LUXOFT

ENGINEERING BUSINESS PERFORMANCE

Comet, Ajax and web apps performance

Paweł Limanówka

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Online gaming app

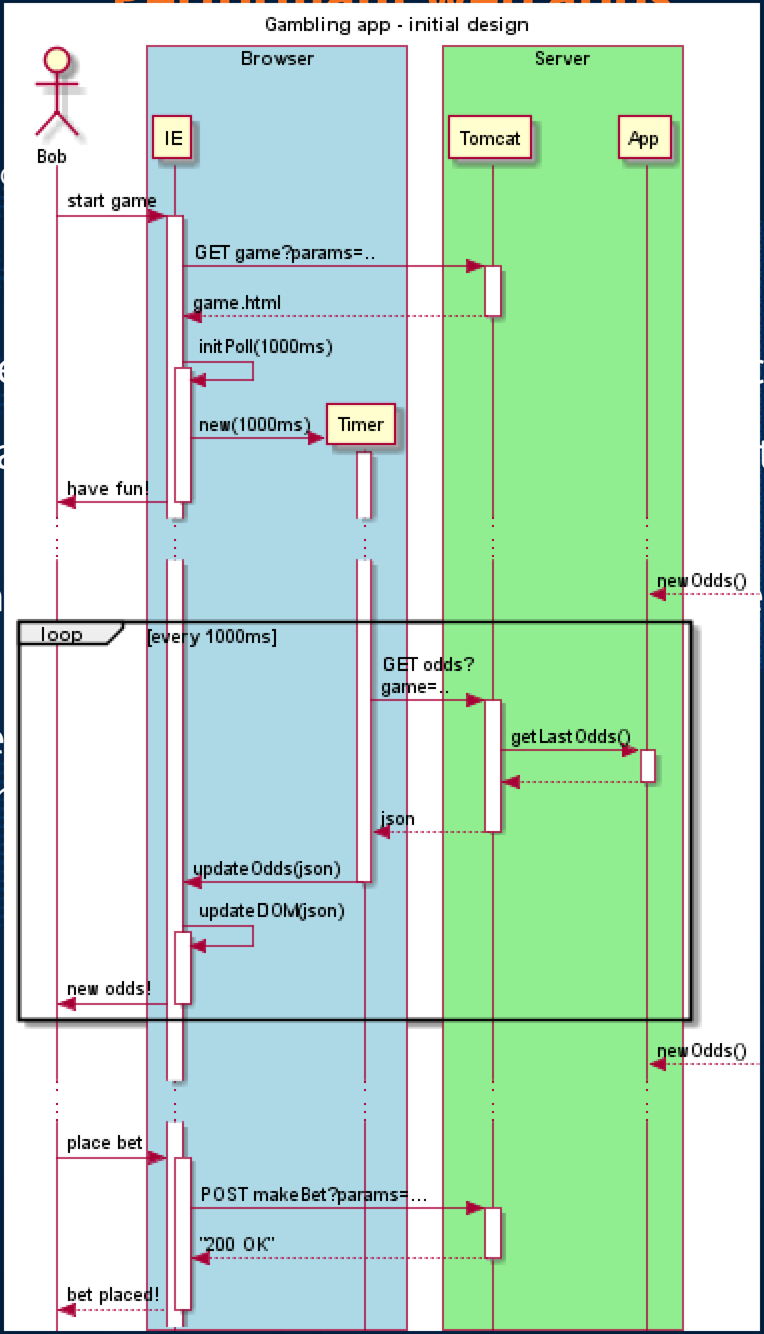
The Requirements:

- Allow placing bets for any of 15 instruments
- Calculate new odds on source data change/bet placement
- Push new odds with latency less than 200ms
- Support up to 5000 simultaneous players
- Do all that on „commodity-hardware” (<3000€ set up, <200 € monthly)

Performant web apps

Ignorance is a bless..

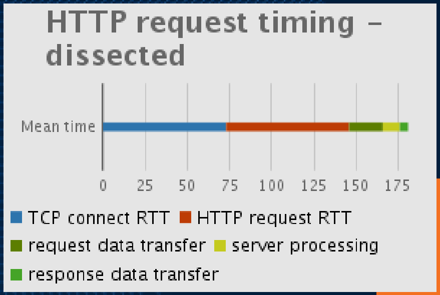
- Standard web apps use … ations scheme
- This will scale like a cha … toss few additional servers” solution
- Unfortunately, this is n … nt updates to 100’s or 1000’s of on-line users
- And, if you are not alre … ’t find out about that sad fact until you deploy th



LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Comet client side
HTTP request anatomy

