Raft Consensus Protocol

• Paxos is hard to understand (single vs multi-paxos)
• Raft - understandable consensus protocol
• State Machine Replication (SMR)
  – Implemented as a replicated log
  – Each server stores a log of commands, executes in order
  – Incoming requests —> replicate into logs of servers
  – Each server executed request log in order: stays consistent

• Raft: first elect a leader
• Leader sends requests (log entries) to followers
• If majority receive entry: safe to apply -> commit
  – If entry committed, all entries preceding it are committed

Today: World Wide Web

• WWW principles

• Case Study: web caching as an illustrative example
  – Invalidate versus updates
  – Push versus Pull
  – Cooperation between replicas
Traditional Web-Based Systems

1. Get document request (HTTP)

2. Server fetches document from local file

3. Response

- The overall organization of a traditional Web site.

Processes – Clients

- The logical components of a Web browser.
Processes – Clients

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

The Apache Web Server

- The general organization of the Apache Web server.
Multitiered Architectures

- The principle of using server-side CGI programs.

Web Server Clusters

- The principle of using a server cluster in combination with a front end to implement a Web service.
Web Server Clusters (2)

- A scalable content-aware cluster of Web servers.

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.

- (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.

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"?>
<Quote xmlns:s="http://example.org/stock-service">
  <TickerSymbol>IBM</TickerSymbol>
  <StockPrice>45.25</StockPrice>
</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"?>
<Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:s="http://www.example.org/stock-service">
  <Body>
    <GetStockQuote>
      <TickerSymbol>IBM</TickerSymbol>
    </GetStockQuote>
  </Body>
</Envelope>

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

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

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

```
<table>
<thead>
<tr>
<th>browser</th>
<th>Web server</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>request</td>
</tr>
<tr>
<td>response</td>
<td>response</td>
</tr>
</tbody>
</table>
```

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 => polling, Infinite leases => 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)
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

Adjustment Measures

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

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