SOAP-binQ: High-Performance SOAP with Continuous Quality Management

Balasubramanian Seshasayee, Karsten Schwan, and Patrick Widener
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{bala, schwan, pmw}@cc.gatech.edu

***Abstract*— There is substantial interest in using SOAP (Simple Object Access Protocol) in distributed applications' inter-process communications due to its promise of universal interoperability. The utility of SOAP is limited, however, by its inefficient implementation, which represents all invocation parameters in XML, for instance. This paper aims to make SOAP useful for high end or resource-constrained applications. The basic idea is to replace SOAP's XML/Ascii-based parameter representations with binary ones. Using SOAP's WSDL parameter descriptions, XML-based parameters are automatically represented as corresponding structured binary data, which are then used in all client-server communications. Data is up- or down-translated to/from XML form only if and when needed by end points. The resulting SOAP-bin communication protocol exhibits substantially improved performance compared to regular SOAP communications, particularly when used in the internal communications occurring across cooperating client/servers or servers. Gains are particularly evident when the same types of parameters are exchanged repeatedly, examples including transactional applications, remote graphics and visualization, distributed scientific codes. A further improvement to SOAP-bin, termed SOAP-binQ, addresses highly resource-constrained, time-dependent applications like distributed media codes, where scarce communication bandwidth, for example, may prevent end users from interacting in real-time. SOAP-binQ offers additional quality management functions that permit SOAP to reduce parameter sizes dynamically, as and when needed. The methods used in size reduction are provided by end users and/or by applications, thereby enabling domain-specific tradeoffs in quality vs. performance, for example. An adaptive use of SOAP-binQ's quality management techniques presented in this paper significantly reduces the jitter experienced in two sample applications: remote sensing and remote visualization.**

I. Introduction

SOAP (Simple Object Access Protocol) is an XML-based remote invocation protocol designed for flexibly composing Internet applications [1]. Its use of text-based messaging (XML) and of HTTP or SMTP for communication provides universality and interoperability, but also creates substantial performance limitations. This reduces the utility of SOAP for large classes of applications. An example is the use of in-vehicle camera sensors to report on traffic or emergency situations, using wireless links with limited bandwidths. Or consider amateur astronomers who use wide-area networks (including from remote sites) to share their captured images and track, in real-time, low-light emitting objects like asteroids[2]. Other examples include high end business applications like the operational information systems presented in [3], [4] and real-time collaborations in which scientists jointly visualize simulation or sensor data to better understand or interpret certain phenomena[5], [6], [7], or in which engineers jointly design airplane wings or similar high performance parts[8]. These applications have in common the use of substantial computation in conjunction with large-data communications, in the presence of limited networking and/or CPU resources. Computations include data selection, filtering, and transformation, as exemplified by image processing pipelines in sensor systems and by the graph-structured data manipulations used in scientific collaboration (e.g., to transform scientific data to match a specific end user's needs[9], [10]).

This work is funded in part by the National Science Foundation's ACIR program.

In all of the applications cited above, SOAP's use of XML with its ascii-based parameter representations can be prohibitively expensive. Costs include server loads due to the need to manipulate and translate ascii data and communication overheads due to the relatively 'bulky' ascii representations of invocation parameters. This paper describes how to attain high performance and low overhead for SOAP-based communications. The approach taken leverages previous work on adaptable object communications [11], [12] and recent results on efficient binary representations for XML-based data [13], [14]. Specifically, the idea is (1) to associate with SOAP communications application-specific handlers that can be used to manipulate parameters used for invocations and transform parameters into more suitable formats, and (2) to make such handlers efficiently executable, even for large data volumes, by using binary representations for both the handlers and the XML data they manipulate. The outcomes are the SOAP-bin and SOAP-binQ protocols described next.

