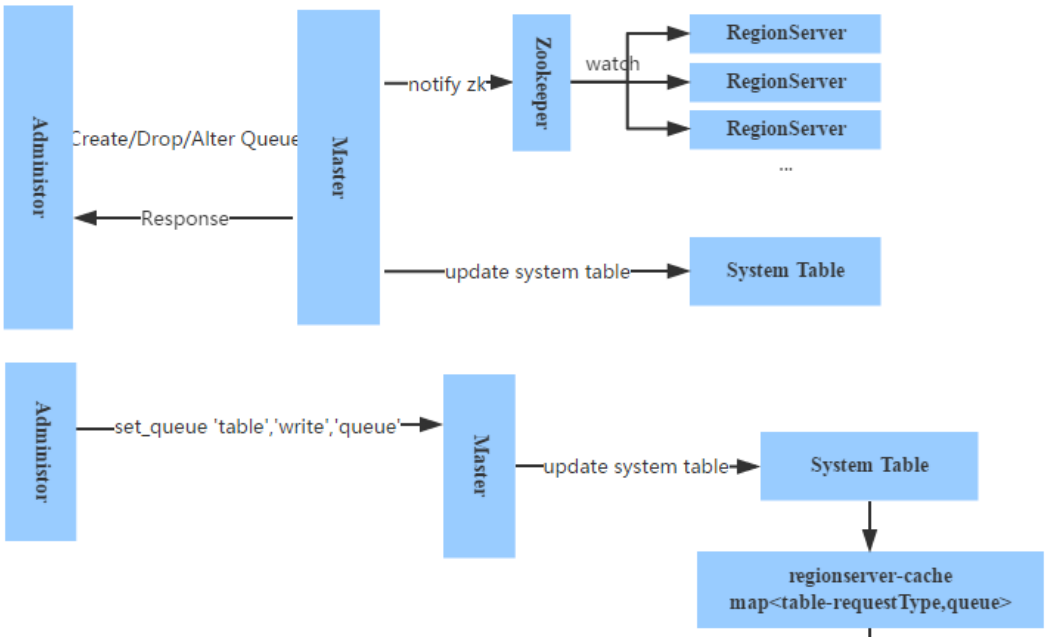
- administor creates a queue to accept all requests which write large object(100 KB record) with command : create ‘large-queue’,{‘handler’ => 5};
- then, administor assigns the write request of the table with large object to the ‘large-queue’ with command: set_queue 'table','write','large-queue' . All the large object write request on the table will be dispatched to the large-queue.
- Similarly, administor can also create a ‘small-queue’ for all the small object write load, then all the small object write request will be dispatched to the small queue.

Since different write request loads would be dispatched to different queue, we believe that the feature is a improvement in the case.

Design Framework

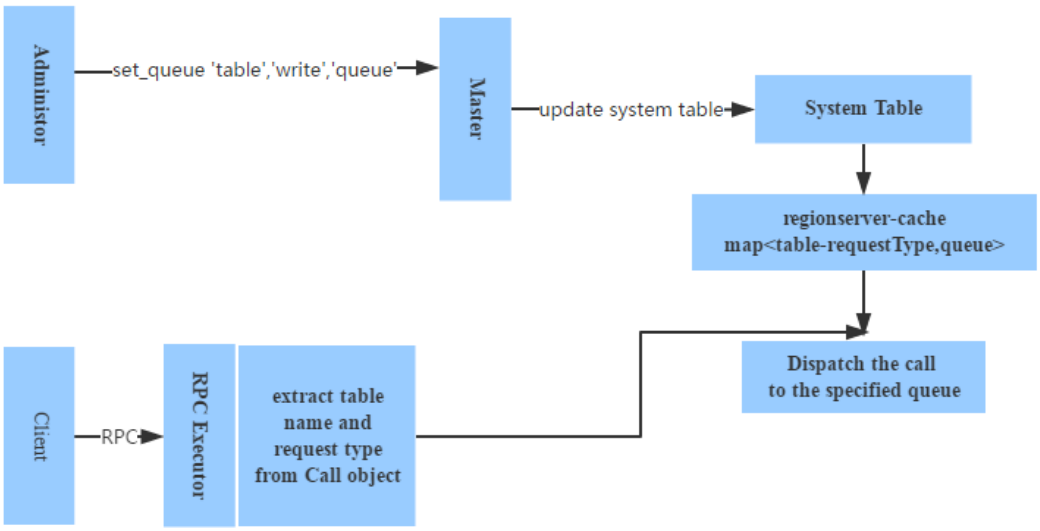
The Manageable Call Queue has two main components, a “ Manageable Queue Manager” and a “Manageable Queue RPC Executor”. The queue manager is responsible to create/drop/update the queue and assign the requests on a table to the specified queue while the rpc executor dispatch the call to specified queue based on it’s request load.

