HedWig

Hedwig

Hedwig is a **publish-subscribe** system designed to carry large amounts of data across the internet in a **guaranteed-delivery** fashion from those who produce it (**publishers**) to those who are interested in it (**subscribers**). The goals of Hedwig are:

1. **Guaranteed Delivery**: The messages may not be lost even when a particular subscriber goes offline for extended periods of time.
2. **Topic-based**: Publishes and subscribes happen to a topic. The system should scale to ~10^6 topics with ~10 subscribers on each topic.
3. **Incremental Scalability**: It should be possible to scale capacity by adding servers on the fly. The up-front hardware investment should not be huge.
4. **High availability**: The system should remain available in the presence of single server failure without manual intervention.

While there are a lot of commercial and open-source products in the general pub-sub category, none of them satisfy all the above 4 requirements (they mostly fail on 3 and 4).

Architecture

At the highest level, Hedwig is a collection of *regions* spread across the internet. Any region may publish on a topic, and those messages must be delivered to any subscriber in any region that has subscribed to that topic.

? Unknown Attachment

Now digging down into a region, it consists of a collection of **hub servers**. Hub servers aggregate messages published in a region and persist them. They also subscribe to hubs in other regions to listen for messages that their clients are subscribed to. Clients always subscribe only to local hub servers. Hedwig plans to use Zookeeper for persistence of metadata, and Bookkeeper for persistence of actual messages.

? Unknown Attachment

Topics are randomly split over hubs. When the hub responsible for a topic fails, another hub should take over responsibility of the topic.

Now digging into a hub, it consists of 6 components:

? Unknown Attachment

1. **Network-I/O component**: For high throughput, we use Java NIO through a framework called Netty.
2. **Topic Manager**: Maintains and coordinates ownership of topics among hubs. It is responsible for doing automatic failover when a hub dies. It will be a Zookeeper client.
3. **Subscription Manager**: Maintains information about which subscriptions exist in the system, and their **consume-points**: the point in the topic until which a subscriber has already received and acknowledged, so that the next delivery of a message can start from this point.
4. **Persistence Manager**: Persists messages in a reliable way so that they can be retrieved sequentially later. It will be a Bookkeeper client.
5. **Remote Subscriber**: Subscribes to hubs in other regions for the topics, so that the messages published there can also reach the local clients. This component will be our own Hedwig Java client.
6. **Delivery Manager**: This component will be responsible for delivering messages to the subscribers. The subscribers can be either our own local clients or the hub in other regions.

Code-Level Design & Details

The hedwig project consists of 3 modules:

- Protocol: This is a simple module that specifies the client-server protocol used by Hedwig. The protocol is specified as a Protocol Buffers file. Protocol buffers can generate serialization and deserialization code for us in multiple languages.
- Client: We support both a c++ and a java client library. The client module obviously depends on the protocol module.
- Server: This the major chunk of the system. A server is responsible for certain topics, and accepting publish and subscribe requests for them. The server uses (and hence depends on) the hedwig java client to subscribe to topics in other regions.

Get started by checking out the repository:

https://svn.apache.org/repos/asf/zookeeper/trunk

Follow the directions in the BUILD.txt in src/contrib/hedwig

The following **coding conventions** are to be followed:

1. Indentation only with 4 spaces, no tabs (adjustable using Eclipse settings).
2. Use curly braces even for single-line ifs and elses. I am sure you have encountered at least 1 bug in your life because the else got associated with the wrong if.
3. No System.outs (only logging)
4. Javadoc (even if brief) for every class. No author tags.
5. We are building a high-throughput server. Watch out for the following performance crippling gotchas:

    a. If you are logging something that requires string concatenation, even if it is at debug level, you pay the price of concatenation. Hence surround by =if(isDebugEnabled())= block
    b. Use StringBuilder while building long strings, don't just use +'s.
6. Remember that we are multithreaded which is a double-edged sword.
    a. Protect your data structures where needed with a lock.
    b. However locking needs to be well-understood and documented. Please bring it up in the design discussions.

Test

Every class that has a non-abstract method must have a test case. Maven directory structure is already set up to be able to write and run tests easily.

Topic Management in Hedwig

TopicManagement