REST architecture for perfSONAR

Nina Jeliazkova (BREN), Candido Montes (RedIris)

GN3 JRA2 Task 3

2010

An investigation of REST architecture for web services and its applicability to perfSONAR web services infrastructure. We present a short background on REST principles and propose a mapping of existing perfSONAR schemas and protocols as REST resources. Additionally, the results of a performance comparison of SOAP and REST demo implementation of perfSONAR services has been performed, with results favouring the REST architecture.

1 Representational State Transfer (REST)

REST is a software architecture style, defined by Roy Fielding in his PhD thesis (2000). There is a multitude of information on RESTful design principles, development frameworks and examples. For a start, the following references could be recommended:

- Chapter 5 of Fielding's dissertation: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- Leonard Richardson, Sam Ruby,RESTful Web Services,O'Reilly 2007: http://oreilly.com/catalog/9780596529260
- From SOA to REST: Designing and Implementing RESTful Services, Tutorial at WWW2009: http://dret.net/netdret/docs/soa-rest-www2009/
- How Do I Model State? Let Me Count the Ways, ACM Queue 2009: http://queue.acm.org/detail.cfm?id=1516638
- rest-discuss Yahoo group: http://groups.yahoo.com/group/rest-discuss/

Contrary to established WS-SOAP standards, there are no (currently) standards for RESTful applications, but merely design guides. Perhaps the first move towards standardization is the upcoming (First International Workshop on RESTful Design WS-REST 2010) to be held at WWW 2010 in Raleigh, North Carolina, USA next 26 April 2010.

Besides all discussions, setting REST against SOAP, this kind of comparison is not entirely correct, for SOAP is a protocol, and REST is an architectural style, not a protocol.

SOAP vs REST : http://www.iks.inf.ethz.ch/education/ss07/ws_soa/slides/SOAPvsREST_ETH.pdf

REST style can be briefly summarized as:

1.1 Design principles

1.1.1 Resource oriented

Every object (resource) is named and addressable (e.g. HTTP URL)

Example: http://perfsonarservice.net/MeasurementArchive/interface/interface_identifier

RESTfull API design starts by identifying most important objects and groups of objects, supported by the software system and proceeds by defining URL patterns. Common patterns are:

Returns list of objects in some format http://myservice.net/myobject

- Returns representation(s) of an object, identified by {myobjectid} http://myservice.net/myobject/{myobjectid}

Patterns may be nested, e.g. http://myservice.net/myobject/{myobjectid}/details/{detailsid}

1.1.2 Transport protocol

HTTP is the most popular choice of transport protocol, but there are examples of systems using other protocols as well.

1.1.3 Operations

Resources (nouns) support limited number of operations (verbs). HTTP operations are the common choice, when the transport protocol is HTTP.

1.1.3.1 RESTful operations

- GET (retrieve the object under specified URL)
- PUT (update the content of an object at the specified URL)
- POST (create a new object and return the URL of the newly created resource)
- DELETE (delete the object)

All operations, except POST should be safe (no side effects) and idempotent (same effect if executed multiple times).

1.1.3.2 Non-RESTful operations

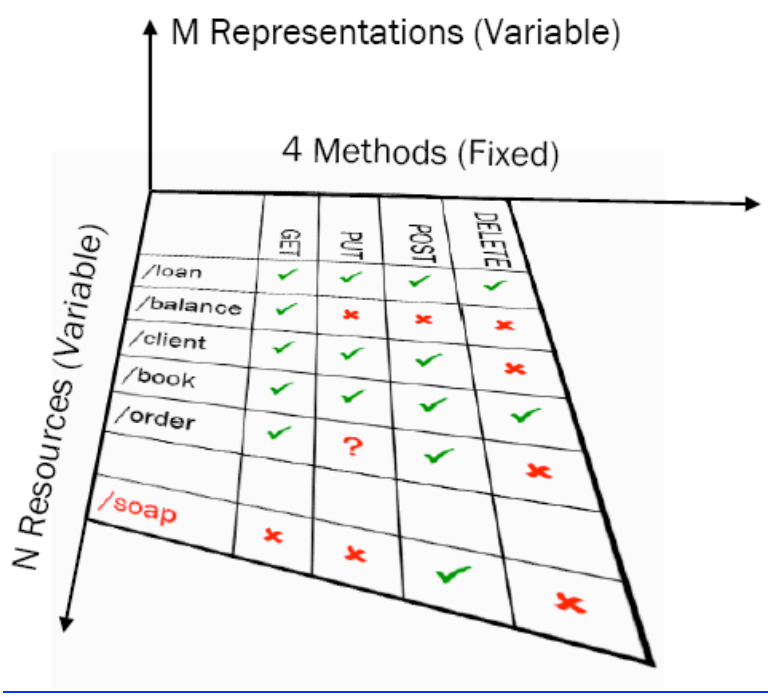
- Everything else , e.g. POST XML message to http://perfsonarservice.net/MAservice

Deciding on the operation to be done, on the basis of interpreting POSTed message content (the way a typical SOAP service works!) is NOT recommended. This is referred to as "overloaded POST" and considered violation of RESTful principles!

1.1.4 Resource representation

Every resource is defined by an URI. If GET operation on a resource URI returns some content, it is regarded as "dereferencable" (effectively it is an URL). A resource may return content in different formats (by specifying MediaType in the Accept: header of GET operation). The content is regarded as resource "representation". There could be multiple representations of the same resource (e.g. text/html or text/xml).

RESTfull API design includes specification of allowed media types for each resource/operation pair.



1.1.5 Hypermedia as the Engine of Application State (HATEOAS)

All resources should be reachable via a single (or minimum) number of entry points into a RESTful applications. Thus, a representation of a resource should return hypermedia links to related resources

- For example /perfsonar/interface/{id} resource should return links to /perfsonar/interface/{id}/metadata and /perfsonar/interface/{id}/metadata

- REST APIs must be hypertext driven (http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven)

1.1.6 Error codes

HTTP Status codes are used to indicate an operation success or failure.

RESTfull API design includes specification of status codes for each resource/operation pair.

1.2 Constraints

Client-server - Clients are separated from servers by a uniform interface.

Cacheable - Clients are able to cache responses.

Stateless design - No client context should be stored on the server between requests. Each request from any client contains all of the information necessary to service the request, and any state is held in the client.

Cookies are considered bad practice!

Layered - A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies.

Uniform interface - The uniform interface between clients and servers, (HTTP GET/PUT/POST/DELETE verbs only) simplifies and decouples the architecture, which enables each part to evolve independently.

1.3 Design goals

- Scalability of component interactions;
- Generality of interfaces;
- Independent deployment of components;
- Intermediary components to reduce latency, enforce security and encapsulate legacy systems

RESTful design principles are advocated as being successful, for underlying the existing WWW architecture. REST application are becoming increasingly popular, the trend with major service vendors

are to offer REST API along with an existing SOAP API. Some report REST API usage is increasing and SOAP API usage decreasing.

1.4 Drawbacks

- Lack of standards.
- Development frameworks are relatively young.
- The commonly recommended practice for security is to use HTTP Authentication mechanisms and SSL. Other solutions are emerging as well (discussed below).
- Examples of RESTful federated systems are rare.

1.5 Frameworks

Non exhaustive list of popular frameworks for developing RESTful applications:

- Java - Restlet, Jersey
- PHP - CakePHP
- Ruby - RubyOnRails, Sinatra
- Perl - Perl Application Framework and ERP Applications

1.6 Examples of REST APIs

- Amazon Web Services
    - Amazon S3 REST API
- Google Data API protocol
- The Sun Cloud API
- Yahoo web services
- Doodle
- Flickr

- NetFlix
- Twitter REST API
- Facebook REST API
- Blinksale

2 REST API for perfSONAR services

The proposal is based on the REST view that design of a REST system starts by identifying domain objects; following by mapping objects to URL pattenrs and defining operations on each object, by using HTTP verbs. In case of PerfSONAR, the objects of interest are monitored topology elements (e.g. interfaces, links, etc.). It seems suitable to represent topology elements as resources,having unique URL.

The proposal follows the recommended practice in REST is to have multiple URLs , corresponding to the domain objects, and only 4 type of actions (verbs), corrsesponding to HTTP GET/POST/PUT/DELETE operations.

There is not necessary a straightforward map between existing Perfsonar SOAP messages and the REST API, but NMWG schema is used as XML representation for all REST PerfSONAR resources. Requests are generally simplified to use URLs, instead of specifying parameters in XML message (there might be exceptions).

2.1 MIME types

A REST resource may have more than one representation. For example, a representation of a monitored interface may return its repreentation in XML format, following NMWG schema, but may also return alternative representation in different formats (even return a graph of the monitored data in image/jpeg format)

MediaType [1]:

- text/xml
- application/vnd.xml
- define PerfSONAR specific MIME type (if not existing already) application/vnd.perfsonar.xml
- text/uri-list (returns list of URLs to REST resources)
- alternative MIME types (e.g. image, or data representation in more efficient / compressed formats)

2.2 Resources and URL patterns

2.2.1 Service

URL: http://servicehostname:port/servicename/

Representation: text/xml

Operations:

- GET - return metadata of the service itself
- PUT - input: XML with service metadata ; output: adds service metadata
- POST - input: XML with service metadata ; output: replaces service metadata
- DELETE - remove service metadata (we may decide not to implement this operation)

2.2.2 Topology elements

URL:
http://servicehostname:port/servicename/topologyelements?paramName=value&paramName=value1&paramName1=value2

Representation: text/xml

Operations:

- GET Returns NMWG representation of topology elements (response of the NMWG MetadataKeyRequest)
- PUT
- POST Create new topology element(s) by sending representation in NMWGT XML
- DELETE Delete all topology elements

Parameters : parameters and values as in NMWG/NMWGT

Examples:

- Get all metadata elements http://servicehostname:port/servicename/topologyelements
- Get all topology elements, where there exist data for utilization and errors metrics http://servicehostname:port/servicename/topologyelements?eventType=http://ggf.org/ns/nmwg/characteristic/utilization/2.0&eventType=http://ggf.org/ns/nmwg/characteristic/errors/2.0&startTime=1262304000&endTime=1262390400&timeType=unix

- Find an interface with IP 10.0.0.1
  http://servicehostname:port/servicename/topologyelements?ipaddress=10.0.0.1 -> returns URL of the interface in the form of
  http://servicehostname:port/servicename/topologyelements/interface/interface-id (see next section)

2.2.3 Topology elements of specific type (Metadata)

SOAP request: MetadataKeyRequest

```
<nmwg:message id="1264011628" type="MetadataKeyRequest"
xmlns:errors="http://ggf.org/ns/nmwg/characteristic/errors/2.0/"
xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/"
xmlns:select="http://ggf.org/ns/nmwg/ops/select/2.0/">
 <nmwg:metadata id="meta1">
   <errors:subject id="iusub1">
     <nmwgt:interface/>
   </errors:subject>
   <nmwg:eventType>
     http://ggf.org/ns/nmwg/characteristic/errors/2.0
   </nmwg:eventType>
 </nmwg:metadata>
 <nmwg:data id="data_1" metadataIdRef="meta1"/>
</nmwg:message>
```

URL (REST request):

http://servicehostname:port/servicename/topologyelements/{type-of-topology-element}/metadata?parameterName=value

SOAP response: MetadataKeyResponse

Representation:

