Scaling Web Services

W3C Workshop on Web Services

Mark Nottingham
mnot@akamai.com
Web Services need:

- **Scalability**: handling increased load, while managing investment in providing service
- **Reliability**: high availability
- **Performance**: end-user perceived latency

• Problems (and solutions) similar to those in scaling the Web infrastructure in general.
Intermediary Solutions

• Intermediaries can address these needs by making a service available from a number of devices.
• They can be deployed on behalf of the client (proxy) or server (surrogate/CDN).
• We classify intermediaries into two broad categories: functional and optimizing.
Functional Intermediaries

• Make services available by taking responsibility for (parts of) them.
• Typically, functional solutions execute service-specific code (and need to distribute and manage it).
• If removed, service cannot be provided, because it performs meaningful processing on messages.
Optimizing Intermediaries

- Enhance service by exploiting certain aspects of services’ semantics, to:
  - Eliminate or delay connections to the origin server
  - Reduce the number of bytes sent to or from the origin server
- Removing from the message path has no effect, except to reduce scalability, performance and reliability.
Why Optimizing Intermediaries?

• Generally, it is easier to retrofit optimization onto existing services.
• Potential to support a wide range of applications, while avoiding code deployment and management issues.
• Optimizing isn’t for every service!
• Functional intermediaries are useful when optimization isn’t powerful or flexible enough.
Use Case: getStockQuote()

• Information services (stock quotes, RSS, etc.) can exploit request locality and reuse response components

• Different components have different triggers:
  – Delayed stock price – cache for 5 minute delta
  – Last close – invalidate at 5am daily
  – Company information – update upon notification
Use Case: Authentication Service

- Optimized by caching state in distributed intermediaries
- Request locality should keep cached state near users
- Updates can be triggered by other messages (new password), notifications, and/or incorrect passwords
- Accounting takes place through traditional logging, or aggregated store-and-forward.
Use Case: Distributed File Store

• Users interact with a local intermediary, writing to and reading from the cache. Events such as ‘logout’ synchronize the cache to a master server.
Proposed Optimization Model

1. Optimization hints about a service are made available to an intermediary.

2. Intermediaries process messages and apply optimization techniques based on the hints.

3. Techniques may be applied to XML elements at a fine granularity, and are triggered by a variety of events.
Applying Optimization Techniques

• XML offers much finer granularity of application (per-element) than traditional HTTP optimization (per-message).

• To associate a technique with an element, different mechanisms might be used:
  – External XML
    • Markup in XML Schema
    • Markup in WSDL
    • Message Header (using Xpath, etc.)
  – Inline XML
    • Attributes <foo m:cache="30”>
    • Namespaces <m:cache value="30”><foo/</m:cache>
Triggering Optimization Techniques

• Time (delta, absolute, schedule)
  – 30 seconds
  – Tuesday at 5pm
  – Jan 1 12:00 2002

• Message to the intermediary directly
  – XMLP service exposed on the intermediary

• Message passing through the intermediary
  – Presence or value of another message’s element (Query?)
Technique: Caching

- Exploits request/response message exchange patterns with locality in requests and similarity in responses.

- Cache can be indexed by some combination of:
  - Service (URL)
  - Element location (Xpath)
  - Element identity
  - Artificial index (URI)

- Cache is kept coherent through invalidation with various triggers.
Technique: Store-and-Forward

- Exploits large or frequent client to server transfers that do not require immediate processing (but may require acknowledgement)
- Forwarding is triggered.
- Acknowledgement message might be standardized, or constructed from cache.
Technique: Partial Messaging

- Specify that only part of a message should be sent
- Can be used to send only the changed parts of a message (e.g., to update a cache)
- Can be used to send parts of messages at different times (e.g., store-and-forward)
Technique: Aggregation

- Exploits locality in messages to combine multiple messages, where possible, into a single message.
- Messages should have at least one common endpoint.
Next Steps

• Validate techniques’ applicability to real-world Web Services
• Design a language for Web Service optimization
• Implement
• Interested? mailto:mnot@akamai.com