Lakshan Perera

Real-time Collaborative Editing with Web Sockets, Node.js & Redis

25 May 2010

Few months ago, I mentioned I'm developing a real-time collaborative code editor (codenamed as Realie) for my individual research project in the university. Since then I did couple of posts on the design decisions and on technologies I experimented for the project. After some excessive hacking, today I've got something tangible to share with you.

Currently, I have implemented the essentials for real-time collaboration including ability watch who else is editing the file, view others edits, chat with the other collaborators and replay how the edits were done. You may think this is more or less similar to what Etherpad had - yes, it is! However, this is only the first part of the project and the final goal would be to extend this to a collaborative code editor (with syntax highlighting, SCM integration).

Web Sockets

The major difference of Realie from other Real-time collaborative editors (i.e. Etherpad, Google Docs & Wave) is it uses web sockets for communication between client and server. Web Sockets perfectly suit for cases like this, where we need to have asynchronous, full-duplex connections.

Compared to alternatives such as long-polling or comet, web sockets are really efficient and reliable.

In traditional HTTP messages, every message needs to be sent with HTTP headers. With web sockets, once a handshake is done between client and server, messages can be sent back and forth with a minimal overhead. This greatly reduces the bandwidth usage, thus improves the performance. Since there is an open connection, server can reliably send updates to client as soon as they become available (no client polling is required). All this makes the app truly real-time.

As of now, only browser to implement web sockets is Google Chrome. However, I hope other browsers would soon catch up and Mozilla has already shown hints for support. There are also Flash based workarounds for other browsers. For now, I decided to stick with the standard web socket API.

Taking Diffs and Applying Patches

In case if you wonder, this is how the real-time collaboration is done:

1. when one user makes a change; a diff will be created for his change and sent to server.
2. Then the server posts this diff to other connected collaborators of the pad.
3. When a user receives a diff, his content will be patched with the update.

So both taking diffs and applying patches gets executed on the client side. Handling these two actions on browser was made trivial thanks to this comprehensive library written by Neil Fraser.

However, on some occasions these two actions needs to be executed concurrently. We know by default client-side scripts get executed in a single thread. This makes execution synchronous and slow. As a solution to this I tried using the Web Workers API in HTML5 (this is implemented in WebKit & Mozilla). Separate worker scripts were used for taking diffs and applying patches. The jobs were passed on to this worker scripts from the main script and the results were passed back to main script after execution was complete. Not only this made the things fast, but also more organized.

Node.js for Backend Servers

Initially, I started off the server implementation in Ruby (and Rails). Ruby happend to be my de-facto choice as it was my favorite language and I had enough competency with it. However, soon I was started feeling Ruby was not the ideal match for such asynchronous application. With EventMachine it was possible to take the things to a certain extent. Yet, most of the Ruby libraries were written in a synchronous manner (including Rails), which didn't help the cause. As an alternative, I started to play with Node.js and soon felt `this is the tool for the job.` It brings the JavaScript's familiar event-driven model to server, making things very flexible. On the other hand, Google's V8 JavaScript engine turned out to be really fast. I decided to ditch the Ruby based implementation and fully use Node.js for the backend system.

Backend is consist of two parts. One for serving normal HTTP requests and other for web socket requests. For serving HTTP requests, I used Node.js based web framework called Express. It followed the same ideology as Sinatra, so it was very easy to adapt.

Web socket server was implemented based on the recently written web socket server module for Node.js by Micheil Smith. If you are interested to learn more about Node.js' web socket server implementation please see my earlier post.

Message delivery with Redis Pub/Sub

On each pad, there are different types of messages that users send on different events. These messages needs to be propagated correctly to other users.

Mainly following messages needs to be sent:

- When a user joins a pad
- When a user leaves a pad
- When a user sends a diff
- When a user sends a chat message

For handling the message delivery, I used Redis' newly introduced pub/sub implementation. Every time a user is connected (i.e. visits a pad) there would be two redis client instances initiated for him. One client is used for publishing his messages, while other will be used to listen to incoming messages from subscribed channels.

Redis as a persistent store

Not only for message handling, I also use Redis as the persistent data store of the application. As a key-value store Redis can provide fast in-memory data access. Also, it will write data to disk on a given interval (also, there is a much safer Append only mode, which will write every change to disk). This mechanism is invaluable for this sort of application where both fast access and integrity of the data matters.

Another advantage of using Redis is the support for different data types. In Realie, the snapshots are stored as strings. The diffs, chat messages and users for a pad are stored as lists.

There is a well written redis client for Node.js which makes the above tasks really simple.

Try it out!

I'm still in the process of setting up an online demo of the app. Meanwhile, you can checkout the code and try to run the app locally.

Here is the link to GitHub page of the project - http://github.com/laktek/realie

Please raise your ideas, suggestions and questions in the comments below. Also, let me know if you are interested to contribute to this project (this project is open source).