- Meaningful load-testing of web apps is non-trivial task, but even simple / micro-benchmark load tests done under real-live network conditions can save the day
- Using simple polling (pure req-resp pattern) for updates delivery is far from optimal
  - For simple messages most of the latency comes from client-server connection RTT parameter
  - If messages are not jumbo-sized, server-side optimizations can save only few % off the latency
  - Most likely the second biggest factor contributing to latency is HTTP request transfer time
    most ppl uses ADSL connections, with relatively slow up-link
- First and most popular Comet technique was born from those observations…



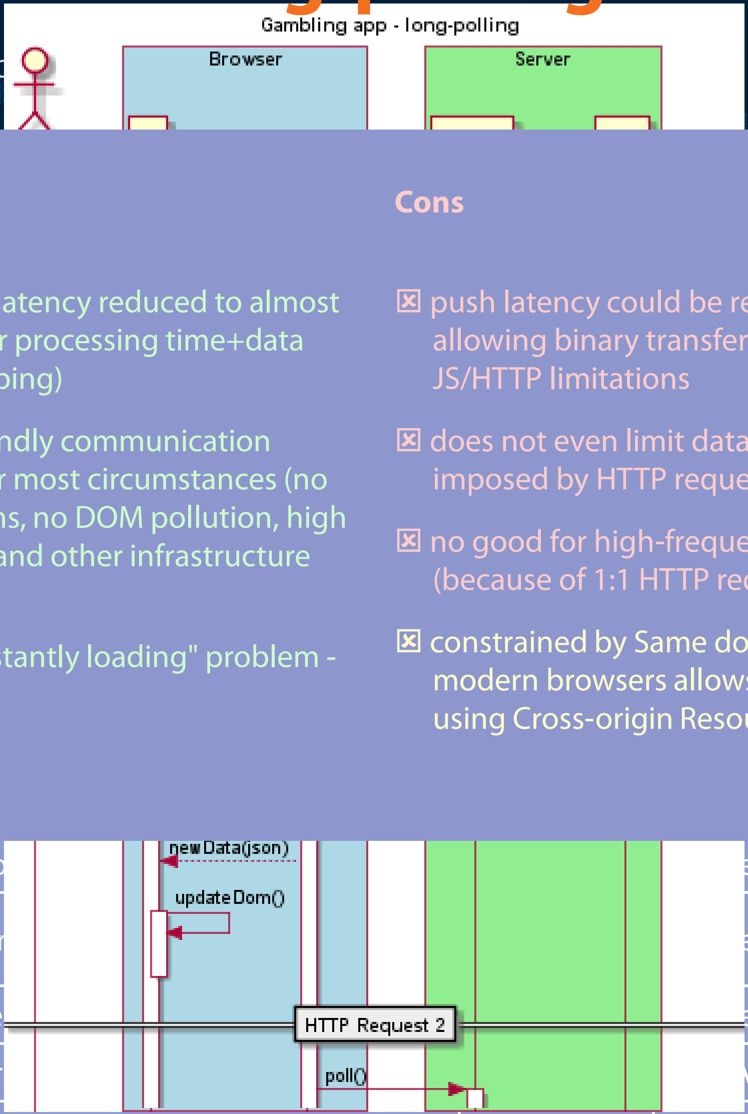
Phase	Mean time
TCP connect RTT	73 ms
HTTP rq RTT	73 ms
Request transfer	20 ms
Server processing	10 ms
Response tranfer	5 ms

```
Host
User-Agent: Mozilla/5.0 (Windows
rv:1.9.1.2) Gecko/20090729 Firefox/3.5
Accept: */*\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
If-Modified-Since: Mon, 31 Aug 2009 17:13:58 GMT\r\n
\r\n
\r\n
```

Comet client side long-polling

Long-po

- It doe … penalty to tim
- For m … hanges on th
- Don't … rowser comp
- Doze



Pros

☑ server-to-client push latency reduced to almost bare minimum (server processing time+data transfer time+half of ping)

☑ browser and web-friendly communication scheme - works under most circumstances (no long-lived connections, no DOM pollution, high tolerance for proxies and other infrastructure elements)

☑ eliminates "page constantly loading" problem - if implemented well

Cons

☒ push latency could be reduced further by allowing binary transfers - impossible due to JS/HTTP limitations

☒ does not even limit data transfer overhead imposed by HTTP request/response headers

☒ no good for high-frequency messaging (because of 1:1 HTTP request to message ratio)

☒ constrained by Same domain policy (most modern browsers allows to work around that using Cross-origin Resource Sharing)

TCP co		efore
HTTP r		efore
Reque		efore
Server		After
HTTP response RTT; server to client trip	40 ms	After
Response tranfer	5 ms	After

