End to End Caching Protocol for Web Services

Amila Liyanaarachchchi
Shahani Weerawarana
Outline

• Background of the problem
• What needs to be improved related to WS performance
• Related research
• Proposed solution
• Architecture and Design
• Performance analysis
• Conclusion
• Future work
Background

- Web services are widely used
- In some areas web services can’t deliver the expected performance requirements
- What are the key factors controlling web services performance?
Literature: web services performance

• Network round-trip time [1]
• XML parsing overhead [2]
  • XML parsing adds an additional overhead compared to binary peers used in distributed computing (Java RMI, CORBA) [3]
• Actual service execution time
Factors affecting performance of WS

Client

Network

Network Delay

Server
Factors affecting performance of WS
Factors affecting performance of WS: XML Processing

```java
public double getPrice(String partNo) {
    ......
}
```
Factors affecting performance of WS: Service Execution

```java
public double getPrice(String partNo) {
    // Database access
    ...
    // Computations
    ...
    return price;
}
```
What needs to be improved

- Avoid/Minimize network round trip delay
- Minimize XML processing overhead
- Avoid/Minimize service execution time
Related research

• T. Takase et. al “A Web Services Cache Architecture Based on XML Canonicalization” [4], discusses the problem of representing semantically identical XML message in syntactically different forms.

• Uses XML canonicalization to address this problem.
Related research ...

• Maintaining a mapping between the cached responses and origin data is always a problem in web services which affects cache invalidation.

Related research ...

- Cache key and cache value representations are very important in any caching scheme.
- T. Takase, M. Tatsubori “Efficient web services response caching by selecting optimal data representation” [1] evaluates several cache key representations
- Cache value representations are also evaluated and XML response, SAX event sequence, application objects are being considered.
• Related research ...

• When the number of parameters of the web service increases, hit rate of reverse proxy cache decreases.

• Application object caching is a good alternative in such scenarios

• Based on this concept research, L. Li et al "An adaptive caching mechanism for Web Services" [2], proposes an adaptive cache scheme based on the hit rate.
• What can we learn from HTTP
  • Information exchange similar to WS
  • In both scenarios,
    • A request from a client
    • Server provides a response
  • Caching has been utilized for improving performance
• How does HTTP 1.1 caching work?
  • Cache-Control HTTP Header & directives
    • max-age
    • public/private
    • no-store
    • must-revalidate
  • HTTP Validation with E-tag
    • Send the request with if-non-match directive
    • Response should be HTTP 304 Not-modified/full resource
• Proposed Solution

- Introduce caches at server as well as at client
- New cache managers controlling the protocol
• Validation
  • A validation mechanism similar to HTTP validation using E-Tags
  • Response hash* will be used as the e-tag in the new protocol
  • Validation takes place against the cached copy at server cache

• * Axis2 implementation of DOMHash Algorithm will be used to generate the hash, which considers only the XML payload ignoring the namespace differences
• Cache hit/miss scenarios

1. Client cache hit, no validation
   - Web Service Request
   - StockQuoteService.getPrice
   - Return Cached Response from Client Cache
   - Axis2 Server Not Contacted

2. Server cache hit
   - Web Service Request
   - StockQuoteService.getPrice
   - Return Response
   - Response found in Server Cache, No Service Invoke
   - Return Cached Response

3. Client cache hit, validate
   - Web Service Request
   - StockQuoteService.getPrice
   - Return Cached Response from Client Cache
   - Compare Response Hashes (E-Tags)
   - Send SOAP 'Not Modified'
   - Validate

4. Cache miss
   - Web Service Request
   - StockQuoteService.getPrice
   - Return Response
   - Service Invoke
   - Service Invoke
   - Return Response
   - Return Cached Response
How each performance bottle-neck is addressed in the proposed scheme?

- **Client cache hit**: Eliminates the service execution, XML processing at server, Network round-trip
- **Server cache hit**: Eliminates XML processing, Service execution
- **Client cache hit with validation**: if the stored copies are identical, full response will not sent to the client saving network bandwidth
• Architecture and Design
  • Proposed caching protocol was developed on Apache Axis2
  • Axis2 is known to be a very good web services development platform
  • Axis2 is easily extensible
• Server Cache