The basic SOAP-bin protocol uses *conversion handlers* to implement XML-to-binary conversions, if needed. It can be used in multiple ways:

- *SOAP-bin - high performance mode*: SOAP-bin can be used to implement server-based communications, as with the internal communications used in web server front-to-backend communications [15]. In such 'internal' communications, SOAP parameters never appear in XML forms. Instead, they have already been converted to corresponding binary forms, for request and for result parameters. Measurements presented in this paper demonstrate that the performance of SOAP-bin used in this mode is com-

petitive with the standard invocation mechanisms used in today's client-server interactions, like Sun RPC[16].

- *SOAP-bin - interoperability mode*: when servers receive requests from and return data to external clients, clients use standard XML data, but servers use binary data, in order to reduce server loads. Measurements presented in this paper demonstrate that the required 'one-sided', just-in-time, client-side data conversions permit SOAP-bin to outperform standard implementations of SOAP, both with respect to the communication bandwidth consumed (in particular for wide area links) and the load imposed on servers.
- *SOAP-bin - compatibility mode*: the lowest performance case for SOAP-bin is where both end users, such as clients operating in peer-to-peer mode, must translate XML text to binary data, in order to be able to operate on such data with standard tools. Adobe's PDF format and image formats used by graphics tools are typical examples.

  To evaluate such costs, we present measurements in which XML data is first converted to binary form[1], transferred, then again converted back to XML, and we compare the performance of SOAP-bin in this mode with the performance of regular SOAP and of SOAP that uses online compression to reduce data size. Both communication overheads and client loads are reported.

An enhanced version of SOAP-bin, termed SOAP-binQ(uality), permits applications to enrich basic conversion handlers with configurable, application-specific functionality. The resulting *quality handlers* are code modules that take as inputs both the binary representations of SOAP parameters and *quality attributes* that determine handlers' behaviors. Attributes may be changed on a per invocation basis, and they may be provided by applications and/or by the underlying network/system levels. An example from the scientific domain is an application-provided data filter that adjusts the amounts (and therefore, the quality – in terms of resolution) of the data sent to the current needs of clients and/or also to currently available network resources. Such client- and network-aware data filtering has been shown useful for a variety of applications and platforms, including to control the data volumes required for remote 3D visualizations across the Internet [18], [19]. Another example is an image filter that crops images provided by clients to focus on areas of current interest in military applications [8], [20].

In measurements presented in this paper, we apply configurable quality handlers to filter both sensor data and scientific data, and we demonstrate the performance improvements derived from the adaptive use of such filters for end user applications. Improvements are due to the improved ability of SOAP-binQ compared to SOAP-bin to control the data volumes exchanged as call or return parameters, thereby better dealing with runtime variations in communication and server resources. This is particularly important for versions of SOAP-binQ used in interoperability or high performance modes.

Initial results of this work are encouraging. With the SOAP-binQ infrastructure in place, message transmission times are improved by a factor of about 15 for 1MByte message sizes. Marshalling and unmarshalling times and therefore, the loads imposed on server systems due to SOAP use are also reduced significantly. Both improvements are due to the use of binary formats for transporting SOAP parameters. Finally, the distributed large-data applications evaluated in this paper experience more uniform response times in congested networks when using SOAP-binQ, due to its ability to dynamically adjust the data volumes sent to available network resources.

The remainder of this paper elaborates on the design, implementation and behavior of SOAP-binQ and compares it with XML-based SOAP implementations. Related work is described in Section II. Section III deals with the overall design and implementation details. Experimental results accompanied by discussions are presented in Section IV. Conclusions and future work appear in Section V.

II. Related Work

The growth of XML-based web services is increasing the interest of high performance end users in these technologies. Efficiency issues with XML and with the SOAP protocol that relies on it, however, have inhibited technology adoption. Inefficiencies with XML are described in detail in [21], which identifies ASCII conversions of digits as one of the most significant bottlenecks associated with XML. A solution approach first demonstrated in [13] is to use binary encodings of XML for large-data objects [21], [22], with resulting performance improvements of up to 75% in terms of reductions in processing costs and message sizes achieved in [23]. We adopt this approach [13], by automatically converting XML schema-based data descriptions to binary forms (and vice versa) [14], 'just in time' (i.e., when needed by end user applications).