Queue Manager



- administor send a manager request (such as create queue) to master, master notifies all regionserver through zookeeper. regionserver will initial a queue and start specified number handler threads blocking on the queue. In addition, master also updates system table 'hbase:callqueue' to store the queue info(queue name and handler number) in hbase.
- administor send the set_queue request (set_queue 'table1','write','queue1') to master, master updates the system table 'hbase:callqueue' to store the mapping relation <table-requestType, queue> in hbase. For example, the above request will convert to mapping relation <table1-write, queue1>. At the same time, all regionservers start a scheduled chore to fetch the mapping relation from system table 'hbase:callqueue' every 3 minute to update the cache (map<table-requestType, queue>).

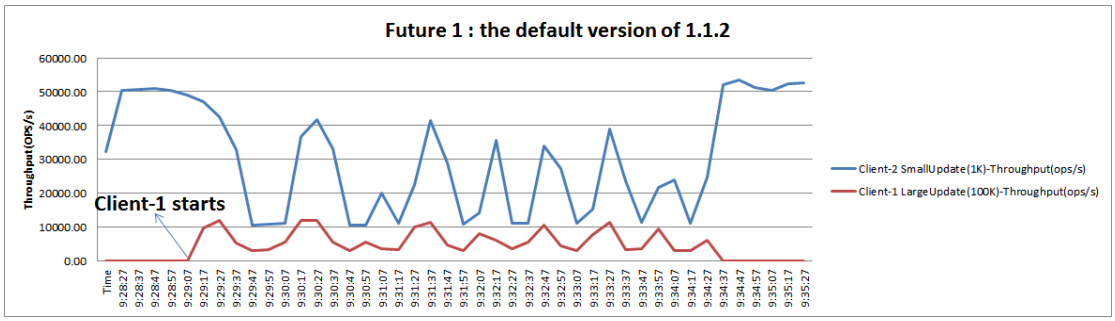
Manageable Queue RPC Executor



The client send request to regionserver, the RPCExecutor will extract the table name and request type from the call object. and then dispatch the call to the specified queue based on the cached mapping relation (map<table-requestType, queue>).

Use Case

I am runing expriment at the following scenario : both client-1 and client-2 send update requests, the client-1 write the large objects(100KB record) into table-1, and client-2 write the small objects (1KB record) into table-2. The charts below shows the effect of the Mutilple-Type queue feature:

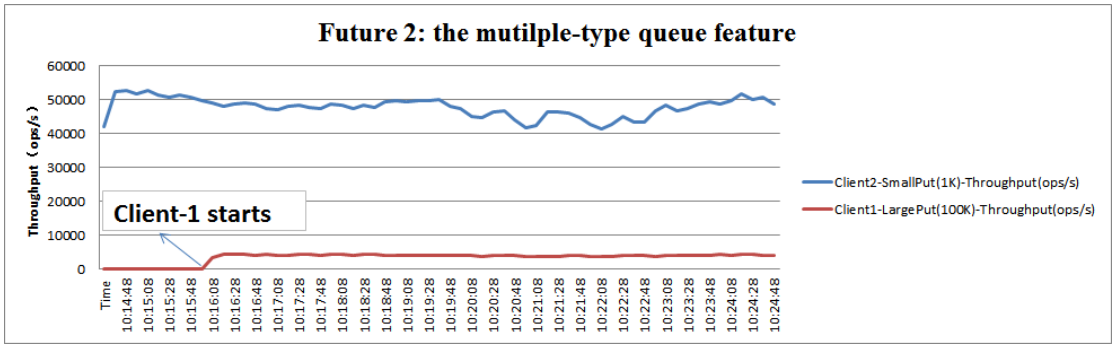


as shown in future 1(the default version of hbase 1.1.2):

1. in the beginning, client-2 starts to sends requests and client-1 not. the throughput of client-2 keep stable

2. when client-1 starts, the throughputs of both client-1 and client-2 are very unstable, that is to say client-1 influence client-2 seriously

3. the average hbase throughput is 36641 .



Afterwords, we assign 150 handlers to table-2 and 3 handlers to table-1 with manageable call queue feature. as shown in future 2, the throughputs of both client-1 and client-2 are more smooth, the average hbase throughput is 51653 , 41% higher.