ENGINEERING BUSINESS PERFORMANCE

Comet client side callback-polling

„Creative workaround" to allow long-polling from domains other main page domain:

- Uses „a JS hack" to break Same Domain policy restrictions of long-polling
- Build on a fact that URL specified for <script> src element can point to an arbitrary domain
- All br load is comp
- Serve JS functi

Unfortur

- Speci)

Pros	Cons
☑ server-to-client push latency reduced to almost bare minimum (server processing time+data transfer time+half of ping)	☒ push latency could be reduced further by allowing binary transfers - impossible due to JS/HTTP limitations
☑ browser and web-friendly communication scheme - works under most circumstances (no long-lived connections, no DOM pollution, high tolerance for proxies and other infrastructure elements)	☒ does not even limit data transfer overhead imposed by HTTP request/response headers
☑ eliminates "page constantly loading" problem - if implemented well	☒ no good for high-frequency messaging (because of 1:1 HTTP request to message ratio)
☑ allows to break out of same origin policy allowing for greater flexibility of deployment architecture (separation of Comet servers from standard business logic servers)	☒ additional <script> elements and JS function envelope can add significant overhead to the data stream
	☒ may force you to reload pages periodically - to avoid out of memory errors

ENGINEERING BUSINESS PERFORMANCE

Comet client side streaming transports

Family of transport schemes sharing same basic concept:

- Single HTTP connection, lasting for the time of whole user session
- Server pushes data incrementally, as they became available, but never closes the connection

Advantages:

- Eliminates HTTP request transfer overhead
- Allows for server-side optimizations – less resources per client
- Since single push means transmitting bare data only, it allows pushing signals at much higher frequencies

Common problems:

- Consumes 1 connection from the available connections pool (HTTP 1.1)
- Requires server-side optimizations – NIO is a must!
- Does not play very well with internet infrastructure and is sensitive to inter-browser compatibility problems (a lot of exception handling on the client side)

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Comet client side
iFrame streaming

Combines „hidden iFrame” with „a JS hack” similar to one used in callback-polling:

- Page contains JS data handler and an inline, hidden *iframe*
- Server sends an iframe content as *chunked block*
- On event, server pushes *script* tag with callback function and data as static values
- Since HTML is rendered incrementally, JS callback executes immediately

Pros	Cons
☑ all browsers, even IE6, supports it	☒ additional <script> elements and JS function envelope can add significant overhead to the data stream
	☒ constantly changing iFrame DOM will cause out-of-memory errors unless periodic reloads are used
	☒ quirks'n'tricks required on per-browser basis (i.e.: some IE versions won’t notice iFrame content until they receive at least 256 bytes)

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Comet client side
XHR streaming

Most popular scheme, implemented using XMLHttpRequest API:

- JS code within the page uses *XHR* object to open streaming connection
- Server suspends the connection right after it's established (multipart/x-mixed-replace MIME type is used)
- On new signal, server pushes data to the connection and than suspends it again
- When data arrives on the client, *XHR.onreadystatechange* callback is invoked

Pros	Cons
☑ no "page constantly loading" problems	☒ some old but still popular browsers (IE 6) does not support it
☑ can stream data packets only - no overhead	☒ quirks'n'tricks required on per-browser basis (i.e.: for some Safari versions one has to emit some junk prepended to actual message - browser can see new content only if >256bytes were transferred)
☑ cleaner and easier to debug JS transport code	☒ streamed data accumulate on the browser side (*XHR.responseText* always contains THE WHOLE so far received server response) thus requires periodic page reloads

Comet client side
server-sent events

Standardized API designed specifically for pushing data:

- Handled by browser – just create *EventSource* and add some listeners to it
- Designed specifically for the task – no memory leaks, good inter-browser compatibility (except IE, naturally ☺)
- Relatively „net infrastructure friendly" (all traffic is pure HTTP, keep-alive at protocol level available)

Pros	Cons
☑ easy to use, clean solution	☒ IE does not support it (as well as older browsers in phones/TV's/etc. - check out this link http://www.eventsourcehq.com/browser-support)
☑ A lot more stable than iFrame/XHR	☒ some headers overhead added to data (not much, but with frequent updates..)
☑ uses HTTP (standard ports, etc) - net infrastructure firendly	

Comet client side
web sockets

Standardized API designed for efficient, two-way communication:

- Handled by browser
- Designed for two way communication with low latency
- Allows for pure binary transfers
- May be blocked by net infrastructure, won’t work for users connecting from most intranets

Pros	Cons
☑ easy to use, clean and really fast	☒ few browsers support it
☑ two way communication	☒ Net infrastructure unfriendly (Layer7 infrastructure elements, intranets, etc)
☑ Allows for binary data transfers	

