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>HTML</td>
<td></td>
<td>Text including HTML markup commands</td>
</tr>
<tr>
<td>XML</td>
<td></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


Corresponding SOAP Call

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


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

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 you own")
- Simpler, less learning curve, less reliance on tools
- Tied to HTTP transport layer
- More concise

Web Proxy Caching

1. Look in local cache
2. Ask neighboring proxy caches
3. Forward request to Web server

- HTTP Get request
- The principle of cooperative caching.
Web Caching

- Example of the web to illustrate caching and replication issues
  - Simpler model: clients are read-only, only server updates data

Consistency Issues

- Web pages tend to be updated over time
  - Some objects are static, others are dynamic
  - Different update frequencies (few minutes to few weeks)
- How can a proxy cache maintain consistency of cached data?
  - Send invalidate or update
  - Push versus pull
Push-based Approach

- Server tracks all proxies that have requested objects
- If a web page is modified, notify each proxy
- Notification types
  - Indicate object has changed [invalidate]
  - Send new version of object [update]
- How to decide between invalidate and updates?
  - Pros and cons?
  - One approach: send updates for more frequent objects, invalidate for rest

Push-based Approaches

- Advantages
  - Provide tight consistency [minimal stale data]
  - Proxies can be passive
- Disadvantages
  - Need to maintain state at the server
    - Recall that HTTP is stateless
    - Need mechanisms beyond HTTP
  - State may need to be maintained indefinitely
    - Not resilient to server crashes
Pull-based Approaches

- Proxy is entirely responsible for maintaining consistency
- Proxy periodically polls the server to see if object has changed
  - Use if-modified-since HTTP messages
- Key question: when should a proxy poll?
  - Server-assigned Time-to-Live (TTL) values
    - No guarantee if the object will change in the interim

Pull-based Approach: Intelligent Polling

- Proxy can dynamically determine the refresh interval
  - Compute based on past observations
    - Start with a conservative refresh interval
    - Increase interval if object has not changed between two successive polls
    - Decrease interval if object is updated between two polls
    - Adaptive: No prior knowledge of object characteristics needed
Pull-based Approach

• Advantages
  – Implementation using HTTP (If-modified-Since)
  – Server remains stateless
  – Resilient to both server and proxy failures

• Disadvantages
  – Weaker consistency guarantees (objects can change between two polls and proxy will contain stale data until next poll)
    • Strong consistency only if poll before every HTTP response
  – More sophisticated proxies required
  – High message overhead

A Hybrid Approach: Leases

• Lease: duration of time for which server agrees to notify proxy of modification
• Issue lease on first request, send notification until expiry
  – Need to renew lease upon expiry
• Smooth tradeoff between state and messages exchanged
  – Zero duration $\Rightarrow$ polling, Infinite leases $\Rightarrow$ server-push
• Efficiency depends on the lease duration
Policies for Leases Duration

• Age-based lease
  – Based on bi-modal nature of object lifetimes
  – Larger the expected lifetime longer the lease

• Renewal-frequency based
  – Based on skewed popularity
  – Proxy at which objects is popular gets longer lease

• Server load based
  – Based on adaptively controlling the state space
  – Shorter leases during heavy load

Cooperative Caching

• Caching infrastructure can have multiple web proxies
  – Proxies can be arranged in a hierarchy or other structures
    • Overlay network of proxies: content distribution network
  – Proxies can cooperate with one another
    • Answer client requests
    • Propagate server notifications
Hierarchical Proxy Caching

Examples: Squid, Harvest

Locating and Accessing Data

Properties
- Lookup is local
- Hit at most 2 hops
- Miss at most 2 hops (1 extra on wrong hint)
Edge Computing

- Web caches effective when deployed close to clients
  - At the “Edge” of the network
- Edge Computing: paradigm where applications run on servers located at the edge of the network
- Benefits
  - Significantly lower latency than “remote” cloud servers
  - Higher bandwidth
  - Can tolerate network or cloud failures

- Complements cloud computing
  - Cloud providers offer edge servers as well as cloud servers

Edge Computing Origins

- Origins come from mobile computing and web caching
- Content delivery networks
  - Network of edge caches deployed as commercial service
  - Cache web content (especially rich content: images, video)
  - Deliver from closest edge proxy server

- Mobile computing
  - Devices have limited resources, limited battery power
  - Computational offload: offload work to more capable edge server
  - Low latency offload important for interactive mobile applications
  - Edge server sends results to the mobile
Content Delivery Networks

- Global network of edge proxies to deliver web content
  - edge clusters of varying sizes deployed in all parts of the world
  - Akamai CDN: 120K servers in 1100 networks, 80 countries
- Content providers are customers of CDN service
  - Examples: news sites, image-rich online stores, streaming sites
  - Decide what content to cache/offload to CDN
  - Embed links to cdn content: `http://cdn.com/company/foo.mp4`
  - Consistency responsibility of content providers
- Users access website normally
  - Some content fetched by browser from CDN cache

CDN Request Routing

- Web request need to be directed to nearby CDN server
- Two level load balancing
  - Global: decide which cluster to use to serve request
  - Local: decide which server in the cluster to use
- DNS-based approach is common
  - Special DNS server: resolve `www.cnn.com/newsvideo.mp4`
  - DNS checks location of client and resolves to IP address of nearby CDN server
  - Different users will get resolved to different edge locations
CDN Issues

• Which proxy answers a client request?
  – Ideally the “closest” proxy
  – Akamai uses a DNS-based approach

• Propagating notifications
  – Can use multicast or application level multicast to reduce overheads (in push-based approaches)

• Active area of research
  – Numerous research papers available

CDN Request Processing

• The principal working of the Akamai CDN.
CDN Hosting of Web Applications

Figure 12-21. Alternatives for caching and replication with Web applications.

Mobile Edge Computing

- Use case: Mobile offload of compute-intensive tasks
- Example: augmented reality, virtual reality (mobile AR/VR)
  - mobile phone or headset has limited resources, limited battery
  - Low latency / response times for interactive use experience
  - mobile devices may lack a GPU or mobile GPU may be limited

- Today’s smartphones are highly capable (multiple cores, mobile GPU, neural processor)
  - mobile offload more suitable for less capable devices (e.g., AR headset)
- 5G cellular providers: deploy edge servers near cell towers
  - industrial automation, autonomous vehicles
Edge Computing Today

• Emerging approach for latency-sensitive applications
• Edge AI: run AI (deep learning) inference at edge
  – home security camera sends feed, run object detection
• Low latency offload: autonomous vehicles, smart city sensors, smart home etc.
• Edge computing as an extension to cloud
  – Cloud regions augmented by local regions
    • Local regions are edge clusters that offer normal cloud service (but at lower latency) E.g., AWS Boston region
  – Internet of Things (IoT) data processing services

Specialized Edge Computing

• Edge accelerators: special hardware to accelerate edge tasks on resource constrained edge servers
  – Nvidia Jetson GPU, Google edge Tensor processing Unit (TUP), Intel Vision Processing Unit (VPU)
• Example: IoT ML inference on edge accelerators
  – Efficient inference on resource-constrained edge servers

Google Edge TPU  
Nvidia Jetson Nano GPU  
Apple Neural Engine
Cloud and Edge Architectures

- Offload to cloud, edge, specialized edge,

![Diagram of cloud and edge architectures](image)

Traditional cloud (2-tier)

Traditional edge (3-tier)

Specialized (3-tier)