- (REST GET response)
- NMWG XML , content type text/xml or to be defined

```
<nmwgt:interface>
  <nmwgt:hostname>hostname</nmwgt:hostname>
  <nmwgt:ifName>ifName</nmwgt:ifName>
  <nmwgt:ifDescription>ifDescription</nmwgt:ifDescription>
  <nmwgt:ifAddress>ifAddress</nmwgt:ifAddress>
```

```
  <nmwgt:ifIndex>ifIndex</nmwgt:ifIndex>
  <nmwgt:direction>direction</nmwgt:direction>
  <nmwgt:capacity>capacity</nmwgt:capacity>
</nmwgt:interface>
```

- text/uri-list , returninglist of URLs of the specific topology elements, matching the query, as below

```
http://servicehostname:port/servicename/topologyelements/interface/id1
http://servicehostname:port/servicename/topologyelements/interface/id2
```

Operations:

- GET Return query results in NMWG XML
- PUT Update information of topology elements input:Content-type:text/xml; output: Content:NMWG representation of an interface metadata
- POST Add information of topology elements input:Content-type:text/xml; output: Content:NMWG representation of an interface
- DELETE Delete all topology elements of this type

Examples:

- Content Type: text/xml
  http://servicehostname:port/servicename/topologyelements/interface/metadata?eventType=http://ggf.org/ns/nmwg/characteristic/utilization/2.0&direction=in

```
<nmwgt:interface>
  <nmwgt:hostname>hostname1</nmwgt:hostname>
  <nmwgt:ifName>ifName1</nmwgt:ifName>
  <nmwgt:ifDescription>ifDescription1</nmwgt:ifDescription>
  <nmwgt:ifAddress>ifAddress<1/nmwgt:ifAddress>
  <nmwgt:ifIndex>ifIndex1</nmwgt:ifIndex>
  <nmwgt:direction>in</nmwgt:direction>
  <nmwgt:capacity>capacity</nmwgt:capacity>
</nmwgt:interface>
<nmwgt:interface>
  <nmwgt:hostname>hostname2</nmwgt:hostname>
  <nmwgt:ifName>ifName2</nmwgt:ifName>
  <nmwgt:ifDescription>ifDescription2</nmwgt:ifDescription>
  <nmwgt:ifAddress>ifAddress2</nmwgt:ifAddress>
  <nmwgt:ifIndex>ifIndex</nmwgt:ifIndex>
  <nmwgt:direction>in</nmwgt:direction>
  <nmwgt:capacity>capacity</nmwgt:capacity>
</nmwgt:interface>
```

- Content Type: text/uri-list
  http://servicehostname:port/servicename/topologyelements/interface/metadata?eventType=http://ggf.org/ns/nmwg/characteristic/utilization/2.0&direction=in

```
http://servicehostname:port/servicename/topologyelements/interface/1
http://servicehostname:port/servicename/topologyelements/interface/2
```

2.2.4 Topology elements of specific type (Data)

SOAP request: SetupDataRequest

```
<nmwg:message id="message1264011954" type="SetupDataRequest"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/"
xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/"
xmlns:select="http://ggf.org/ns/nmwg/ops/select/2.0/">
 <nmwg:metadata id="meta_in">
   <netutil:subject id="subject_1">
     <nmwgt:interface>
       <nmwgt:hostName>rtr.chic.net.internet2.edu</nmwgt:hostName>
       <nmwgt:ifName>ge-2/1/0.178</nmwgt:ifName>
       <nmwgt:ifAddress>192.245.196.110</nmwgt:ifAddress>
       <nmwgt:direction>in</nmwgt:direction>
     </nmwgt:interface>
   </netutil:subject>
   <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
 </nmwg:metadata>
 <nmwg:metadata id="m_param_in">
   <select:subject id="subject1_2" metadataIdRef="meta_in"/>
   <nmwg:eventType>http://ggf.org/ns/nmwg/ops/select/2.0</nmwg:eventType>
   <select:parameters id="param_3">
     <nmwg:parameter name="startTime">1264011835</nmwg:parameter>
     <nmwg:parameter name="endTime">1264271035</nmwg:parameter>
     <nmwg:parameter name="consolidationFunction">AVERAGE</nmwg:parameter>
     <nmwg:parameter name="resolution">300</nmwg:parameter>
   </select:parameters>
 </nmwg:metadata>
 <nmwg:data id="data_in" metadataIdRef="m_param_in"/>
 <nmwg:metadata id="meta_out">
   <netutil:subject id="subject_3">
     <nmwgt:interface>
       <nmwgt:hostName>rtr.chic.net.internet2.edu</nmwgt:hostName>
       <nmwgt:ifName>ge-2/1/0.178</nmwgt:ifName>
       <nmwgt:ifAddress>192.245.196.110</nmwgt:ifAddress>
```

```
      <nmwgt:direction>out</nmwgt:direction>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
</nmwg:metadata>
<nmwg:metadata id="m_param_out">
  <select:subject id="subject1_4" metadataIdRef="meta_out"/>
  <nmwg:eventType>http://ggf.org/ns/nmwg/ops/select/2.0</nmwg:eventType>
  <select:parameters id="param_5">
    <nmwg:parameter name="startTime">1264011835</nmwg:parameter>
    <nmwg:parameter name="endTime">1264271035</nmwg:parameter>
    <nmwg:parameter name="consolidationFunction">AVERAGE</nmwg:parameter>
    <nmwg:parameter name="resolution">300</nmwg:parameter>
  </select:parameters>
</nmwg:metadata>
<nmwg:data id="data_out" metadataIdRef="m_param_out"/>
</nmwg:message>
```

URL (REST request):

http://servicehostname:port/servicename/topologyelements/{type-of-topology-element}/data?parameterName=value&paramName=value1

Representation:

- NMWG XML , content type text/xml or to be defined
- text/uri-list , returning URLs of the specific topology elements, matching the query, as below

```
http://servicehostname:port/servicename/topologyelements/interface/id1/data?parameterName=value
http://servicehostname:port/servicename/topologyelements/interface/id2/data?parameterName=value
```

Operations:

- GET Return query results in NMWG XML
- PUT - none
- POST - none
- DELETE Delete all data for the topology elements of this type

Examples:

- Retrieve utilisation data for all interfaces with specified time interval and resolution

http://servicehostname:port/servicename/topologyelements/interface/data?eventType=http://ggf.org/ns/nmwg/characteristic/utilization/2.0&startTime={unixtime}&endTime={unixTime}&resolution={resolution}

SOAP and REST (text/xml) Response (SetupDataResponse)

```
<?xml version="1.0" encoding="UTF-8" ?>
<nmwg:message id="message1264013036_resp" messageIdRef="message1264013036"
type="SetupDataResponse" xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
<nmwg:metadata id="meta542_0">
<netutil:subject id="subj542"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/">
<nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
<nmwgt:hostName>swiNE1.switch.ch</nmwgt:hostName>
<nmwgt:ifName>GigabitEthernet1/4</nmwgt:ifName>
<nmwgt:ifDescription>bidir ZX-SFP to NE2</nmwgt:ifDescription>
<nmwgt:ifAddress type="ipv4">130.59.36.117</nmwgt:ifAddress>
<nmwgt:direction>out</nmwgt:direction>
<nmwgt:authRealm>TestRealm</nmwgt:authRealm>
<nmwgt:capacity>1000000000</nmwgt:capacity>
</nmwgt:interface>
</netutil:subject>
<nmwg:eventType>
   http://ggf.org/ns/nmwg/characteristic/utilization/2.0
</nmwg:eventType>
</nmwg:metadata>
<nmwg:metadata id="meta541_1">
<netutil:subject id="subj541"
  xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/">
<nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
<nmwgt:hostName>swiNE1.switch.ch</nmwgt:hostName>
<nmwgt:ifName>GigabitEthernet1/4</nmwgt:ifName>
<nmwgt:ifDescription>bidir ZX-SFP to NE2</nmwgt:ifDescription>
<nmwgt:ifAddress type="ipv4">130.59.36.117</nmwgt:ifAddress>
<nmwgt:direction>in</nmwgt:direction>
<nmwgt:authRealm>TestRealm</nmwgt:authRealm>
<nmwgt:capacity>1000000000</nmwgt:capacity>
</nmwgt:interface>
</netutil:subject>
<nmwg:eventType>
   http://ggf.org/ns/nmwg/characteristic/utilization/2.0
 </nmwg:eventType>
</nmwg:metadata>
<nmwg:data id="localhost.616fe810:1264cf81d1b:-38e6_0" metadataIdRef="meta542_0">
```

```
 <netutil:datum timeType="unix" timeValue="1263852300" value="31074.162409547513"
valueUnits="Bps"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" />

 <netutil:datum timeType="unix" timeValue="1263852600" value="28453.496940521556"
valueUnits="Bps"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" />

 <netutil:datum timeType="unix" timeValue="1263852900" value="40126.91894595665"
valueUnits="Bps"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" />

....

 <netutil:datum timeType="unix" timeValue="1263938400" value="14961.938034206443"
valueUnits="Bps"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" />

 <netutil:datum timeType="unix" timeValue="1263938700" value="4117.60638664301"
valueUnits="Bps"
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" />

 <nmwg:parameters id="localhost.616fe810:1264cf81d1b:-38d3" />

 </nmwg:data>

 </nmwg:message>
```

2.2.5 A single topology element

URL:

http://servicehostname:port/servicename/topologyelements/{type-of-topology-element}/{id-of-the-topology-element}

Representation: text/xml

Operations:

- GET Retrieve metadata of the topology element
- PUT Add metadata of the topology element input: NMWGT XML
- POST Replace metadata of the topology element input: NMWGT XML
- DELETE Delete topology element

Examples:

- GET http://servicehostname:port/servicename/topologyelements/interface/1

```
<nmwgt:interface>
  <nmwgt:hostname>myhostname</nmwgt:hostname>
  <nmwgt:ifName>myifname</nmwgt:ifName>
  <nmwgt:ifDescription>My Interface</nmwgt:ifDescription>
  <nmwgt:ifAddress>10.0.0.1</nmwgt:ifAddress>
  <nmwgt:ifIndex>eth0</nmwgt:ifIndex>
```

```
  <nmwgt:direction>in</nmwgt:direction>
  <nmwgt:capacity>100000000</nmwgt:capacity>
</nmwgt:interface>
```

GET http://servicehostname:port/servicename/topologyelements/link/XYZ

TODO

2.2.6 Monitoring data of a topology element

URL:

http://servicehostname:port/servicename/topologyelements/{type-of-topology-element}/{id-of-the-topology-element}/data

http://servicehostname:port/servicename/topologyelements/{type-of-topology-element}/{id-of-the-topology-element}/data?parameterName=value

Representation:

- text/xml (e.g. nmwg:datum elements)

Operations:

- All operations are on the topology element (e.g. interface), determined by the URL
- GET Retrieves monitoring data of the topology element (e.g. nmwg:datum elements)
- PUT Adds monitoring data of the topology element
- POST Replace monitoring data of the topology element
- DELETE Delete monitoring data of the topology element

Examples:

- GET http://servicehostname:port/servicename/topologyelements/interface/1/data?startTime1263945600&endTime=1263946200&resolution=300

```
<nmwg:data>
  <netutil:datum value="31074.162409547513" timeValue="1263945600"
valueUnits="Bps" timeType="unix"/>
  <netutil:datum value="31074.162409547513" timeValue="1263945900"
valueUnits="Bps" timeType="unix"/>
  <netutil:datum value="31074.162409547513" timeValue="1263946200"
valueUnits="Bps" timeType="unix"/>
</nmwg:data>
```

- POST - similar to StoreDataRequest

2.2.7 Metadata chaining

The concept of metadata chaining doesn't seem to fit with the REST view of representing each topology element as a resource with unique URL. There are two type of metadata chaining - merging and filtering.