Server side story
standard servlets

Simple and easy (recipe for disaster):

- Simple, well known API; „no sweat" coding, ready to roll in no time – but really?
- Default config limits are there for a reason
- Blocking hread
  - OOM c
  - Heap s
  - OOM c
  - some (
- Last but r em-copied at least once

	Param	Desc	Def.
Apache	MaxClients	Max no of connections tah will be processed simultaneously	256
	ServerLimit*ThreadsPerChild	Max no of processes and no of threads created by each process	400 (16*25)
Tomcat	maxThreads	Max no of request processing threads	200
	acceptsCount	Max no of connections queued for processing	100
Sys	ip_conntrack_max ☺		various

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Server side story
non blocking IO

Java NIO API, introduced with Java 1.4, helps to overcome those problems:

- It's block oriented – allows for bit faster implementations
  especially when pushing same message to many users
- Non-blocking reads and writes with selectors – no more thread per connection ☺
- You can allocate *direct* buffer and send it multiple times – in practice going down from 2 mem-copy's to 1 per connected user
- It's much easier to implement SLA checking or enforcing for broadcasts

But it has it's drawbacks too

- It's not so straightforward to use as a standard blocking IO – minor issue
- Can mean writing HTTP impl if you are using older server – bigger issue
  (yeah, I've heard about Apache commons, but try to implement HTTPS on your own)

Quick micro-benchmarking proved that this is a right path to follow…

But without reinventing the wheel, if possible..

ENGINEERING BUSINESS PERFORMANCE

Server side story

Tomcat NIO adapter

Fortunately, NIO was 5 years old ☺:

-
  ```
public interface CometProcessor extends Servlet {
```

Param	Desc	Def. / My
acceptorThreadCount	Number of threads used to accept new connections; should be less or equal to number of available cores.	1 / 1
pollerThreadCount	Number of threads to be used for polling events - that is, checking for new data that clients might have sent.	1 / 1
maxThreads	Maximum number of worker threads - threads that will run application-supplied handlers to actually handle incoming data.	200 / 200
processorCache	Max number of Http11NioProcessor objects that will be cached in memory (to speed up things) - should be close to maxThreads value.	200 / 200
socket.rxBufSize	Size (in bytes) of receive buffer.	25188 / 512
socket.txBufSize	Size (in bytes) of transmit buffer.	8192 / 512
socket.directBuffer	true/false - if set to true, tomcat will use direct buffers, potentially speeding things up (1 mem-copy less)	false / true

```
    org.apache.tomcat.sendfile.start
    org.apache.tomcat.sendfile.end
```

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Web apps load testing

Setting up right environment for load tests is not a trivial task; you should consider putting your server(s) behind one or more proxies:

- For bandwidth and latency emulation you can try Netem/MasterShaper/WANem or dedicated filtering-bridge machine (http://www.freebsd.org/doc/en/articles/filtering-bridges/article.html)
- Don’t forget about simulating proxies performing NAT and tossing in at least one Apache-based proxy with most popular configuration you can find (to test against things like long-lived connection dropping, etc.)
- Don’t settle on using micro-benchmarks or replaying pre-recorded browsers sessions from handful of test machines. For realistic load testing you should try to use as many stand-alone machines located outside of your simulated network as possible; consider encapsulating JMeter/Grinder/YourFavoriteTestTool as Java Web Start app and running it on „friends machines”..
- If budget and time constraints allows, maybe you should even try one of the load-testing services out there (i.e. http://www.soasta.com/ , http://loadstorm.com/ , http://blazemeter.com/ , http://www.acutest.co.uk)

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Don’t promise too much

As a warning I would like to share with you one funny story:

Remember about non-func requirement I promised to fulfil – that all connected users will get their odds within 200ms from calculation finish timestamp on the server?

Well, The Client took it very seriously:

- When visiting friend in Auckland, I’ve noticed that odds seems to be lagging. Quick check using the debug panel showed that odds latency is above 200ms. When you’ll deploy the fix?

Hmm, it took me really long time to convince him not to sue me.. As I cannot fulfil this requirement:

- Distance between our server room and Auckland was about 17 300km
- Given that speed of light in fiber is around 200 000 km/s, *the shortest* possible signal propagation time (one way) was about 86.5ms
- Unless one uses s-f hardware to measure one-way trip time accurately, one cannot see RTT values less than about 170ms + some time for h/w along the path => RTT should be well over 200ms in any possible case
- So, it’s physically impossible to guarantee latency < 200ms under any circumstances

LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Thank You!

http://blog.luxoft.com

plimanowka@gmail.com