Efforts to improve XML performance are complemented by the development of efficient implementations of remote invocation protocols, like [24], [25], [26], [27]. [28] implements a fast RMI protocol that is interoperable with Java/C++-based SOAP implementations, with an RMI system that uses SOAP for its communications. Performance gains rely in part on the use of the XML Pull Parser [29], a stream-based fast XML parser. However, this work targets interoperability, by adopting XML for communication between heterogeneous systems. Our work, in contrast, targets performance, while retaining interoperability.

Interactive scientific applications, remote instrument usage, remote sensing, and multi-media applications typically require quality of service (QoS) support. The QoS mechanisms defined for SOAP (version 1.2 [30]) focus on QoS guarantees provided by the transport level and/or by intermediaries involved in SOAP processing. In accordance with ongoing research on QoS in middleware [31], [19], our approach generalizes such transport-focused QoS to enable applications to directly specify and manipulate the data sent and received by SOAP participants. This permits application-specific tradeoffs in the amounts and therefore, quality of information sent and received vs. the available network and processing resources on participating machines[18]. The handlers and quality attributes

[1]The Expat toolkit [17] is used for parsing XML

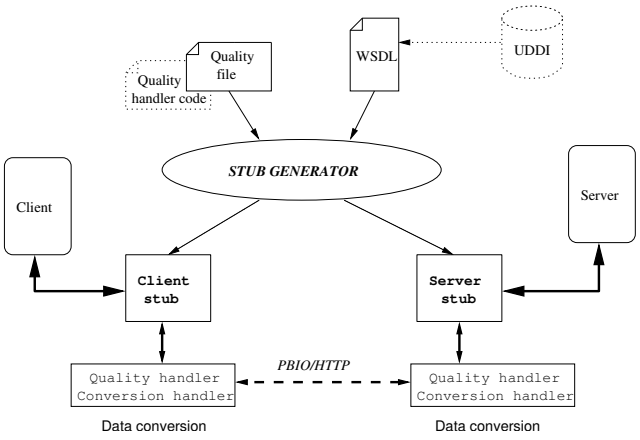

Fig. 1. Software Architecture of SOAP-binQ.

used in SOAP-bin have been used in grid middleware to implement client-specific behaviors[18] and in real-time object systems [32], [33] to control the timing behavior of invocations. The contract-based quality support used in SOAP-binQ is also used in [34], [35]. Within RMI and in object-based distributed operating systems, related functionality, termed subcontracts, has been used to control invocation behavior like automatic replication and server selection [12], [36], but such mechanisms do not support per invocation parameterization.

III. Software Architecture of SOAP-binQ

A. Architecture Description

Scientific computing in distributed environments must efficiently deal with data heterogeneity, at the level of machine architectures and for applications. In scientific visualizations, for instance, the data produced by an application must typically undergo a sequence of transformations before it is displayed to a particular end user [9]. Soap-binQ deals with data heterogeneity by *describing* SOAP parameters with XML, and by *operating* only on the binary representations of such data. The intent is to eliminate or reduce serialization and deserialization costs, while also maintaining the universality implied by the use of XML.

Figure 1 shows the overall design of SOAP-binQ. It consists of a WSDL compiler that generates the client and server side stubs, with conversion handlers for XML/binary interconversion. The specific binary format used is PBIO, which in previous work, has been shown to efficiently represent the structured XML that constitute SOAP parameters[13]. Quality attributes are specified in a *quality file*, which is compiled jointly with the WSDL file to generate stub files. The information contained in this file are the data types of the parameters sent in SOAP messages in conjunction with various quality attribute values. It also references the quality handlers specified by end users (when present) or generates trivial quality handlers otherwise.