- Example 1
    - SOAP example

      http://anonsvn.internet2.edu/svn/perfsonar/branches/new-structure-with-base2/ps-mdm-rrd-ma/samples/requests/SetupDataRequest-Utilization-2.xml

      This example specifies the interface searching criteria in first metadata and then defines time range, resolution and consolidation function in the second metadata. The response will return monitoring data of the specified interface and time range, if such exist.

    - REST

      REST implementation could be obtained by simply sending subsequent HTTP GET requests for the required data.

      - Find the interface

        ```
        curl -X GET "Content-Type:text/uri-list"
        http://servicehostname:port/servicename/topologyelements/interface/metadat
        a?ifAddress=10.1.2.3&hostName=test-hostName&direction=in
        ```

        The result is URL of the interface, if exist, with status code 200 and status code 400 "Not Found", if no such interface exist.

        ```
        http://servicehostname:port/servicename/topologyelements/interface/1
        ```

      - Retrieve monitoring data

        ```
        curl -X GET "Content-Type:text/xml;charset=UTF-8" GET
        http://servicehostname:port/servicename/topologyelements/interface/1/data?
        startTime=2007-08-10T10:40:00Z&endTime=2007-08-
        10T11:45:00Z&timeType=ISO&consolidationFunction=AVERAGE&resolution=60
        ```

        This will return nmwg:datum elements, if such exist, otherwise status code 400 Not found should be returned

- Example 2
    - SOAP

      http://anonsvn.internet2.edu/svn/perfsonar/branches/GEANT2-JAVA-RRD-MA-STABLE/samples/requests/SetupDataRequest-Utilization-5.xml

- REST

  Similar to Example 1, achieved by sequence of HTTP GET requests

2.3 Error codes

HTTP status codes should be defined for each REST operation

2.4 Echo message

There are two kind of requests:

- **Echo request**: checks if the service is alive.
- **Self tests request**: checks if the service has been configured properly.

Echo request using NMWG

The request is:

```
<nmwg:message type="EchoRequest"
            id="id1"
            xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
  <nmwg:metadata id="meta">
      <nmwg:eventType>http://schemas.perfsonar.net/tools/admin/echo/2.0</nmwg:eventType>
  </nmwg:metadata>
  <nmwg:data id="data" metadataIdRef="meta"/>
</nmwg:message>
```

And the response saying that the service is ready is:

```
<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
         id="id1_resp"
         messageIdRef="id1"
         type="EchoResponse">
    <nmwg:metadata id="resultCodeMetadata">
       <nmwg:eventType>success.echo</nmwg:eventType>
    </nmwg:metadata>
    <nmwg:data id="resultDescriptionData_for_resultCodeMetadata"
               metadataIdRef="resultCodeMetadata">
       <nmwgr:datum xmlns:nmwgr="http://ggf.org/ns/nmwg/result/2.0/">
```

```
          This is the echo response from the service.
       </nmwgr:datum>
    </nmwg:data>
</nmwg:message>
```

Echo request in REST

The request is:

http://ps_rest_service/admin/echo/2.0

or

http://servicehostname:port/servicename/echo/2.0

And the response saying that the service is ready is:

```
<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
         id="id1_resp"
         messageIdRef="id1"
         type="EchoResponse">
    <nmwg:metadata id="resultCodeMetadata">
       <nmwg:eventType>success.echo</nmwg:eventType>
    </nmwg:metadata>
    <nmwg:data id="resultDescriptionData_for_resultCodeMetadata"
               metadataIdRef="resultCodeMetadata">
       <nmwgr:datum xmlns:nmwgr="http://ggf.org/ns/nmwg/result/2.0/">
          This is the echo response from the service.
       </nmwgr:datum>
    </nmwg:data>
</nmwg:message>
```

or could be just HTTP 200 Success.

2.5 Authentication model

The authentication of the official perfSONAR is based on Web Services Security, as they are services using SOAP 1.1. Furthermore, it uses the X.509 digital certificate profile and the SAML profile in order in to include the security tokens defined in its architecture.

The main problem with RESTful web services is that there is no defined standard for protecting them. However, there is one draft, HTTP Authentication: Token Access Authentication which can be used in order to include the security tokens in every request.

Although that draft was defined by its author for finding out a better solution for OAuth, it's quite generic and it can be used for other kind of tokens.

2.5.1 Profile based on X.509 digital certificates

When the security token is based on digital certificates, the http request will include the next attributes in the http authentication header:

- class: contains the value x509, specifying that the token is a X.509 digital certificate.
- method: contains the value rsassa-pkcs1-v1.5-sha-256.
- nonce: contains a random string as the draft specifies.
- token: contains the digital certificate in base 64 codification.
- timestamp: contains the timestamp of the user's client.
- auth: contains a hash using the private key of the client/user.

The next request is an example:

```
GET aaaa HTTP / 1.1
Host: example.com
Authorization: Token token="h480djs93hd8...yZT4=",
      class="x509",
      method="rsassa-pkcs1-v1.5-sha-256",
      nonce="dj83hs9s",
      timestamp="137134190",
      auth="djosJKDKJSD8743243/jdk33klY="
```

2.5.2 Profile based on SAML assertions

When the security token is based on SAML assertions, the http request will include the next attributes in the http authentication header:

- **class**: contains the value **samlv2.0**, specifying that the token is an assertion of the SAML v2 specification.
- **method**: contains the value *none*.

- **token**: contains the SAML assertion in base 64 codification.

The next request is an example:

```
GET aaaa HTTP / 1.1
Host: example.com
Authorization: Token token="h480djs93hd8...yZT4=",
      class="samlv2.0",
      method="none"
```

2.5.3 Request not valid

In case the request doesn't apply any of both previous profiles, the service must reply saying which that the client is not authorized and which profiles must apply.

An example of a request:

```
GET aaaa HTTP / 1.1
Host: example.com
```

An example of a response:

```
HTTP / 1.1 401 Unauthorized
WWW-Authenticate: Token class="samlv2.0",
                methods="none",
                timestamp="137131190"
WWW-Authenticate: Token class="x509",
                methods="rsassa-pkcs1-v1.5-sha-256",
                timestamp="137131190"
```

2.5.4 New model (REST) vs Classic model (SOAP)

There aren't main differences between both models as they're using the same security tokens but described in different ways. The 90% of the source code related to Authentication can be re-used so it won't required a lot of effort to implement these profiles.

2.6 Performance comparison

Simple SOAP and REST services have been developed specifically for the purpose of performance comparison between SOAP and REST implementations of Perfsonar. The SOAP service is a stripped-down implementation, using same code base, as production Perfsonar services and Apache Axis2 SOAP implementation. The REST services are based on Restlet framework.

2.6.1 SOAP vs REST performance comparison

Response time of test implementations of SOAP and REST services were compared with the use of SmokePing, curl probes, running on two different servers at BREN - one on the same server, as the test services itself, (Central BREN node, Sofia, BG), and one on a server in a different city (BREN PoP at Plovdiv University, Plovdiv, BG), connected via 1Gb link. SmokePing curl probes are configured to send 10 requests for every target every 5 min.

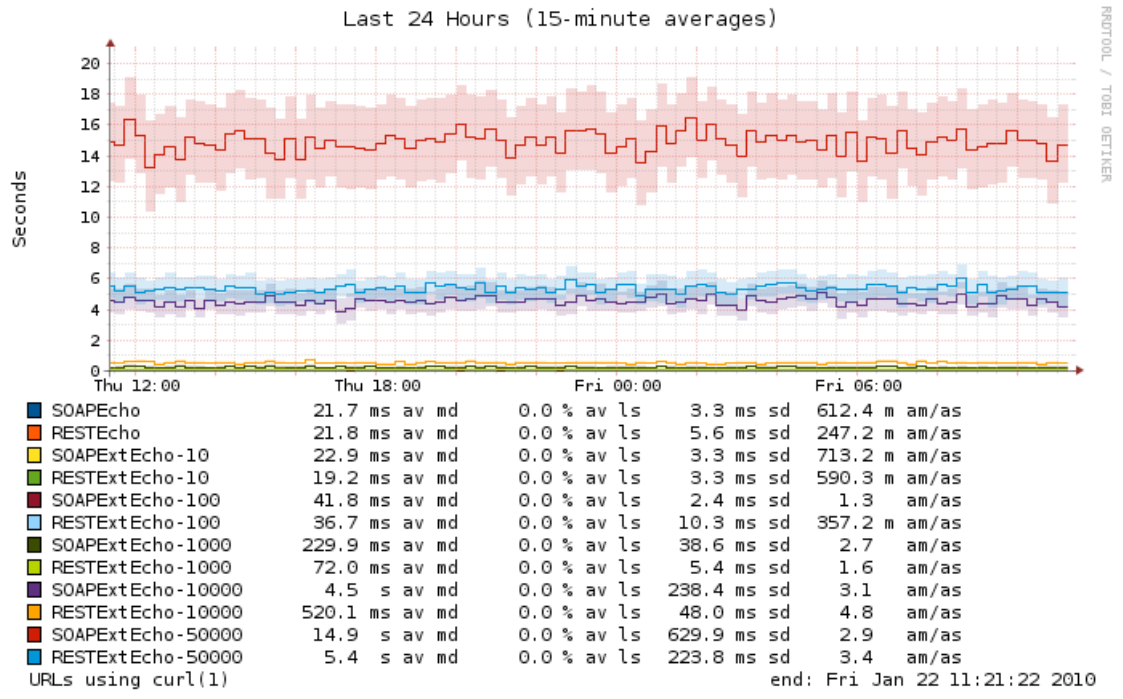
Server:

- Intel(R) Xeon(TM) CPU 2.00GHz
- Linux 2.6.18-6-686
- Location Sofia

Client (smokeping curl probe):

- AMD Athlon(tm) Processor LE-1640 2.6GHz
- Linux version 2.6.26-2-amd64 (Debian 2.6.26-19lenny2)
- Location Plovdiv University

All charts in this reports are from the SmokePing installation in Plovidiv University. This measurement setup mimics the real use case of consuming web services remotely.



2.6.1.1 Perfsonar ECHO message

- ECHO request: as above
- ECHO response: as above
- Response message size: ~400b

SOAPEcho:

- SOAP implementation: http://uran.acad.bg:8180/rest-test/services/SimpleService
- Response message generation: NMWG classes, perfsonar-base2
- TEST query:

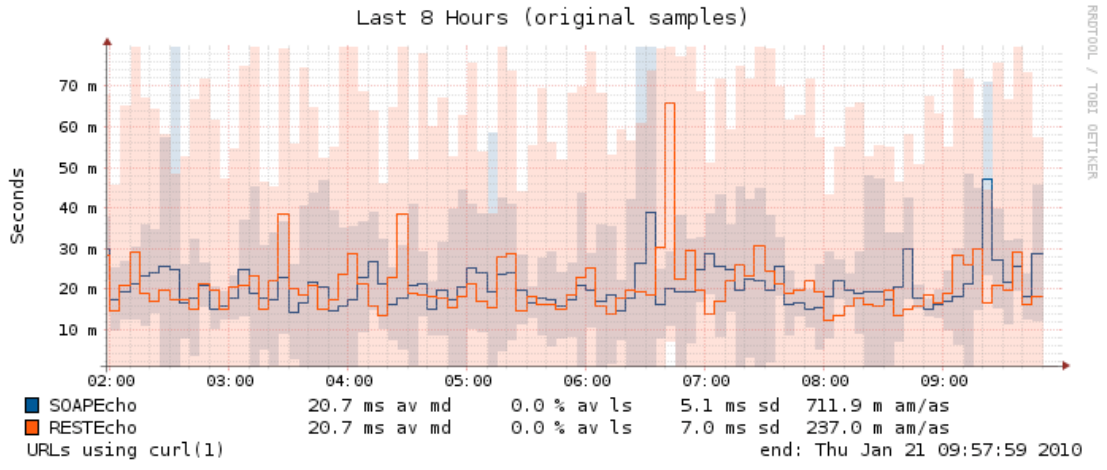
```
curl -X POST -H Content-Type:text/xml;charset=UTF-8 -d @FileWithEchoRequest
http://uran.acad.bg:8180/rest-test/services/SimpleService
```

RESTEcho:

- REST implementation: http://uran.acad.bg:8180/rest-test/rest/echo
- Response message generation: NMWG classes, perfsonar-base2
- TEST query:

```
curl -X GET http://uran.acad.bg:8180/rest-test/rest/echo
```

Response time:



2.6.1.2 Extended ECHO message

In order to test performance of services when handling large messages, and extended ECHO request was introduced (for this test only) which adds nmwg:metadata/nmwg:data pairs to the ECHO response. Tests are performed with messages, containing 10,100,1000,10000 and 50000 metadata/data pairs.