• Extended WSO2 Carbon caching module

• Cache configuration information will be stored on deployment descriptors (module.xml, services.xml)
• **CachingInHandler**
  - Initialize cache manager (if not already initialized)
  - Check if the service is cacheable
  - Check for already cached responses for the same request
  - Attach the cached response object to operation context, if it is expired then attach an empty object and clear the expired response from cache
  - Service specific cache control parameters will be read from the services.xml and set them in the cached object
  - Validation is also takes place at this handler, and the state will be stored in cached object
• CachedMessageReceiver
  • Extract the cached object from the operation context, and check if it is empty
  • If the extracted cached object is non-empty then it will be returned in the out path
  • Otherwise the actual service invocation will be done by the message receiver and the response will be returned in the out path
CachingOutHandler

- Cached object will be extracted from the operation context
- If the cached object is empty then it will be updated with the response information
- Updated cached object will be linked with the cache manager
- Finally it will set the cache control SOAP headers in the response message
• Configuration: module.xml

```xml
<supported-policy-namespaces xmlns:supportedPolicyNamespaces="http://www.wso2.org/ns/commons/caching"/>
  xmlns:wsch="http://www.wso2.org/ns/2007/06/commons/caching"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <wsch:CachingAssertion>
    <wsp:Policy>
      <wsp:All>
        <wsch:XMLIdentifier>org.apache.axis2.caching.digest.DOMHashGenerator</wsch:XMLIdentifier>
        <wsch:ExpireTime>60000</wsch:ExpireTime>
        <wsch:MaxCacheSize>1000</wsch:MaxCacheSize>
        <wsch:MaxMessageSize>1000</wsch:MaxMessageSize>
      </wsp:All>
    </wsp:Policy>
  </wsch:CachingAssertion>
</wsp:Policy>
```

• Configuration: services.xml

```xml
<parameter name="ServiceClass">samples.ping.service.PingService</parameter>
<parameter name="revalidate">true</parameter>
<parameter name="maxage">6000</parameter>
<parameter name="no-store">false</parameter>
```
Axis2 extended Client API
(Class: CachedServiceClient)

- CachedServiceClient: extended from ServiceClient
- SendReceive:
  - This method has overwritten to support caching
  - Overloads to force service execution, get cached only responses
- Applications are supposed to use the CachedServiceClient instead of ServiceClient
• Apache Derby database is used in the embedded mode to store cached responses at each client
• Idea: Managing the caching protocol

HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
        <MAX-AGE>3600</MAX-AGE>
        <ETAG>3e86-410-3596fbbc</ETAG>
    </soap:Header>
    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:StockPrice>
            <m:StockName>IBM</m:StockName>
            <m:Price>200</m:Price>
        </m:StockPrice>
    </soap:Body>
</soap:Envelope>
New SOAP headers used in the protocol

- max-age
- no-store
- must-revalidate
- etag
- if-non-match
- not-modified
- only-if-cached
- force-service
• Performance Analysis
  • Following tests were performed
    • Performance with different cache hit/miss scenarios
    • Average response time vs. response message size
    • Average response time vs. service execution time
  • Average response time was measured by repeating the same test 100 times
A service with service execution time ~500ms was used in this test.
Results: Average response time vs. service execution time

A: Average response time with re-validation off
B: Average response time with re-validation on
E: Average response time with no caching
Results: Average response time vs. service execution time

A: Average response time with re-validation off
B: Average response time with re-validation on
E: Average response time with no caching
• Results: Average response time vs. service execution time

A: Average response time with re-validation off
B: Average response time with re-validation on
E: Average response time with no caching
• Conclusion

• This protocol addresses 2 out of 3 performance bottle necks identified in WS user scenarios

• According to the results this protocol delivers better performance when the service execution time is greater than 80ms
• Future Work

• Use of HTTP headers instead of new SOAP headers to exchange cache control information when the transport is HTTP
References


THANK YOU!
Appendix: Architecture - Axis Engine

Axis2 engine’s SOAP processing model consists of in and out pipes names flows.

There are four such flows,

- In-Flow
- Out-Flow
- In-Fault-Flow
- Out-Fault-Flow

All incoming messages will come through the in-flow and the out going messages will follow the out-flow.
• Appendix: Architecture - Axis Engine ...

• Each flow consists of a set of phases
• Phases are further divided into smaller components called handlers
• A group of logically related handlers are grouped into one module
Appendix: Axis2 Information model
Appendix: Axis2 Client API
(Class: ServiceClient)

- **SendRobust:**
  - Send request and ignore response, exception on invocation problems.

- **FireAndForget:**
  - Send request and ignore response (IN-ONLY MEP)

- **SendReceive:**
  - Send request, accept return value (IN-OUT MEP)

- **SendReceiveNonBlocking:**
  - Similar to SendReceive but non blocking invocation