Figure 2 illustrates the transformations a message can undergo during a simple SOAP request/response interaction. The figure also depicts the different options available for such interactions. These options exist because the application layer

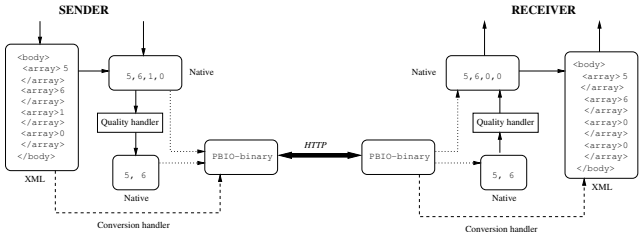

Fig. 2. Data flow in Soap-binQ communications.

can communicate with the transport layer either via XML data or via native data. In the first case, data is converted to native form. Once this is done, handlers can perform data transformations or filtering. This is discussed in more detail in the following paragraphs. The data format used for data transmission via an HTTP connection is PBIO[14], which deals with heterogeneous machine architectures via a 'receiver makes right' paradigm, thereby avoiding the symmetric up and down translation used in most grid middleware. At the receiving end, the entire process takes place in reverse order, providing the data back to the application layer in the appropriate format.

B. SOAP-binQ Implementation

a) SOAP-bin.: Our prototype implementation of SOAP-binQ is based on *Soup* [37], a version of SOAP developed by Ximian, written in C. Soup is modified to read in both a handler file and a WSDL file, and from these files, produces the modified stubs and support files that implement the desired quality management functionality. Specifically, the WSDL compiler reads XML typecodes from the WSDL file, generates the client-side and server-side stubs as well as a file with support functions and a header. From these, server- and client-side applications are built. The schema used in Soup identifies the basic types as integer, char, string and float, and it allows the user to build more complex types through the use of lists and structs. Soup uses *libxml2* for conversion from and to XML. Our implementation alters this conversion to use the PBIO binary data format.

PBIO allows the sender to send data in its native format, using dynamic code generation to automatically generate code that performs the format conversion at the receiver side. PBIO data is defined through what is known as formats. The formats are similar to XML schemas, in that they define how data is structured. Every PBIO transaction begins with a registration of the format with a "format server", which collects and caches PBIO formats. Whenever a new type is encountered, the application consults the format server to interpret the message. This transaction occurs only once, since the format is cached locally thereafter.

The WSDL compiler generates PBIO formats based on the description given in the WSDL file, and these formats are used in the binary transmission of SOAP parameters. The example in Figure 3 illustrates this conversion, for an example in which raw sensor data represented in ppm format is transported to servers that perform analyses on these images (note that in

Fig. 3. Generation of PBIO formats.

such cases, it is not suitable to use lossy compression methods like JPEG).

b) SOAP-binQ: Quality Files.: The purpose of SOAP-binQ's quality support is to allow transformations to be applied to messages, as when matching message sizes to available resources. Since many applications face tradeoffs between data accuracy and speed, it is useful to provide transport-level support for varying the volume of data exchanged between participants. SOAP-binQ implements this by permitting developers or end users to create a 'quality' file accompanying the WSDL information. This file dictates the data precision to be used under differing resource availabilities. For instance, data with a specified number of array values could be replaced by a smaller sized array, if the loss in precision is not as critical as the time that could be lost serializing, transmitting and deserializing a larger array. As an example, in this paper, we demonstrate the tradeoffs existing for an image server used in astronomy applications. Here, the resolution of the image is adjusted based on the quality of the network link. Network quality is represented by the cumulative RTT values for SOAP requests.

Quality files are created by end users or provided by domain-knowledgable developers. Such files contain handlers, the QoS attributes in the form of intervals of quality attributes, where round trip time - RTT - is the QoS attribute used in our current implementation.