ECHO request

REST

http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/NUMBER_OF_METADATA_DATA_PAIRS

SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
     <nmwg:message type="ExtendedEchoRequest" id="id1"
xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
        <nmwg:metadata id="meta">
<nmwg:eventType>http://schemas.perfsonar.net/tools/admin/echo/2.0/NUMBER_OF_METADA
TA_DATA_PAIRS</nmwg:eventType>
        </nmwg:metadata>
        <nmwg:data id="data" metadataIdRef="meta"/>
     </nmwg:message>
  </soapenv:Body>
</soapenv:Envelope>
```

ECHO Response (SOAP and REST)

```
<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" type="EchoResponse" id="uri:ps-
element-id:1263973869874:1280641648">
  <nmwg:data metadataIdRef="meta0" id="data0"></nmwg:data>
  <nmwg:metadata id="meta0">
   <nmwg:eventType>http://schemas.perfsonar.net/tools/admin/echo/2.0</nmwg:eventType>
  </nmwg:metadata>
....
  <nmwg:data metadataIdRef="meta9999" id="data9999"></nmwg:data>
  <nmwg:metadata id="meta9999">
    <nmwg:eventType>http://schemas.perfsonar.net/tools/admin/echo/2.0</nmwg:eventType>
  </nmwg:metadata>
</nmwg:message>
```

Performance

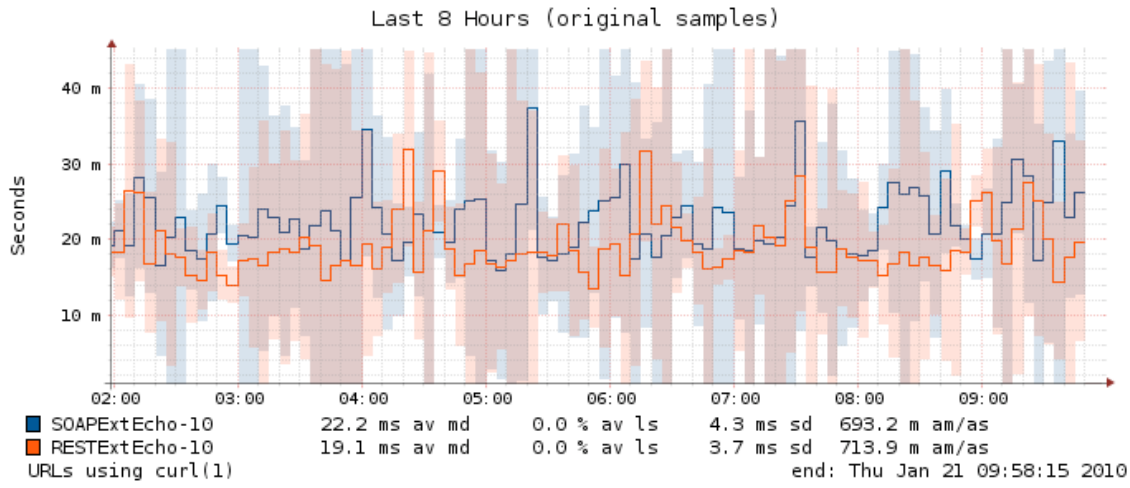
SOAPExtEcho-10 vs RESTExtEcho-10

- Message size (metadata/data pairs): 10
- Message size (b):
- REST service:

```
curl -X http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/10
```

- SOAP service:

```
curl -X POST -H Content-Type:text/xml;charset=UTF-8 -d
@FileWithExtendedEchoRequest http://uran.acad.bg:8180/rest-
test/services/SimpleService
```



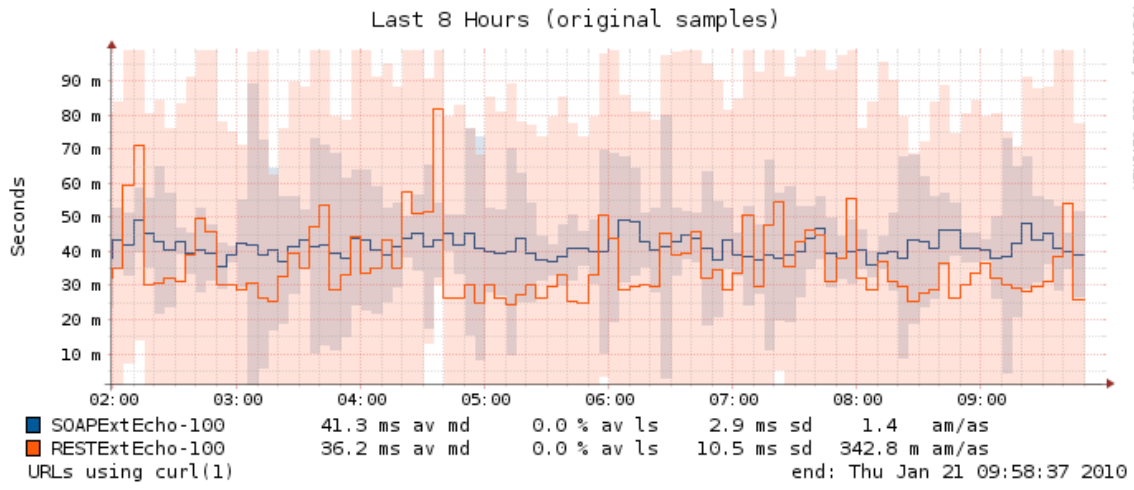
SOAPExtEcho-100 vs RESTExtEcho-100

- Message size (metadata/data pairs): 100
- Message size (b):
- REST service:

```
curl -X http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/100
```

- SOAP service:

```
curl -X POST -H Content-Type:text/xml;charset=UTF-8 -d
@FileWithExtendedEchoRequest http://uran.acad.bg:8180/rest-
test/services/SimpleService
```



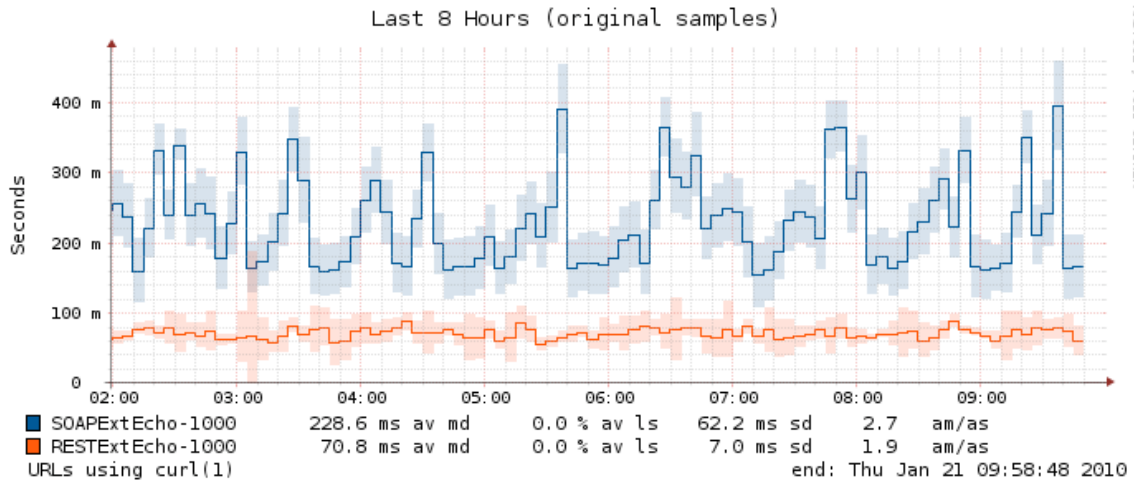
SOAPExtEcho-1000 vs RESTExtEcho-1000

- Message size (metadata/data pairs): 1000
- Message size (b):
- REST service:

```
curl -X http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/1000
```

- SOAP service:

```
curl -X POST -H Content-Type:text/xml;charset=UTF-8 -d
@FileWithExtendedEchoRequest http://uran.acad.bg:8180/rest-
test/services/SimpleService
```



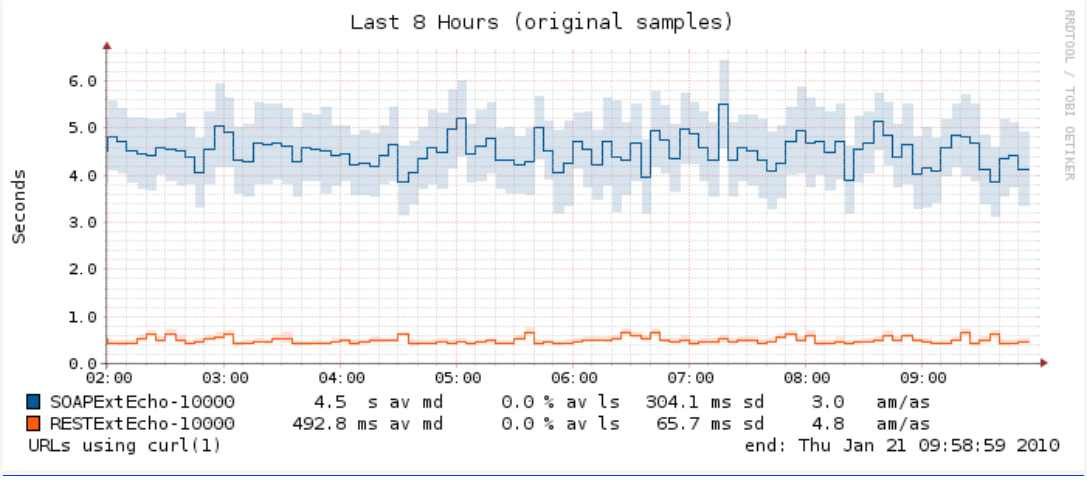
SOAPExtEcho-10000 vs RESTExtEcho-10000

- Message size (metadata/data pairs): 10000
- Message size (b):
- REST service:

```
curl -X http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/10000
```

- SOAP service:

