Server Side Story

An asynchronous ballet

abstract

EuroPython 2013

Simone Deponti - simone.deponti@abstract.it

Who am I

Python web developer since 2006
Work at Abstract as web developer
Plone contributor
Do more Javascript than I'm willing to admit

Am I in the wrong room?

This talk will focus on:

- Where web development comes from
- Challenges of real-time applications
- Patterns and tools to deal with that

CODE SAMPLE FREE! (ALMOST)

A LITTLE BIT OF HISTORY

Let there be <a> link

The web was born as:

- Simple document retrieval
- Interlinking of documents

Have you seen?

Next step:

- Searching documents
- Query parameters
- POST and Forms
- CGI
- We start having stuff that's not documents

Statefulness

HTTP is stateless

- stateless isn't good for applications
- cookies add statefulness

Javascript

Executing code on the client was needed:

- provisions for scripting were there
- we lacked a language and a way to interact with the document
- Javascript and the DOM came

AJAX

Javascript was limited to mangling the DOM

- AJAX added the ability to make requests "on the side"
- One-page applications
- State lives on the client, too!

CURRENT SITUATION

Web applications

So far:

- the server is the master
- all data lives there
- the client requests stuff and gets a reply

Web applications (server side)

Every "classic" web app does this on every request:

- pull up the state
- process the request
- assemble a response
- save the new state and return it back

I've got something to say!

The server can speak only when asked.
This maps poorly on these cases:

- Notifications
- Chat systems
- *Real time* applications

SOLUTIONS

Polling

Polling is the simplest solution, and the worst

- You keep annoying the server
- Wasting memory, CPU, bandwidth on both sides just to annoy a component of your stack
- Logarithmic polling isn't a solution

COMET

Also called long polling

- Still based on request-reply
- The server replies very slowly, allowing time to elapse and hoping that something happens

BOSH

The formal way to do COMET, as done by XMPP:

- Client starts a long-polling connection
- If something happens on the client side, client sends a second request, server replies on the first and closes, second remains open
- Before expiration time, server sends empty message and closes, client reopens connection.

http://xmpp.org/extensions/xep-0124.html#technique

You might not be able to deal with this

If your execution model calls for one thread/process per request, you're out of luck.
You must use an asynchronous server.

Websockets

A new protocol

- a bidirectional connection tunneled through HTTP (similar to a raw TCP connection)
- HTTP 1.1 has provisions for this
- Sadly, it's the part of the protocol where many implementors got bored

http://tools.ietf.org/html/rfc6455

Websockets in Python

```
from geventwebsocket.handler import WebSocketHandler
from gevent.pywsgi import WSGIServer
[...]

@app.route('/api')
def api():
   if request.environ.get('wsgi.websocket'):
       ws = request.environ['wsgi.websocket']
       while True:
           message = ws.wait()
           ws.send(message)
   return

if __name__ == '__main__':
   server = WSGIServer([...],handler_class=WebSocketHandler)
   [...]
```

https://gist.github.com/lrvick/1185629

Asyncronous I/O

Is hard.

- Processes and traditional threads don't scale for real-time apps
- Callback based systems are one solution
- Green threads are another one

In Python, there is no clean and easy solution.

http://stackoverflow.com/a/3325985/967274

Tulip (PEP 3156)

Tries to provide a common groud for all frameworks targeting asyncronous I/O

- Has a pluggable event loop interface
- Uses futures to abstract callbacks
- Allows using callbacks or coroutines (via yield from)
- Uses the concept of transports and protocols

Tulip (PEP 3156)

Pros and cons

- Tulip isn't simple or straightforward
- Because of its constraints
- Reinventing the whole Python ecosystem ain't an option
- Tulip is a library, frameworks can build on top of it
- Planned to land in 3.4
- Conveniently wraps parts of the standard library

Tulip and websockets

Tulip supports websockets

- Has an example in the source, solving the broadcaster use case
- The example is fairly complete (and complex)

http://code.google.com/p/tulip/source/browse/examples/wssrv.py

Architectural considerations

Most of the quick fixes you're used to won't work

- Caching can't be layered over the frontend to mask problems
- Code flow must receive extra care
- You still need to deal with an awful lot of synchronous calls, and orchestrate them with asynchronous calls

http://python-notes.boredomandlaziness.org/en/latest/pep_ideas/async_programming.html

Shar(d)ed state

Real time applications behave like clusters

- State is sharded between nodes
- You must orchestrate and deal with inconsistencies
- People doing clusters (or cloud systems) have already some theoretical work done

Scaling

Once you're able to manage the shared state, scaling becomes easier and linear. However, due to the nature of the control flow, it will cost more than with non-realtime web applications.

Security considerations

Websockets have security provisions to avoid blatant abuses.
However:

- Authentication and authorization are delegated to the application
- Statefulness is a double-edged sword
- The protocol itself is fairly new and underdeployed

Security considerations (2)

Websockets use the origin model

- These provisions are relevant for browsers only
- Intended to impede browser hijacking
- RFC clearly states that validation of data should always be performed on both sides
- Resist the urge to allow for arbitrary objects to be passed between server and client

Conclusions

- Real time applications require a fundamental change of paradigm
- This paradigm change spawned a new protocol on the client side
- The server side should abandon the standard model and embrace event based asynchronous systems
- In python we already have tools to deal with these challenges, but the landscape is fragmented

Conclusions (2)

- Python 3.4 will (hopefully) lay a common layer to address these architectural problems
- There are fundamental architectural changes that the paradigm brings to web applications, making it almost equal to native applications
- The area still needs to mature to fully expose challenges especially in the area of security

Questions?

Simone Deponti
simone.deponti@abstract.it
@simonedeponti

abstract
Home
Servizi
Portfolio
News
Blog
Abstract
PLOG2013
Nuovo portale per Videohifi.com
Maire rinnova il suo Portale
Abstract annuncia il Plone Open Garden 2013
linkcheck - per u
indicizzazione d