The template for a quality file, sample attributes, and the corresponding message types is depicted below:

quality_attribute_1 quality_attribute_2 - message_type_0
quality_attribute_2 quality_attribute_3 - message_type_1
quality_attribute_3 quality_attribute_4 - message_type_2

Typically, a set of message types, described in the quality file, corresponds to a single, larger message type, which is the actual request/response message used by the application layer. During the sending process, the transport looks up the quality file to find the right message type to be sent. It then copies the relevant fields (those fields that are common to the data structure acquired from the application and those to be sent) and ignores the rest. At the other end, the application layer expects the original message sent out by the sender. Hence, the relevant fields are copied from the message received from the transport, and the remaining entries are padded with zeroes. This feature permits legacy applications to be integrated seamlessly with SOAP-binQ, but it could be removed by transmitting quality attributes along with SOAP communications[18] and then using them to match sender with receiver actions.

In our applications, the message types with the largest sizes are used along with small RTT intervals, and the smaller types are used with larger RTT intervals. This reflects the assumption that a lower RTT indicates good network conditions, and hence larger message types can be sent, and vice-versa. Thus, the quality file is used both by the server side and client side stubs, to determine the message type and corresponding size to be used under each circumstance.

When there is no direct correlation between message types (for instance, there are fields in the smaller message type that do not occur in the larger type), or if complex handlers are to be used to transform data (applying resizing handlers to images, for example), the necessary quality handlers are specified by the user along with the quality file. Such handlers are directly included in the stub code, and the ones used as quality handlers in lieu of the basic conversion handlers generated from WSDL files.

The policy-level information encoded in quality files is formulated based on the requirements of the application. Since this information consists of different data formats used in messaging, the data types involved in messaging must be known when the quality file is created. In the future, we foresee the designer providing a quality file along with the WSDL file, through UDDI or a similar WSDL repository. This would let the user directly access the service, without knowledge of the actual message types used in data transmission.

c) Quality Attributes.: As described above, quality files relate quality attributes to message types, where RTT is used as the monitored value in our current examples. However, a monitored attribute can use any value that is suitable for triggering changes in data quality, including attributes specified by end users, such as desired image resolution[18]. Quality attributes can also be specific to an application, as demonstrated with a web server in [38]. Other attributes suitable for the sample applications and execution environments used in this paper may capture CPU load, by measuring marshalling or unmarshalling costs, memory consumption, or similar factors. The reader is referred to [39] for a more detailed discussion of how to link monitored attributes to changes in the quality of service (QoS) or quality of information (QoI) in quality-managed applications.

d) Dynamic Quality Changes.: Soup was modified to read in a quality file and a WSDL file, and to turn out the modified stubs and support files that implement runtime quality management. While the quality file draws the correspondence between quality attributes and message types in the stubs, it may be necessary for the application to change quality management at runtime. Our current implementation does not permit runtime changes in the handlers or policies used for quality management, but it does permit applications to dynamically update the values of quality attributes. This is done via the API call *update_attribute()*. This function can be useful when, for instance, the application changes

its sensitivity. For example, an application that obtains stock quotes over a period of time, from a server, might use an attribute that dictates the granularity of the data. If the client decides to get finer grain data at the expense of additional tolerance for delay, the value of a quality attribute would be changed accordingly. Note that in all such cases, while the messages to be included in the quality policy are determined by studying the application's needs, performance testing is required to determine suitable values of quality attributes.

IV. Experimental Evaluation

We illustrate the performance of SOAP-binQ through measurements obtained in the following experiments. First, we demonstrate the performance of SOAP-bin by comparing it with Sun RPC (which uses the XDR data representation). The intent is to show that the basic performance of SOAP-bin is competitive with that of standard client-server communication mechanisms. Next, we systematically characterize the marshalling, unmarshalling and transmission costs of binary SOAP and compare it with XML-based SOAP, thereby evaluating SOAP-bin's high performance and interoperability modes. Next, these results are compared with XML-based and with compressed XML implementations of SOAP, using SOAP-binQ in compatibility mode. Last, the performance of SOAP-binQ and the utility of runtime quality management are evaluated with representative applications.