```
curl -X POST -H Content-Type:text/xml;charset=UTF-8 -d
@FileWithExtendedEchoRequest http://uran.acad.bg:8180/rest-
test/services/SimpleService
```



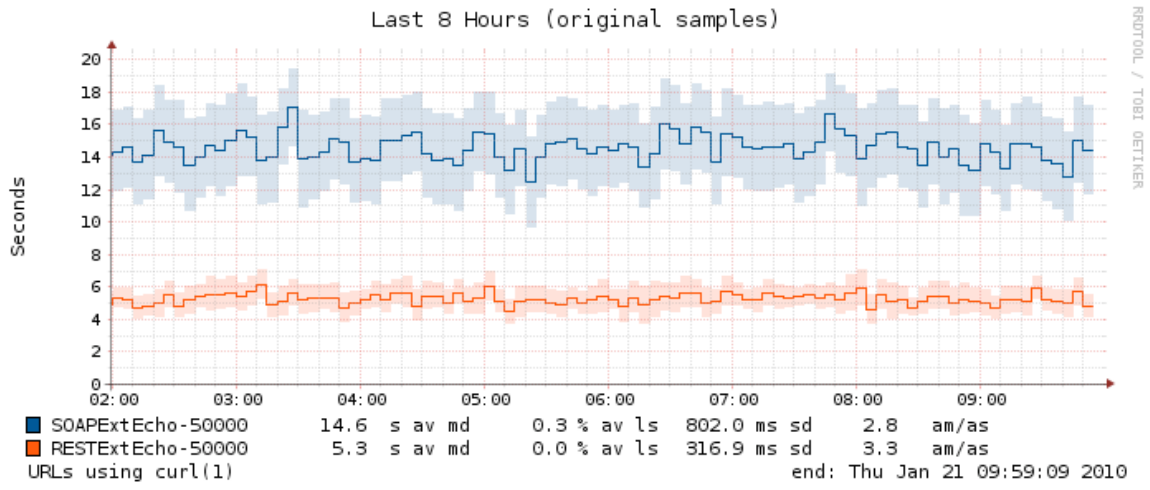
SOAPExtEcho-50000 vs RESTExtEcho-50000

- Message size (metadata/data pairs): 50000
- Message size (b):
- REST service:

