Distributed Web Applications

• WWW principles

• Case Study: web caching as an illustrative example
  – Invalidate versus updates
  – Push versus Pull
  – Cooperation between replicas

Traditional Web-Based Systems

• Client-server web applications

1. Get document request (HTTP)

2. Server fetches document from local file

3. Response
Web Browser Clients

- The logical components of a Web browser.

The Apache Web Server

- The general organization of the Apache Web server.
Proxy Servers

- Using a Web proxy when the browser does not speak FTP (or for caching and offloading)

Multitiered Architectures

- Three tiers: HTTP, application, and database tier
Web Server Clusters

- Clients connect to front-end dispatcher, which forwards requests to a replica (recall discussion from Cluster scheduling)
- Each replica can be a tiered system
  - For consistency, database can be a common/non-replicated

Web Server Clusters (2)

- A scalable content-aware cluster of Web servers.
Web Clusters

• Request-based scheduling
  – Forward each request to a replica based on a policy

• Session-based scheduling
  – Forward each session to a replica based on a policy
• Scheduling policy: round-robin, least loaded

• HTTP redirect vs TCP splicing vs TCP handoff

Elastic Scaling

• Web workloads: temporal time of day, seasonal variations
  – Flash crowds: black friday, sports events, news events
• Overloads can occur even with clustering and replication
• Elastic scaling: dynamically vary application capacity based on workload (aka auto-scaling, dynamic provisioning)
• Two approaches:
  – Horizontal scaling: increase or decrease # of replicas based on load
  – Vertical scaling: increase or decrease size of replica (e.g., # of cores allocated to container or VM) based on load
  – Proactive versus reactive scaling
    – Proactive: predict future load and scale in advance
    – Reactive: scale based on observed workload
• Common in large cloud-based web applications
Micro-services Architecture

• Micro-services: application is a collection of smaller services
  • Example of service-oriented architecture
  • Modular approach to overcome “monolith hell”

• Each microservice is small and can be maintained independently of others

• Each is independently deployable

• Clustering and auto-scaling can be performed independently

Scaling Web applications

• Three approaches for scaling

https://microservices.io/articles/scalecube.html
Web Documents

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Plain</td>
<td>Unformatted text</td>
</tr>
<tr>
<td></td>
<td>HTML</td>
<td>Text including HTML markup commands</td>
</tr>
<tr>
<td></td>
<td>XML</td>
<td>Text including XML markup commands</td>
</tr>
<tr>
<td>Image</td>
<td>GIF</td>
<td>Still image in GIF format</td>
</tr>
<tr>
<td></td>
<td>JPEG</td>
<td>Still image in JPEG format</td>
</tr>
<tr>
<td>Audio</td>
<td>Basic</td>
<td>Audio, 8-bit PCM sampled at 8000 Hz</td>
</tr>
<tr>
<td></td>
<td>Tone</td>
<td>A specific audible tone</td>
</tr>
<tr>
<td>Video</td>
<td>MPEG</td>
<td>Movie in MPEG format</td>
</tr>
<tr>
<td></td>
<td>Pointer</td>
<td>Representation of a pointer device for presentations</td>
</tr>
<tr>
<td>Application</td>
<td>Octet-stream</td>
<td>An uninterpreted byte sequence</td>
</tr>
<tr>
<td></td>
<td>Postscript</td>
<td>A printable document in Postscript</td>
</tr>
<tr>
<td></td>
<td>PDF</td>
<td>A printable document in PDF</td>
</tr>
<tr>
<td>Multipart</td>
<td>Mixed</td>
<td>Independent parts in the specified order</td>
</tr>
<tr>
<td></td>
<td>Parallel</td>
<td>Parts must be viewed simultaneously</td>
</tr>
</tbody>
</table>

- Six top-level MIME types and some common subtypes.

HTTP Connections

- Using nonpersistent connections.
HTTP 1.1 Connections

(b) Using persistent connections.

HTTP Methods

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>Request to return the header of a document</td>
</tr>
<tr>
<td>Get</td>
<td>Request to return a document to the client</td>
</tr>
<tr>
<td>Put</td>
<td>Request to store a document</td>
</tr>
<tr>
<td>Post</td>
<td>Provide data that are to be added to a document (collection)</td>
</tr>
<tr>
<td>Delete</td>
<td>Request to delete a document</td>
</tr>
</tbody>
</table>

Operations supported by HTTP.
HTTP 2.0

- Http 1.1 allows pipelining over same connection
  - Most browsers do not use this feature
- HTTP v2: Designed to reduce message latency
  - No new message or response types
- Key features
  - Binary headers (over text headers of http 1.1)
  - Uses compression of headers and messages
  - Multiplex concurrent connection over same TCP connection
    - each connection has multiple “streams”, each carrying a request and response
      - No blocking caused by pipelining in http 1.1

See https://developers.google.com/web/fundamentals/performance/http2/

Web Services Fundamentals

- The principle of a Web service.
Simple Object Access Protocol

• An example of an XML-based SOAP message.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

RESTful Web Services

• SOAP heavy-weight protocol for web-based distributed computing
  – RESTful web service: lightweight, point-to-point XML comm

• REST=representative state transfer
  – HTTP GET => read
  – HTTP POST => create, update, delete
  – HTTP PUT => create, update
  – HTTP DELETE => delete

• Simpler than RPC-style SOAP
  – closer to the web
RESTful Example

GET /StockPrice/IBM HTTP/1.1
Host: example.org
Accept: text/xml
Accept-Charset: utf-8

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<s:Quote xmlns:s="http://example.org/stock-service"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <s:TickerSymbol>IBM</s:TickerSymbol>
  <s:StockPrice>45.25</s:StockPrice>
</s:Quote>

Corresponding SOAP Call

GET /StockPrice HTTP/1.1
Host: example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:s="http://www.example.org/stock-service">
  <env:Body>
    <s:GetStockQuote>
      <s:TickerSymbol>IBM</s:TickerSymbol>
    </s:GetStockQuote>
  </env:Body>
</env:Envelope>

HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:s="http://www.example.org/stock-service">
  <env:Body>
    <s:GetStockQuoteResponse>
      <s:StockPrice>45.25</s:StockPrice>
    </s:GetStockQuoteResponse>
  </env:Body>
</env:Envelope>
SOAP vs RESTful WS

- Language, platform and transport agnostic
- Supports general distributed computing
- Standards based (WSDL, UDDI dir. service...)
- Built-in error handling
- Extensible
- More heavy-weight
- Harder to develop

- Language and platform agnostic
- Point-to-point only; no intermediaries
- Lack of standards support for security, reliability (“roll your own”)
- Simpler, less learning curve, less reliance on tools
- Tied to HTTP transport layer
- More concise