HedWig TopicManagement

Topic management in Hedwig

ZooKeeper data structure

Metadata about topics, subscribers, and hubs will be stored in ZooKeeper. For a given Hedwig region, we will store the following structure:

The rectangles in this diagram are znodes; rectangles with dashed borders are ephemeral znodes.

- **hedwig** is the root. If the ZooKeeper instance is shared between regions, then there would be a region-specific root.
    - **regionname** is the region this Hedwig cluster lives in.
        - **topics** is the root for the topics subtree. All topics that have been created live under this root.
            - **T1, T2 ...** are nodes for each topic. The nodes are named by the topic name. The fact that a topic has a node under the **Topics** node means that the topic exists. If we had any other topic metadata (like permissions) that we wanted to store, we could store them as the content of the node.
                - **Ti.hub** is the hub that is currently assigned to the topic. This is an ephemeral node, so that if the hub fails, another hub can take over. The name of the node is "Hub" and the content is the hostname of the hub.
                - **hubscribers** is the root of the tree of subscribers to the topic.
                    - **Sub1, Sub2 ...** are the subscribers. Each node is named with the subscriber ID (which should be descriptive enough for the hub to be able to deliver messages to the subscriber.) The content of the node includes the subscriber's current consume mark for the topic.
        - **hosts** is the root of the tree of hubs in the region.
            - **S1:port, S2:port ...** are the hubs. Each node is named with the hostname:port of the hub. The fact that a node exists for the hub means that the hub is in the cluster, and is eligible to be assigned topics.
                - **Alive** is an ephemeral node that indicates whether the hub is currently alive.

Topic creation

Topic creation and assignment to a hub is a lazy process. Topics are created on demand (e.g., when there is a subscriber) and assigned to a hub on demand (e.g., when there is a new subscriber or a message published.) When a hub responsible for a topic fails, we reassign the topic on demand; e.g. when the connected subscribers reconnect.

Subscription process

The subscribe call takes three parameters:

- The subscriber ID (string)
- The topic name (string)
- A flag (boolean) indicating whether the client has been redirected.

When a client C subscribes to a topic T, it will contact one of the hubs (say, H1) and send a *subscribe(C,T,False)* message. When a client receives a *redire ct* message from a hub, it will retry its subscription to the hub listed in the message (e.g. H2). It will do this by sending a *subscribe(C,T,true)* message to the hub H2. The flow is similar to the "false" case, except that the hub H2 knows that it should try to become the owner of the topic, instead of choosing a random hub.

Upon receiving a *subscribe* message for topic T, the hub H1 will follow these steps:

- The hub H1 will check in ZooKeeper to see if the topic T exists as a child of **Topics**. If the topic T does not exist:
    - H1 will create the node **Topics.T**, and the node **Topics.T.Subscribers**.
- The hub H1 will read **T.Hub**, the current hub assigned to the topic (say, H2).
    - If a hub exists, and is the same hub the client contacted (e.g. H1==H2), then H1 will add C to the list of subscribers (under **T. Subscribers**), and set up its internal bookkeeping to begin delivering messages to C.
    - If a hub exists, and is a different hub than the one the client contacted (e.g. H1!=H2), then H1 will return to the client a *redirect* message, requesting that the client retry its subscription at H2.
    - If no hub exists, H1 will check the flag of the *subscribe* call to see if the client was redirected.
        - If false (the client has not yet been redirected) H1 will choose a random hub H3 (possibly itself) to manage the topic.
            - If H1 chooses itself (H1==H3), then H1 will try to create an ephemeral node under **T** called **Hub** with its own hostname (e.g. H1) as the content. This creation should be done using test-and-set, so that if a **Hub** node already exists, the creation fails.
                - If the ephemeral node creation succeeds, then H1 will set up its internal bookkeeping to begin delivering messages to C.
                - Otherwise (ephemeral node creation fails) then H1 will read the hostname (e.g. H4) of the hub assigned to the topic from the **T.Hub** node, and return to the client a *redirect* message, requesting that the client retry its subscription at H4.
            - Otherwise (H1!=H3), then H1 will return to the client a *redirect* message, requesting that the client retry its subscription at H3.
        - If true (the client has been redirected), then H1 will try to become the owner of the topic.
            - H1 will try to create an ephemeral node under **T** called **Hub** with its own hostname (e.g. H1) as the content. This creation should be done using test-and-set, so that if a **Hub** node already exists, the creation fails.
                - If the ephemeral node creation succeeds, then H1 will set up its internal bookkeeping to begin delivering messages to C.

        - Otherwise (ephemeral node creation fails) then H1 will read the hostname (e.g. H3) of the hub assigned to the topic from the **T.Hub** node, and return to the client a *redirect* message, requesting that the client retry its subscription at H3.