```
curl -X http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/50000
```

- SOAP service:

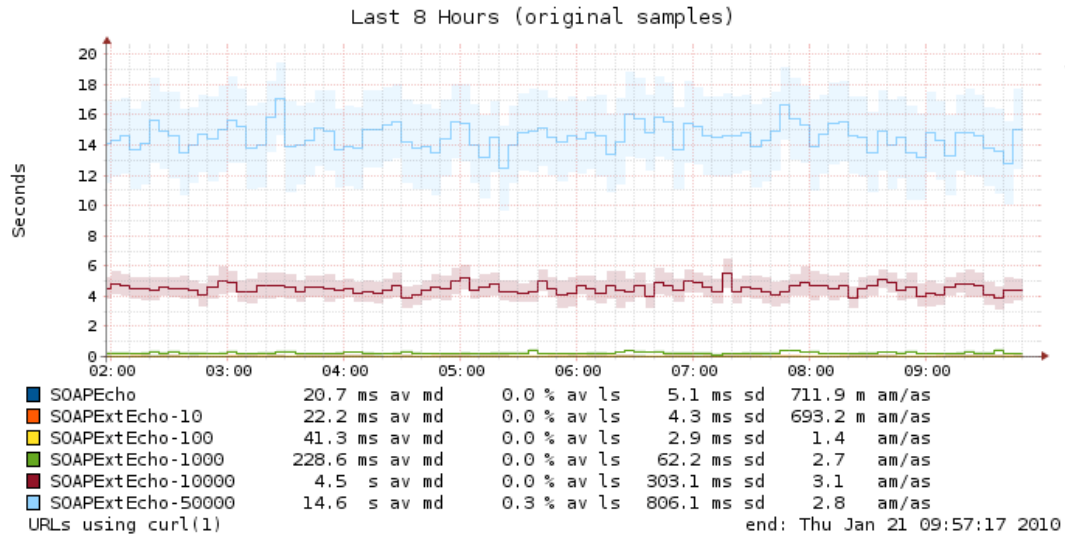
```
curl -X POST -H Content-Type:text/xml;charset=UTF-8 -d
@FileWithExtendedEchoRequest http://uran.acad.bg:8180/rest-
test/services/SimpleService
```



2.6.1.3 Summary

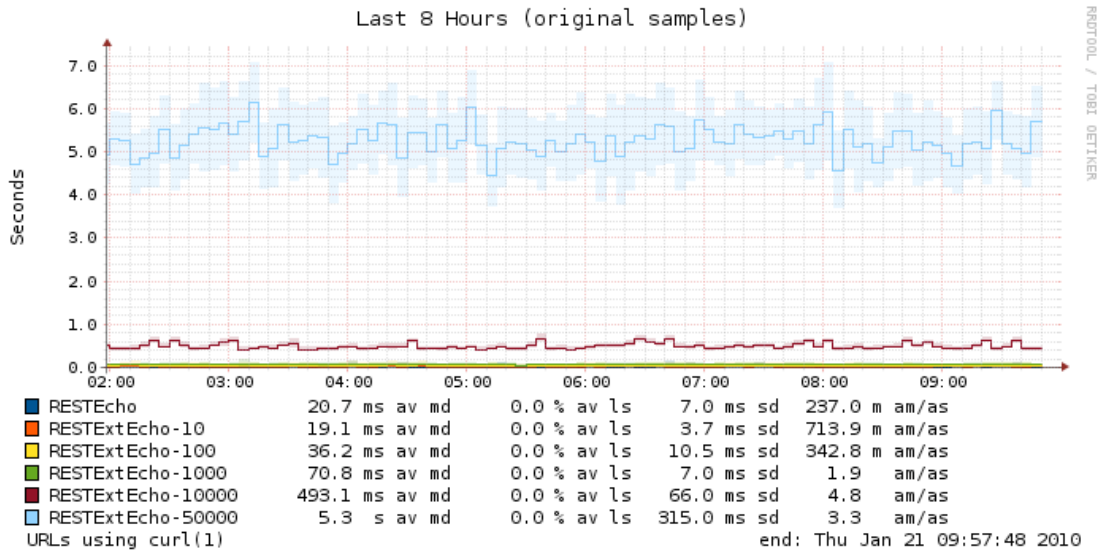
Performance of SOAP services, related to the size of response message

Extended echo response messages generated by NMWG Java classes



Performance of REST services, related to the size of response message

Extended echo response messages generated by NMWG Java classes



2.6.1.4 Conclusions

- Small response messages

  With the size of response message about 400b, SOAP and REST implementations' response time is practically the same.

- Large response messages

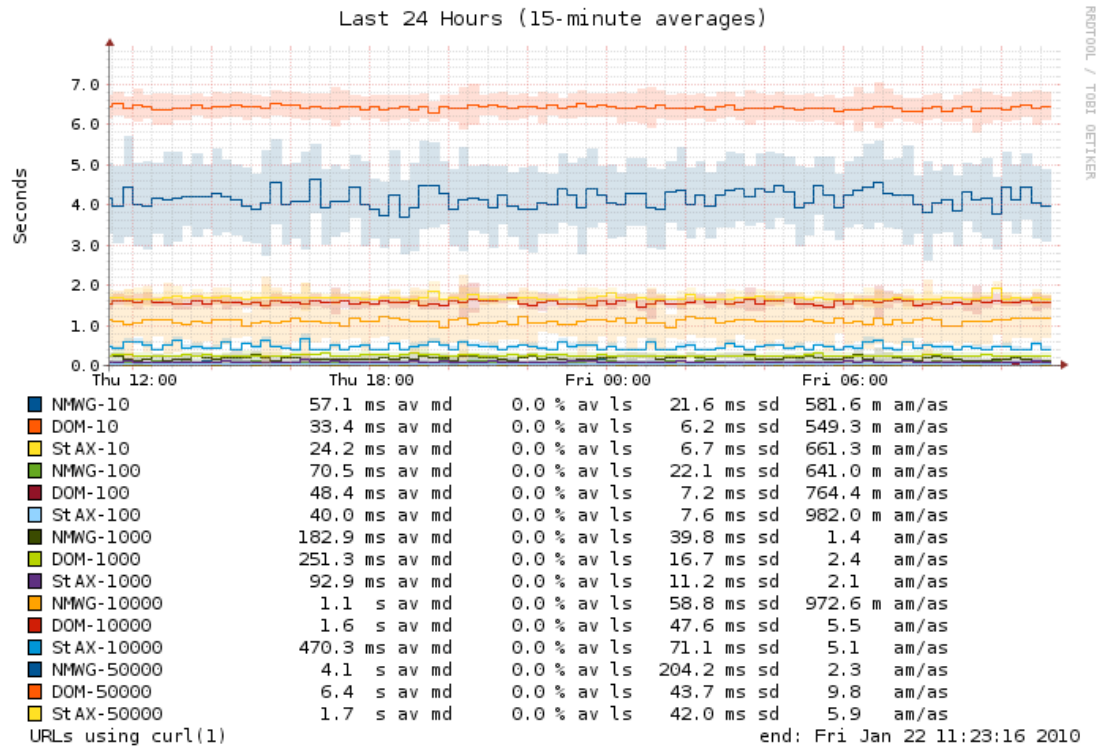
  With the size of response message close to 2Mb, both implementations are considerably slower, than in the case of small messages, but REST implementation win by being 3 times faster.

2.6.2 Performance comparison of different REST implementations of extended ECHO services by message generation and message size

With the observation of message size related to the response time, the following test setup has been established:

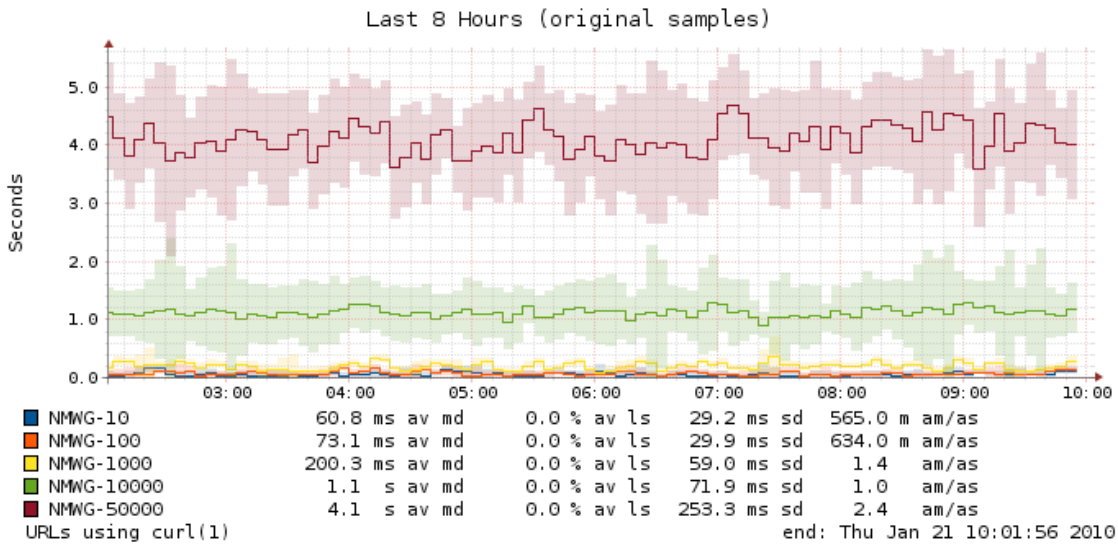
2.6.2.1 Performance of REST services, related to the size of response message

Extended echo response messages generated by NMWG Java classes, DOM and StaX. Each grop has subgroups by response message size in metadata/data pairs.



NMWG classes

NMWG java classes [URL-TODO] are used in all java based perfsonar services.



NMWG-10:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/10

- Message size (metadata/data pairs): 10
- Message size (b): 1942

NMWG-100:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/100
- Message size (metadata/data pairs): 100
- Message size (b): 18412

NMWG-1000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/1000
- Message size (metadata/data pairs): 1000
- Message size (b): 185812

NMWG-10000:

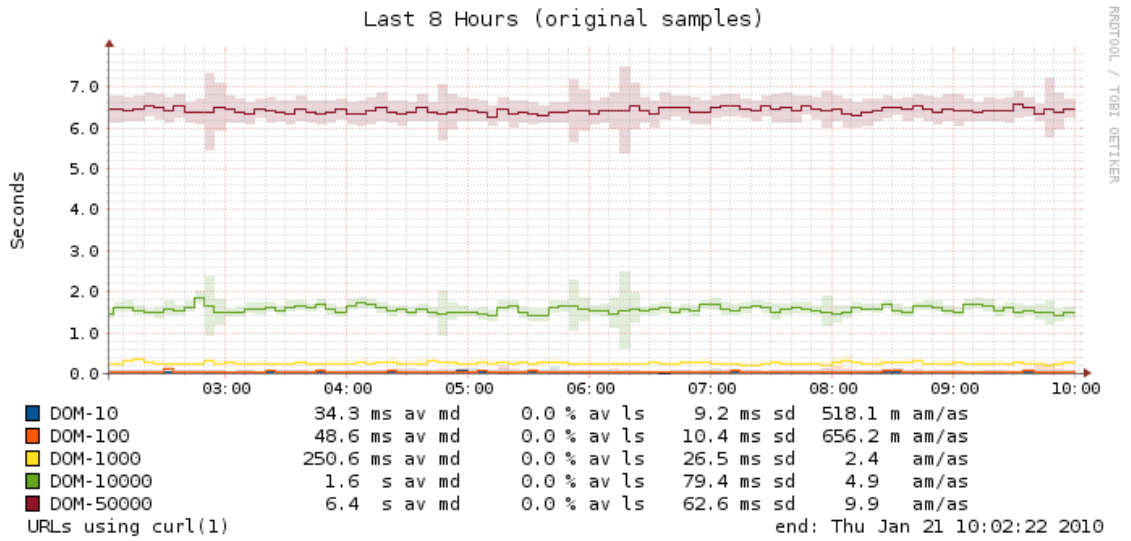
- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/10000
- Message size (metadata/data pairs): 10000
- Message size (b): 1886812

NMWG-50000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/50000
- Message size (metadata/data pairs): 50000
- Message size (b): 9566813

DOM (without NMWG classes)

Performance of REST services, implementing extended echo response by Document Object Model (DOM)



DOM-10:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/10
- Message size (metadata/data pairs): 10
- Message size (b): 1892

DOM-100:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/100
- Message size (metadata/data pairs): 100
- Message size (b): 17372

DOM-1000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/1000
- Message size (metadata/data pairs): 1000
- Message size (b): 174872

DOM-10000:

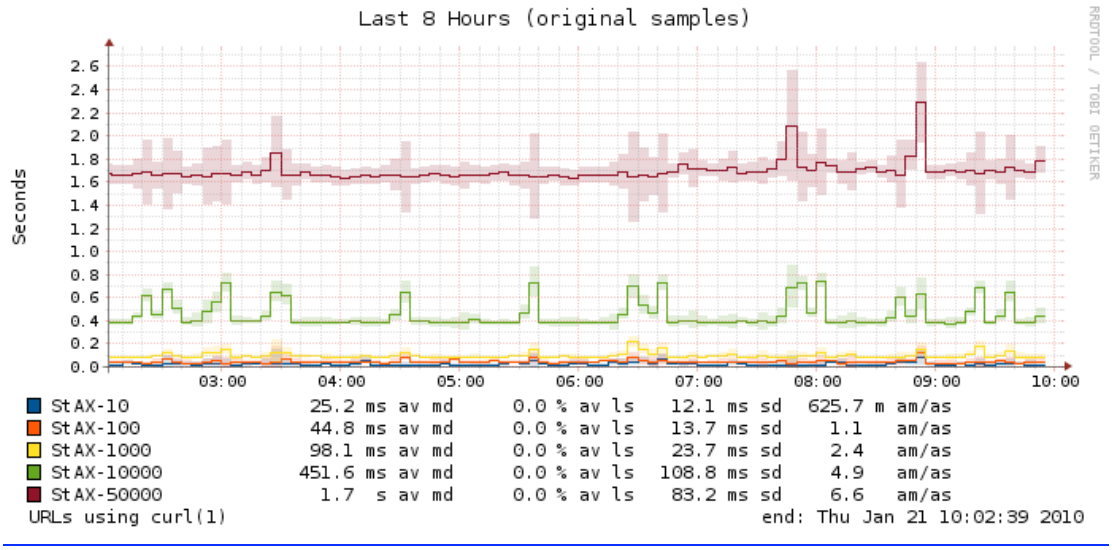
- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/10000
- Message size (metadata/data pairs): 10000
- Message size (b): 1776872

DOM-50000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/50000
- Message size (metadata/data pairs): 50000
- Message size (b): 9016872

StAX

Performance of REST services, implementing extended echo response by Streaming API for XML (StAX)



StAX-10:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/10
- Message size (metadata/data pairs): 10
- Message size (b): 1881

StAX-100:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/100
- Message size (metadata/data pairs): 100
- Message size (b): 17451

StAX-1000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/1000

- Message size (metadata/data pairs): 1000
- Message size (b): 175851

StAX-10000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/10000
- Message size (metadata/data pairs): 10000
- Message size (b): 1786851

StAX-50000:

- REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/50000