A. Sun RPC vs SOAP-bin

The overall performance of SOAP-bin is compared to TCP-based Sun RPC. Experiments are conducted between a 2.2GHz Pentium IV with 512MB RAM running Linux kernel 2.4.18-27.7.x, and a dual 750MHz SPARC with 2560MB RAM running SunOS 5.8, connected by a 100Mbps Ethernet link. Overall cost for marshalling, transmission and unmarshalling, of arrays and nested structs, of different sizes, are measured. The justification for these data types appear in Section IV-B.

The results shown in Figure 4 demonstrate that SOAP-bin's performance is close to that of Sun RPC when array data are used, but that Sun RPC outperforms the former in the case of nested structs (by about a factor of 5.4 in the worst case). The delay is mainly due to SOAP-bin's use of HTTP for its transactions. It can hence be concluded that, while SOAP-bin cannot compete with Sun RPC, the performance attained is reasonable for Internet applications. However, since PBIO has the added advantage that the sender can issue data in its native binary format, servers dealing with PBIO data can scale better, especially for large data sizes[14].

B. SOAP-bin: Microbenchmarks

A series of microbenchmarks evaluate the performance of SOAP-bin with different data types and sizes. Two sets of entirely different data types are used, one representing scientific applications via arrays of different sizes, and a second representing business applications via a nested structure of varying depth. Arrays are at one end of the spectrum, where marshalling simply means enumerating the elements - enclosing them with tags in the case of XML and simple enumeration in the case of PBIO. At the other end of the spectrum are nested structs, since marshalling and unmarshalling require recursive function calls and the addition of tags (in the case of PBIO, this involves traversing the struct and copying the fields to a send buffer).

Experiments also use different network links, one representing a high end link in a company's intranet and the other a low end, remote Internet link. The high end link is a 100Mbps LAN link in one of our laboratories; the low end link connects a laboratory machine to a home machine via ADSL. In both cases, the client machines are 2.2GHz Pentium IV with 512MB RAM running Linux version 2.4.18-27.7.x. The server has the same configuration as the client in the case of the 100Mbps link, but it is a 1.9GHz Pentium IV with 512MB RAM, running Linux kernel 2.4.18-3 for the ADSL link.

Measurements are derived from sets of 10-1000 experiments, reporting the averages over all readings, after discarding the first set (to eliminate cold start effects). Variances are less than 1% on the average and are therefore, not reported.

e) SOAP-bin: Marshalling/Unmarshalling Costs.: Figure **??** shows the overall costs for conversion of data between native and PBIO, XML compression and XML and PBIO, and also the sizes of the resultant data. Compression is achieved using Lempel-Ziv encoding. The figure indicates the inordinately large sizes for XML data as compared to equivalent PBIO messages. The XML parameters generated are about 4-5 times the size of the corresponding PBIO messages, in part due to redundant tags (i.e., tags enclosing every element of an array). The difference is even greater for the nested structure, since its document size increases exponentially, where elements are enclosed within tags at each level of the struct. This results in a ninefold increase in the size of the XML document vs. the corresponding PBIO message. This result agrees with the observations in [40]. Compressed XML is mostly the same size as, and sometimes smaller than the equivalent PBIO data. This is in part due to the highly structured nature of the data. Compression, however, may not be the solution if the data available to the transport is in the native format of the sender.

The time taken for PBIO encoding and decoding is relatively small when compared to data transmission costs, especially with larger data sizes. This is due to the fast nature of the participating machines, making communication latency the restricting factor. This effect is more pronounced in the case of a slower connection, the ADSL, where the gap between PBIO encoding/decoding and the transport costs is quite high, even in a log scale graph.