Notes:

- Because we want subscribers to be directly connected to the hub responsible for the topic, we will redirect the client to that hub.
- Because the **T.Hub** node is ephemeral, it must be created by the hub that owns the topic, not by any other hub.
- To decide which hub to assign a topic to, the deciding hub should use the current list of alive nodes from **Hedwig.Hubs** in ZooKeeper.
- When choosing a random hub to assign a topic to, we can either do it uniformly randomly or by weighting the random choice based on the hub's current load. To do the latter, each hub must record in its **Hubs.Si.Alive** node its current load, and then hubs doing topic assignment use these values to make decisions.

Re-subscription process

A client may become disconnected from a hub, for many reasons including:

- The hub has failed
- The network experienced a problem
- The hub has abandoned the topic (see topic redistribution below)

When this happens, the client can just resubscribe to the topic. Using the same subscription process as above, Hedwig will direct the client to the appropriate hub, either the (old) hub which still owns the topic, or a (new) hub which has taken over the topic.

Publish process

When a client C publishes to a topic T, C contacts a hub (say, H1) and tries to publish. The publish call takes four parameters:

- The publisher ID (string)
- The topic T
- The message M
- A flag (boolean) which indicates whether the client has been redirected.

When the client C sends a *publish* call to H1 to publish a message on topic T, H1 follows these steps:

- The hub H1 will check in ZooKeeper to see if the topic T exists as a child of **Topics**. If the topic T does not exist:
    - H1 will create the node **Topics.T**, and the node **Topics.T.Subscribers**.
- The hub H1 will read **T.Hub**, the current hub assigned to the topic (say, H2).
    - If a hub exists, and is the same hub the client contacted (e.g. H1==H2), then the hub accepts the publish and writes it into the BookKeeper log.
    - If a hub exists, but is a different hub than the one the client contacted (e.g. H1!=H2), then the hub returns a *redirect* message, requesting that the client retry its publish at H2.
    - If no hub exists, H1 will check the flag of the *publish* call to see if the client was redirected.
        - If false (the client has not yet been redirected) H1 will choose a random hub H3 (possibly itself) to manage the topic.
            - If H1 chooses itself (H1==H3), then H1 will try to create an ephemeral node under **T** called **Hub** with its own hostname (e.g. H1) as the content. This creation should be done using test-and-set, so that if a **Hub** node already exists, the creation fails.
                - If the ephemeral node creation succeeds, then H1 will set up its internal bookkeeping to begin publishing messages on T, and will accept and publish C's message.
                - Otherwise (ephemeral node creation fails) then H1 will read the hostname (e.g. H4) of the hub assigned to the topic from the **T.Hub** node, and return to the client a *redirect* message, requesting that the client retry its publish at H4.
            - Otherwise (H1!=H3), then H1 will return to the client a *redirect* message, requesting that the client retry its subscription at H3.
        - If true (the client has been redirected), then H1 will try to become the owner of the topic.
            - H1 will try to create an ephemeral node under **T** called **Hub** with its own hostname (e.g. H1) as the content. This creation should be done using test-and-set, so that if a **Hub** node already exists, the creation fails.
                - If the ephemeral node creation succeeds, then H1 will set up its internal bookkeeping to begin publishing messages on T, and will accept and publish C's message.
                - Otherwise (ephemeral node creation fails) then H1 will read the hostname (e.g. H3) of the hub assigned to the topic from the **T.Hub** node, and return to the client a *redirect* message, requesting that the client retry its publish at H3.

Notes:

- We may want to optimize this procedure so that the hub does not have to contact ZooKeeper on every publish. This could be done using leases, or by depending on the disconnect exception...the proper way to do this is an open question.
- Instead of redirecting the client, we could forward the published message to the correct hub. But redirecting seems cleaner, since we would really like the publisher to be directly connected to the correct hub.

Topic redistribution

Occasionally, we should shuffle topics between hubs to ensure load balancing. For example, when a new hub joins, we want topics to be assigned to it. Similarly, if some topics are hotter than others, the hub should be able to shed load. Since all of the persistent state about a topic is in ZooKeeper or BookKeeper, shuffling a single topic can be easy: the hub just stops accepting publishes and deletes its ephemeral node. The next time a client tries to subscribe or publish to the topic, it will get assigned to a random hub.

When should a hub abandon a topic? It should do so at least under the following conditions:

- If the hub is overloaded, compared to the other hubs in the system (determined by load statistics stored by the hubs in their ZooKeeper **hedwig.regionname.hosts.Si** node.)
- Periodically (e.g. when the hub is closing the BookKeeper log for the topic)

The constant shuffling of topics should help to keep load evenly balanced across hubs, without human intervention. Moreover, lazily abandoning topics will help the shuffling to occur in an incremental fashion spread out over time.

Open questions

- How to handle the lease of the hub's primary status for a topic
- How to handle access control - who is allowed to create topics, who is allowed to subscribe to them, who is allowed to publish to them