- Message size (metadata/data pairs): 50000
- Message size (b): 9066851

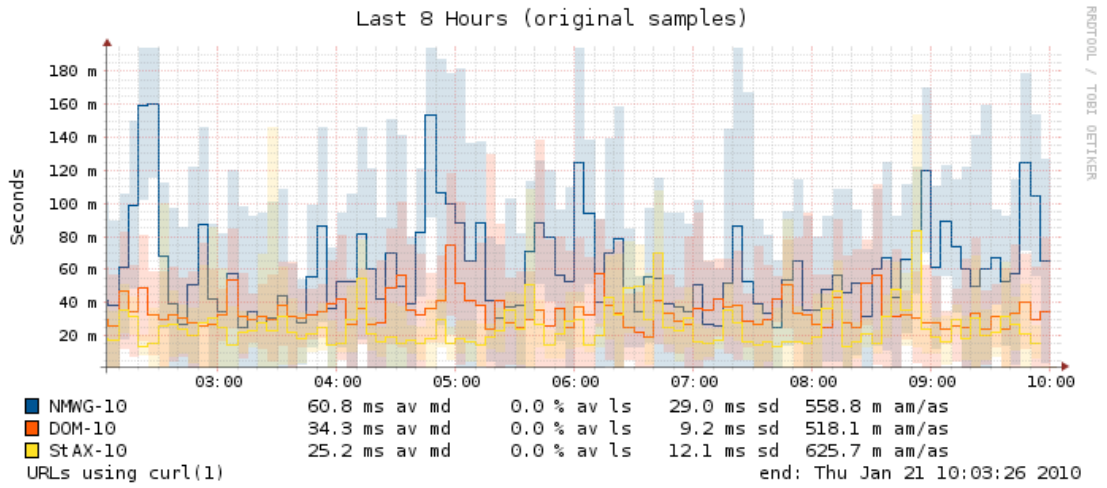
Conclusions

- Test measurements REST services with different message size and using different technologies for generating XML response has been performed.
- REST performance is better than SOAP
- SOAP performance with small message size is relatively close to REST performance and can be considered sufficient
- The technology used to generate XML response has a huge influence on the response time
    - DOM performance is worst, probably because handling XML documents in memory becomes prohibitive with large messages
    - StAX performance is best, because of its streaming nature.
    - NMWG performance (based on StAX parser) is quite close and can be considered good.

2.6.2.2 Performance of REST services for the same message size, using different Java libraries for message generation

Performance of REST services, generating extended echo response messages by NMWG Java classes,DOM and StaX. Response time grouped by response message size.

NMWGvsDOMvsStAX-10



NMWG-10

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/10

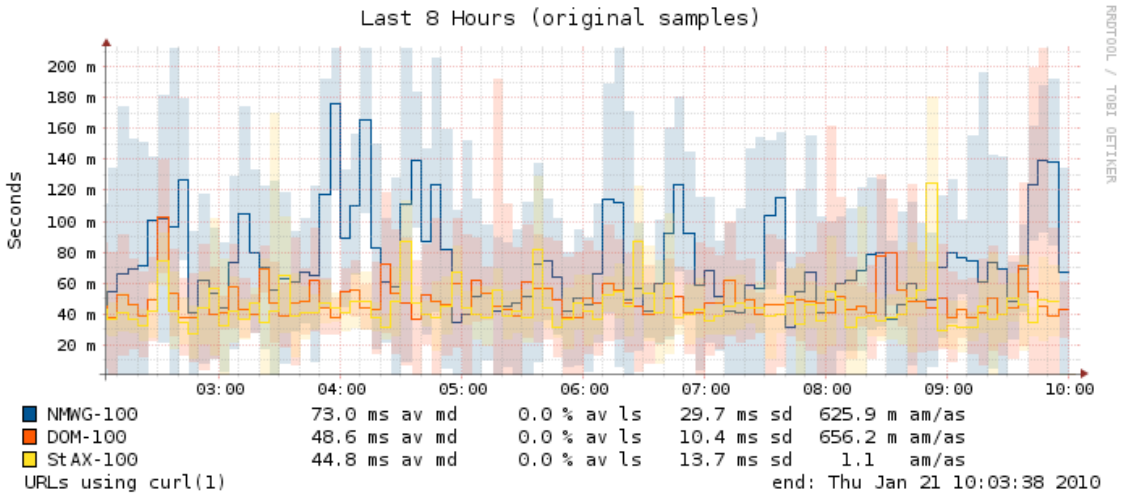
DOM-10

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/10

StAX-10

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/10

NMWGvsDOMvsStAX-100



NMWG-100

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/100

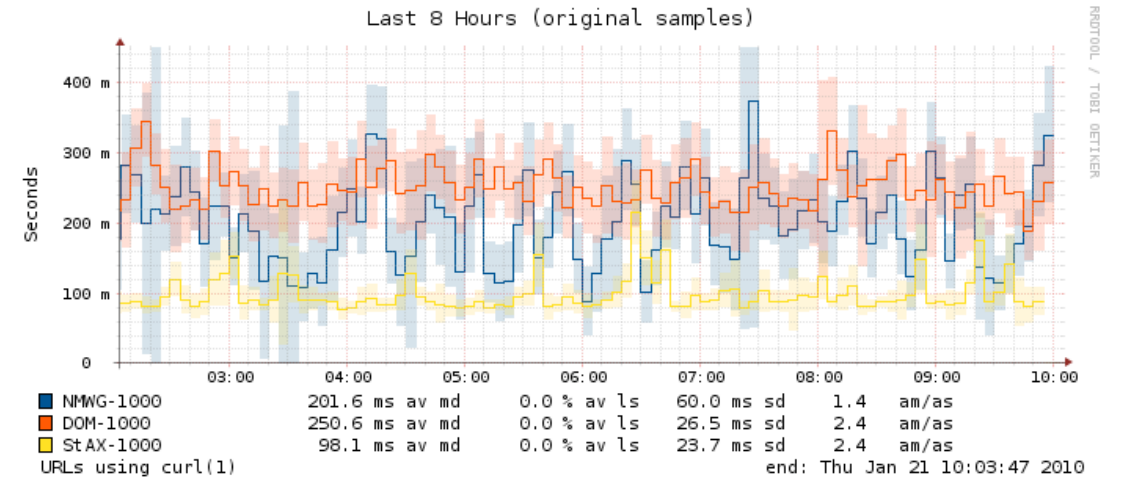
DOM-100

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/100

StAX-100

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/100

NMWGvsDOMvsStAX-1000



NMWG-1000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/1000

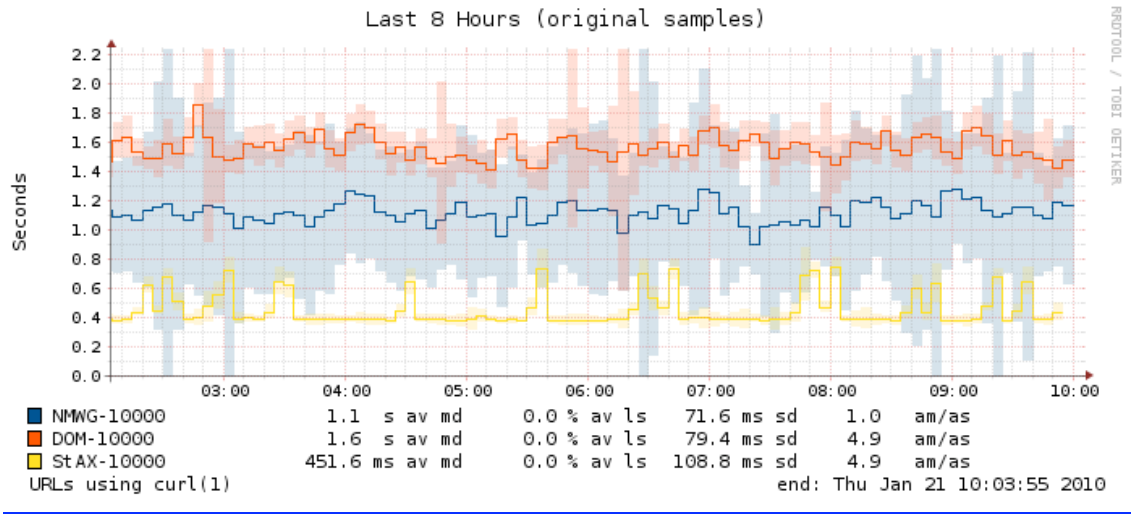
DOM-1000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/1000

StAX-1000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/1000

NMWGvsDOMvsStAX-10000



NMWG-10000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/10000

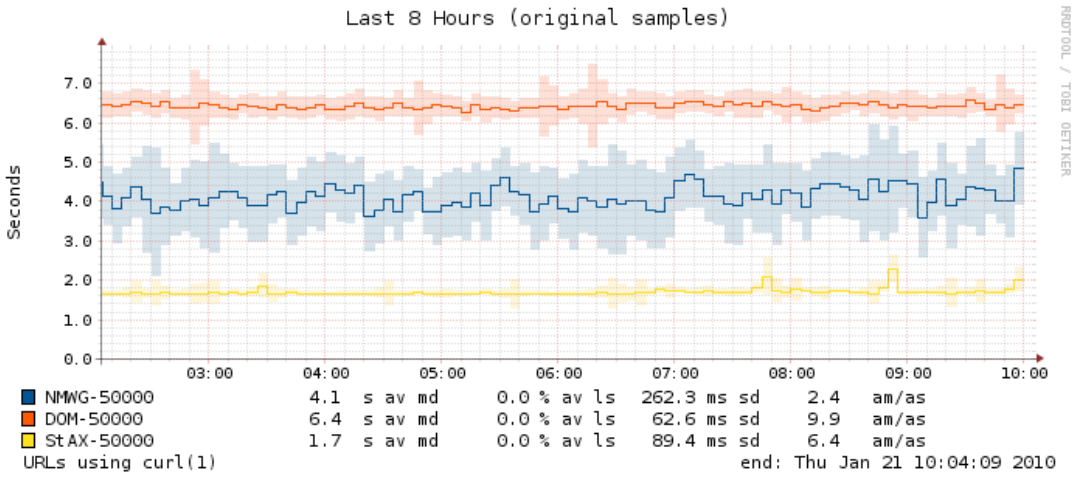
DOM-10000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/10000

StAX-10000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/10000

NMWGvsDOMvsStAX-50000



NMWG-500000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nmwg/50000

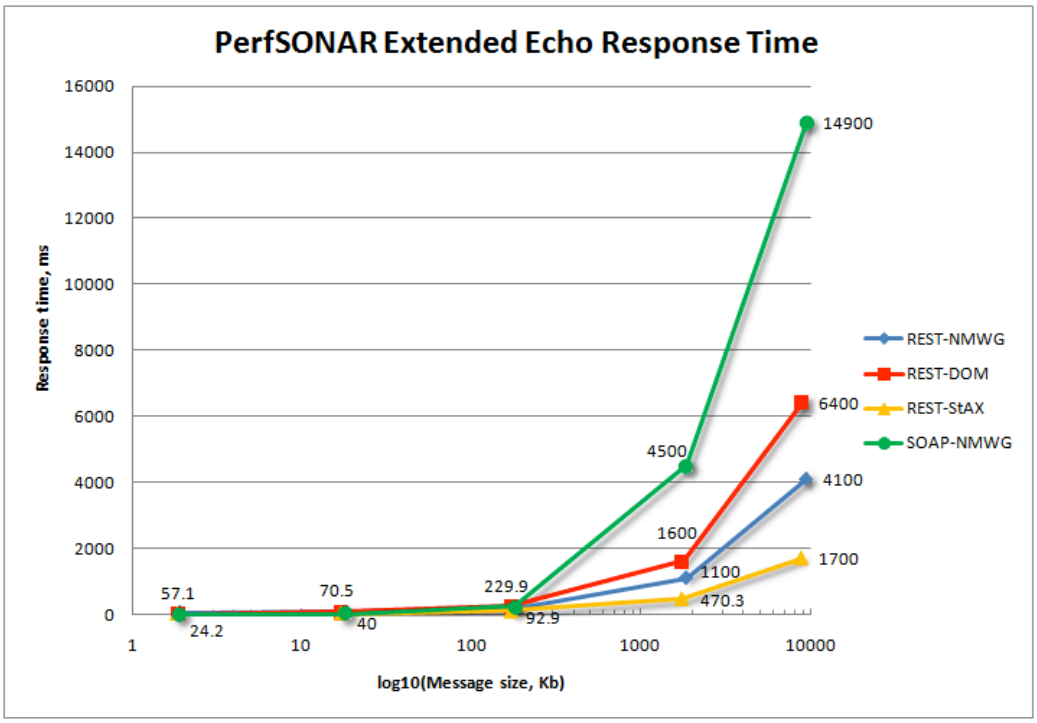
DOM-50000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nonmwg/50000

StAX-50000

REST service: http://uran.acad.bg:8180/rest-test/rest/extendedecho/nodom/50000

2.6.2.3 Summary



The test implementation make use of random data, rather than retrieving real metadata and data from RRD file or databases. The response time figures should be regarded as the minimum response time for the relevant implementations, since RRD/database interactions will add additional latency.

The size of XML responses becomes prohibitively large with the amount of data requested.

2.6.3 Performance comparison of SOAP and different REST implementations of SetupData response

SetupDataRequest/Response provides means to retrieve data for selected performance metric, topology element and time range (e.g. utilization data for an interface and selected time range).

Tests are done for several time ranges. The test services generate response, based on startTime, endTime and resolution and random utilization data. The message side depends (linearly) on the queried time range.

Message size vs response time for one interface, two directions (in/out) , one metric, 5 min resolution

Size (b) Data for the time interval:

- 35776863 1year
- 8823352 3months

- 3040086 1month
- 687826 1week
- 295860 3days
- 99864 24h
- 34498 8h
- 7942 90min
- 2839 15min

2.6.3.1 Performance of SOAP services, related to the query time range

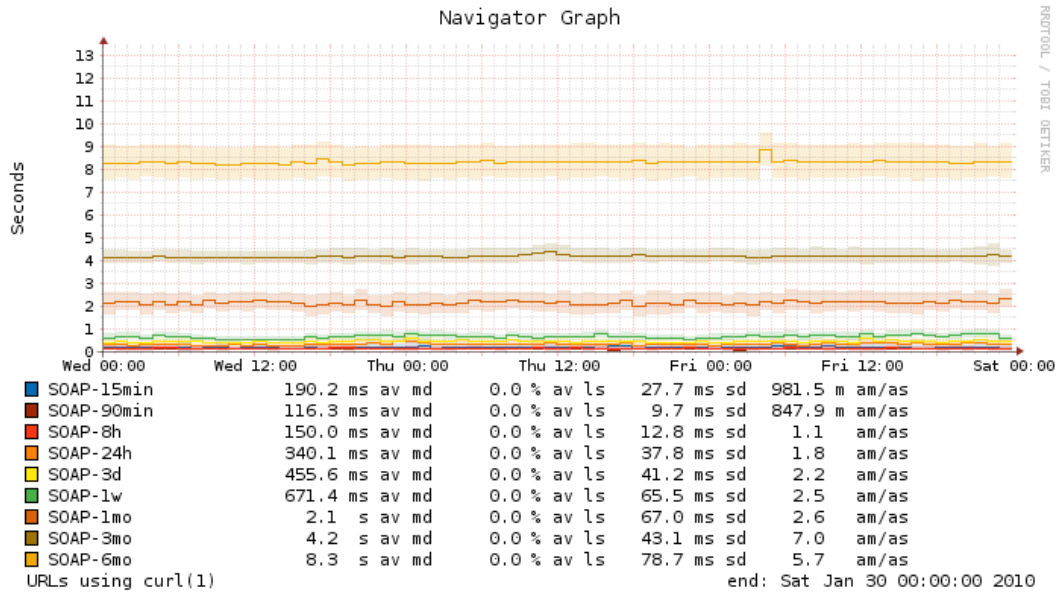
SetupData response messages generated by NMWG Java classes.

2.6.3.2 Performance of REST services, related to the query time range

Extended echo response messages generated by NMWG Java classes and StaX. Each grop has subgroups by requested time range.

SOAP-NMWG