The encoding and decoding times for the nested struct are perceptibly high compared to an array of equivalent size. One factor that doesn't appear in the graph is the initial cost of registering a format. As noted before, PBIO needs to "register" a format with a format server, during the first message transfer. This acts as a hand shake between the sender and the receiver, so the receiver knows what type of message it gets and how to decode it. The additional cost incurred for the first message is negligible when small formats are used, and it becomes significant only for very deeply nested structures. Subsequent exchanges of messages are compared against cached formats, resulting in much faster decoding.

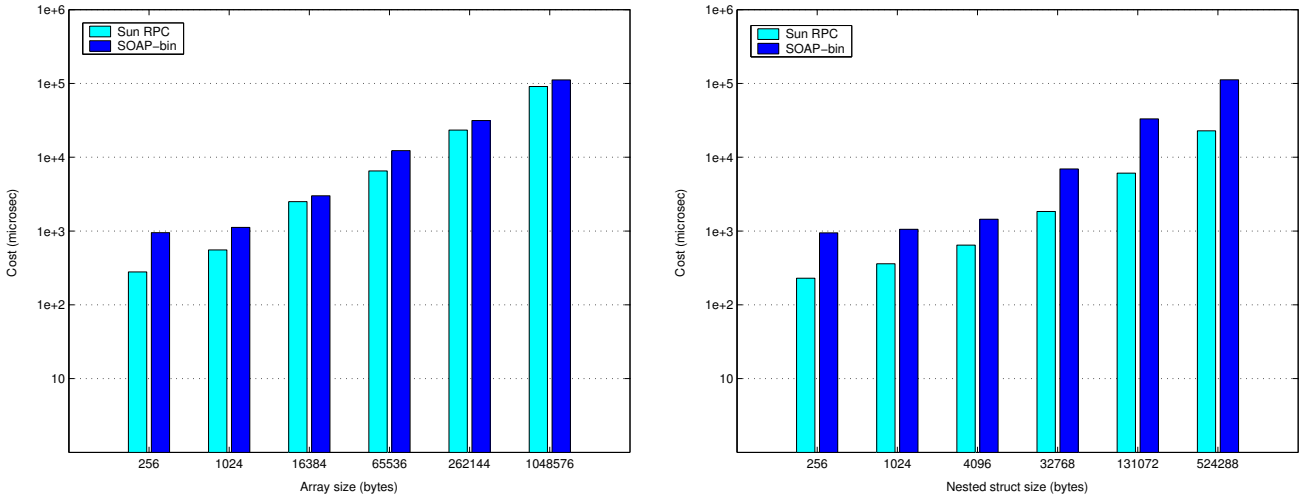

Fig. 4. Overall times for Sun RPC and SOAP-bin for (a) integer arrays and (b) nested structs.

