Comet Web Applications

How to Scale Server-Side Event-Driven Scenarios

Simone Bordet
sbordet@webtide.com
About me

- Simone Bordet (sbordet@webtide.com)
- Senior Java Engineer at Webtide
  - Previously freelance, SimulaLabs, Hewlett-Packard
- Active in OpenSource and Java Communities
  - Jetty, Cometd, MX4J, Foxtrot, Livetribe, etc.
  - Co-Leader of Java User Group Torino, Italy
- Currently working on:
  - Comet server-side and client-side applications
    - Clients for browsers, J2ME and Android
  - Server-side asynchronous IO and protocols
Agenda

- What are Comet web applications
- Impacts of Comet web applications
- The CometD project
- Demo
- Questions & Answers
What are Comet Web Applications
Web Apps History

Web Classic Interaction
Web Apps History

- Web Classic Interaction

- Request Pattern
  - Bursts of requests for HTML, images, CSS, JS

- Navigation Mode
  - Full page based

- Interaction with Server
  - Reactive, changes happen on user click
  - Resources download
Web Apps History

Web Dynamic HTML Interaction
Web Apps History

- Web Dynamic HTML Interaction

- Request Pattern, Navigation Mode, Interaction with Server
  - Same as before

- Uses JavaScript
  - Input validation
  - DOM manipulation
  - Local UI effects
Web Apps History

- Web Classic + XMLHttpRequest
Web Apps History

- Web Classic + XHR

- Request Pattern: **changed**
  - Bursts (for classic) + Concurrent (for XHR)
  - Requires synchronization in server code

- Navigation Mode: **radically changed**
  - Sometimes full page, most often single page with partial changes

- Interaction with Server: **changed**
  - Reactive as before
  - Data download
Server-side Events

- Heavy usage of client-side JavaScript and XHR changed the way we create and develop webapps
  - They become rich, and raise expectations

- Traditionally, web experience was driven by the client
  - Can webapps now be also driven by server-side events?
Server-side Event-driven **Polling** Web Application
Polling Strategy for Server-side Events Notification

- Simple to implement ("I can do that!")
- Sensible latency for event notifications

Can look like a denial of service attack to the poor server

- When poll interval is short to reduce event latency
- When the number of clients is large

We can do better!
Server-side Events

- Server-side Event-driven Comet Web Application
- Comet Strategy for Server-side Events Notification
  - Difficult to implement right
  - Minimal latency for event notification

- One request (the “long poll”) is held by the server until:
  - A server-side events arrives
  - A timeout expires
  - The client disconnects
Impacts on the Server
Polling vs Comet

What are the impacts of the polling and comet models on servers?

Polling
- 1000 clients, each polling every 5 seconds
- Assume 100 ms poll processing time
- Yields 200 requests/s, 20 concurrent requests
- 20 threads x 1 MB stack size = 20 MB

Limits
- Most likely, the server is CPU bound, then connection bound
Polling vs Comet

Comet (Classic)
- 1000 clients, long poll timeout 20 seconds
- Yields 1000 concurrent requests!
- 1000 x 1 MB stack size = 1 GB

Limits
- Most likely, the server is memory bound
- Note that stack memory is outside Java heap
Polling vs Comet

- Comet has huge impacts on server-side

- You cannot just deploy your comet application in a normal configuration

- You cannot deploy your comet application behind Apache Httpd
  - Does not scale for the same reasons

- You need a new generation of servers
Greg Wilkins was the first to explore these problems

He created the **Jetty Continuations** which allow the Jetty server to scale Comet applications

The continuation concept has been incorporated in the new Servlet 3.0 specification

Jetty 6 and Jetty 7 successfully deployed Comet applications worldwide

Jetty 8 will implement Servlet 3.0
Server-side Event-driven Comet Web Application
Server-side Event-driven **Comet** Web Application
Polling vs Comet

- Comet (Continuations)
  - 1000 clients, long poll timeout 20 seconds
  - Assume 160 ms processing time
    - Request is run twice
  - Yields 50 requests/s, 8 concurrent requests
  - 8 x 1 MB stack size = 8 MB

- Limits
  - Most likely, the server is connection bound, then CPU bound
  - Scales better than normal polling
With Comet, we have an asynchronous bidirectional web
- Looks like messaging to me

Writing server-side code based on continuations (or, in the future, with Servlet 3.0) is difficult
- It really is, you don't want to do it

Web applications should be easy to write
- Can we abstract the gory details into a library?
The CometD Project
The CometD Project

- We have now a scalable bidirectional web
- What do we need to write applications?
- We don't want to care about long-polling the server, respecting possible constraints
  - In browsers, the same-origin policy and the two connection limit
- We want:
  - A clean way to publish data to the server
  - A clean way to receive data from the server
  - A clean way to “emit” server-side events to clients
The CometD Project

- There are a lot of other details to take care of
  - Authentication
  - Network Failures
    - With possible automatic retry
  - Message Batching
  - Message Acknowledgment
  - Etc.

- From these and other requirements and input, the CometD project was born
The CometD Project

The CometD project delivers libraries that use the Comet technique (long poll) to transport Bayeux messages.

The Bayeux protocol specifies the format of the Bayeux messages:
- It is based on JSON.

Libraries are available in:
- JavaScript (client)
- Java (client and server)
- Perl & Python (less active)
Bayeux is at its core a publish/subscribe messaging system
  - Very similar to JMS

A Bayeux channel is a “topic” you may subscribe interest to
  - And you will be delivered messages published onto

You can publish messages to a channel
  - The server automatically delivers the messages to all channel subscribers
var cometd = $.cometd; // jQuery style

cometd.init('http://myserver/cometd');

cometd.subscribe('/my/channel', function(message)
{
    var data = message.data;
    // Do something with the data
});

cometd.publish('/my/channel', {
    userId: 1,
    chatText: 'hello!' 
});

cometd.disconnect();
The JavaScript library features

- Common code with bindings for Dojo and jQuery
- Highly configurable
- Message batching
- Automatic reconnection
- Pluggable transports
  - Long Polling and Callback Polling available
- Supports Cross-Origin servers
- Extensible
  - Many extensions already available
The Java library features

- Highly scalable (the client is based on Jetty asynchronous HTTP Client)
- Message batching
- Lazy messages
- A variety of listeners to be notified of relevant events in client and server
- Data filters to automatically convert data
- Extensions
- SecurityPolicy
Performance

Clients vs Latency vs Throughput

Latency (ms)

Clients (circle diameter indicates throughput 0-3800/s)

- 50% CPU, 100 Rooms
- 90% CPU, 100 Rooms
- 50% CPU, 1000 Room
- 90% CPU, 1000 Rooms
- 90% CPU, 1000 Rooms, multiple client simulators
DEMO
Questions & Answers