```
curl -X POST -H "Content-Type:text/xml;charset=UTF-8" -d @message
http://host.net/MA/interface/
```



REST-NMWG

NMWG java classes are used in all java based perfsonar services.

- 15 min

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1263945600&endTime=1263946500&resolution=300&mode=nmwg

- 90 min

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1263945600&endTime=1263951000&resolution=300&mode=nmwg

- 8 hours

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1263945600&endTime=1263974400&resolution=300&mode=nmwg

- 24h

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1263945600&endTime=1264032000&resolution=300&mode=nmwg

- 3Days

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1262304000&endTime=1262563200&resolution=300&mode=nmwg

- 1 week

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1262304000&endTime=1262908800&resolution=300&mode=nmwg

- 1 month

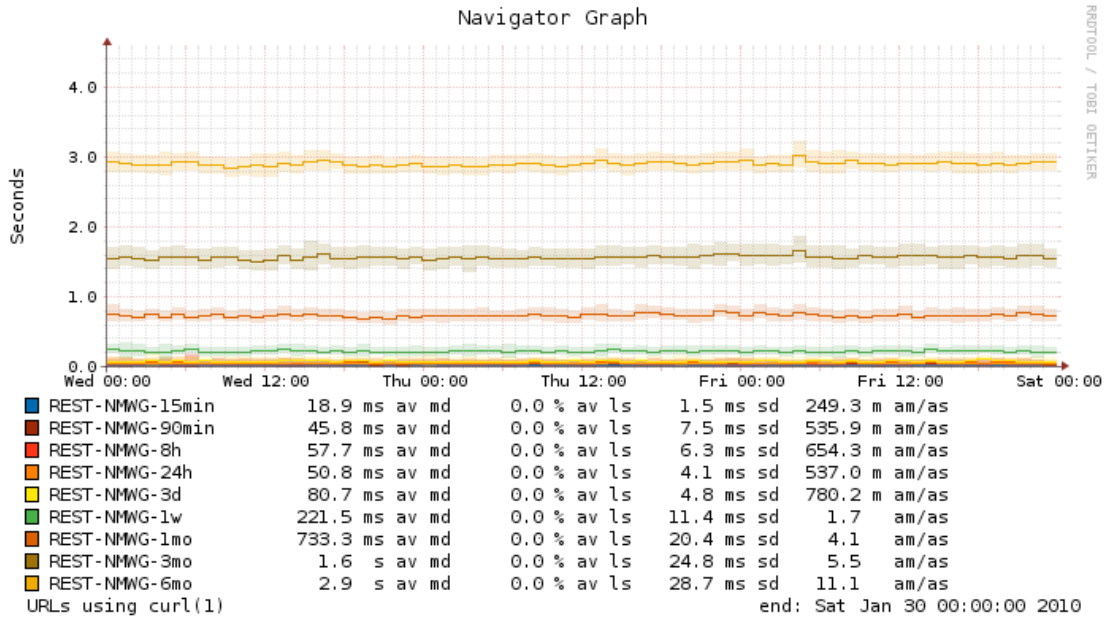
  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1262304000&endTime=1264982400&resolution=300&mode=nmwg

- 3 months

http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1262304000&endTime=1270080000&resolution=300&mode=nmwg

- 6 months

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1230768000&endTime=1246320000&resolution=300&mode=nmwg



2.6.3.3 REST-StAX

- 15 min

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/10.0.0.1/data?startTime=1263945600&endTime=1263946500&resolution=300

- 90 min

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/90min/data?startTime=1263945600&endTime=1263951000&resolution=300

- 8 hours

http://uran.acad.bg:8180/rest-test/rest/ma/interface/8hours/data?startTime=1263945600&endTime=1263974400&resolution=300

- 24h

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/24hours/data?startTime=1263945600&endTime=1264032000&resolution=300

- 3Days

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/3days/data?startTime=1262304000&endTime=1262563200&resolution=300

- 1 week

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/1week/data?startTime=1262304000&endTime=1262908800&resolution=300

- 1 month

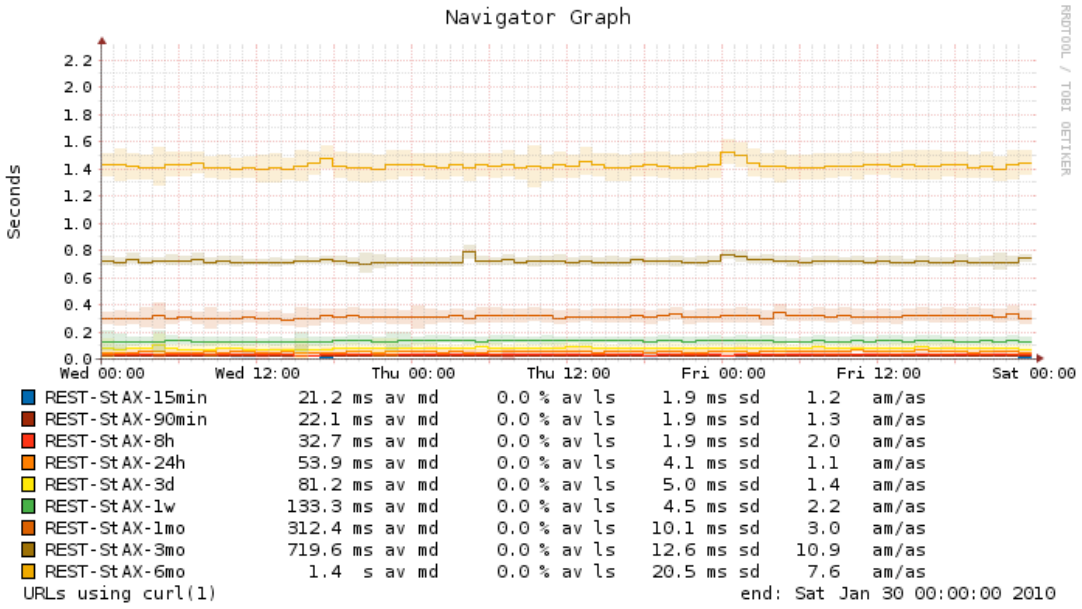
  http://uran.acad.bg:8180/rest-test/rest/ma/interface/1month/data?startTime=1262304000&endTime=1264982400&resolution=300

- 3 months

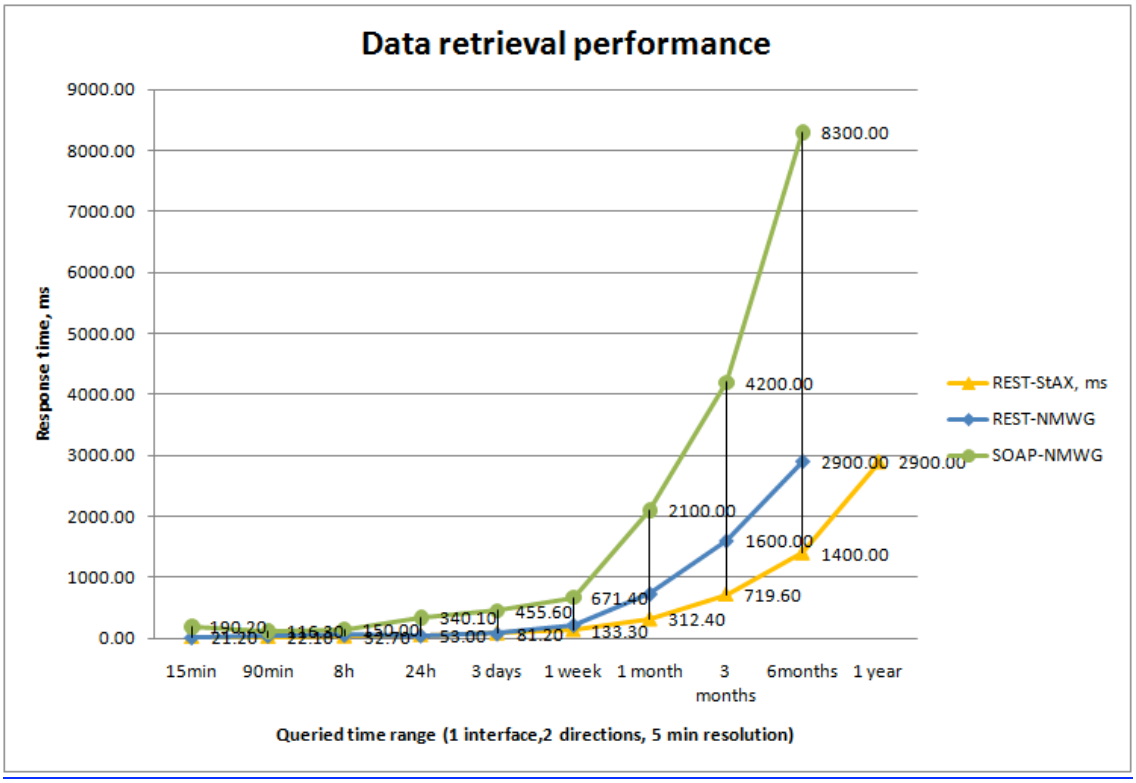
  http://uran.acad.bg:8180/rest-test/rest/ma/interface/3months/data?startTime=1262304000&endTime=1270080000&resolution=300

- 6 months

  http://uran.acad.bg:8180/rest-test/rest/ma/interface/1year/data?startTime=1230768000&endTime=1246320000&resolution=300



2.6.3.4 Summary



2.6.3.5 Conclusions

Same as in previous section. Clients should avoid working with large messages. Services might be implemented to return errors when huge amount of data is requested.

Servers should obey the search parameters, especially "resolution", when requesting monitoring data. This will allow clients to implement more efficient requests of the data.

2.6.4 Performance of SOAP implementations

We have not done any tests to compare existing SOAP libraries, and provide here excerpts from studies done elsewhere:

- Apache Axis2, CXF and Sun JAX-WS RI in comparison

  Published: 20 Oct 2008

  http://predic8.com/axis2-cxf-jax-ws-comparison.htm

  Performance

  Due to the modern streaming XML parser, the performance of all three SOAP engines is very well. The ping time for a locale roundtrip is about 2-4 milliseconds (message size about 3KB, Dual Core Notebook). Therefore the time delay by the SOAP communication is negligible in many projects.

  This is in agreement with our initial tests, done with small Echo messages, locally from the machine running the services. However, the real use case involve larger message size and network latencies.

- Java Web services

  Metro vs. Axis2 performance

  Published: 19 Jan 2010

  http://www.ibm.com/developerworks/java/library/j-jws11/

  Compares performance of Metro and Axis2 with and without WS-Security, with small and large response messages.

  Response time 1-16 seconds

  Metro offers performance equal to Axis2 for plain-text message exchanges and much better performance than Axis2 when WS-Security is involved.

Response time within 1-16 seconds is consistent with our tests for perfsonar SOAP services performance

- Coldfusion SOAP vs REST benchmark

  http://www.lynchconsulting.com.au/blog/index.cfm/2008/3/13/CFMX--SOAP-vs-REST-benchmarks

  March 2008

  5 fold performance increase SOAP -> REST

- Latency Performance of SOAP Implementations,IEEE CLUSTER COMPUTING AND THE GRID 2002

  Compares SOAP vs RMI and CORBA, with small and large messages, same and separate client and server,

  Published 2002

  http://www.caip.rutgers.edu/TASSL/Papers/p2p-p2pws02-soap.pdf

Might be outdated, but interesting to see the test setup and numbers; SOAP slower 10-100 times than predecessors.

2.7 Conclusions

- The response time of SOAP service was slightly higher than REST services on very small response messages and almost 10 times higher than REST with the largest message size tested. SOAP response time increases with response message size
- The performance critically depends on the technology used for generating XML messages. Streaming XML (StAX) is always better, with NMWG performance close.
- The test StAX implementation doesn't store anything in memory, hence the good performance. But it should be regarded as an estimate of the best response time, in real implementations one could hardly do any processing without having objects in memory.
- It is possible to define REST interface to perfsonar SOAP services. The proposed REST interface doesn't strive to establish direct mapping between REST and SOAP messages, but rather follows REST design guides to represent resources by URLs and define operations on resources. This is in contrast to the SOAP style of having a single entry point and deciding on operations by inspecting the incoming XML message.
- A topology element is represented by REST resource and available via URL. Metadata and data of the resource can be requested, by sending HTTP GET requests to this URL and specifying

query parameters. The default resource representation is text/xml, but multiple formats can be supported by HTTP Content-type header. Write operations to the resource are performed via HTTP PUT, POST or DELETE requests.

- REST and SOAP interfaces to perfSONAR monitoring data **may exist together and allow clients to use the preferred interface**. This is the current practice with most popular services worldwide. Available interfaces could be specified when registering to a Lookup Service.
- The Authentication and authorization model is quite similar in REST and SOAP. The security tokens and profiles are the same and the 90% of the source code could be re-used (or releasing an AA base library for both implementations). For the future, OAuth should be considered to use in perfSONAR. OAuth WRAP has been released recently and it fits perfectly for perfSONAR.