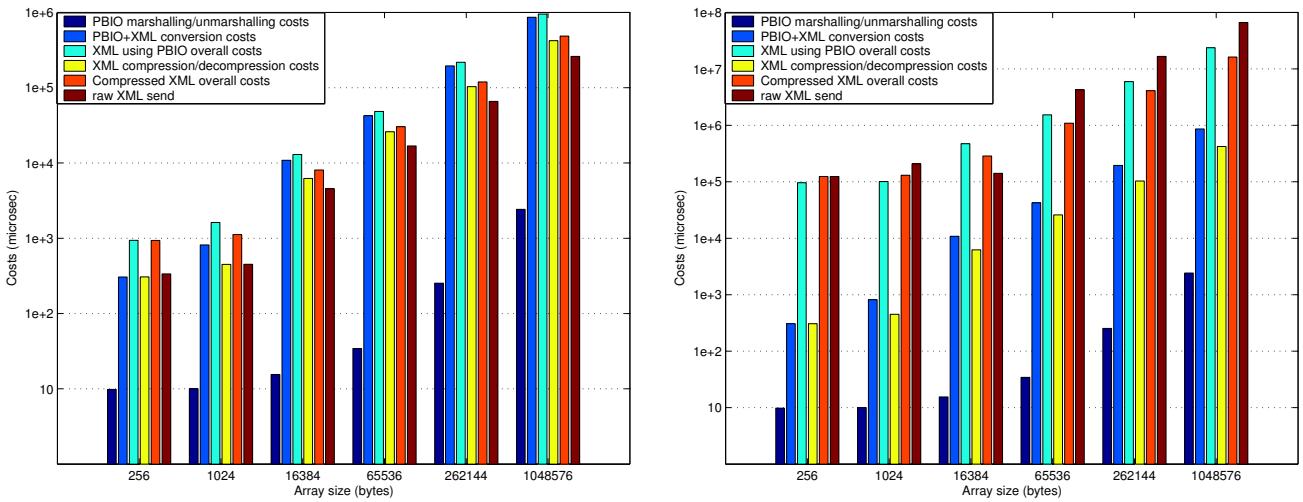

Fig. 5. Comparison of SOAP-bin costs with XML compression and direct send for an array over (a) 100Mbps and (b) ADSL links.

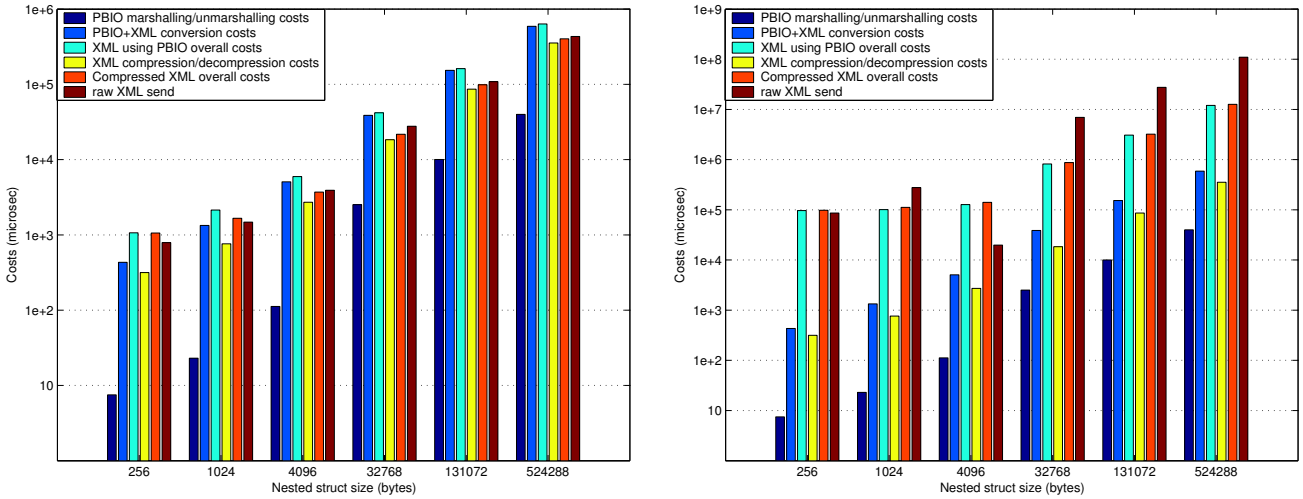

Fig. 6. Comparison of SOAP-bin costs with XML compression and direct send for a nested struct over (a) 100Mbps and (b) ADSL links.

f) SOAP-bin: Costs with XML data.: A second set of experiments conducted with the same setup assumes that data is available to the transport in XML rather than in native format. A typical scenario is the use of SOAP by an end user application interacting with a database, for instance. In such cases, additional work is needed to perform data conversion from and to XML. Experiments demonstrate that the time spent on conversion (especially on fast machines) is slightly offset by the time saved due to reductions in communication costs, due to the use of binary vs. text messages.

Experimental results in Figure 6 depict the times taken for conversion from XML to PBIO (and then back to XML), the overall time taken for such conversion plus the transport time, and the time taken to send XML messages (i.e., without conversion). The advantages of using binary data encodings for SOAP parameter transmission are less evident in this case. This is principally due to the need to explicitly parse XML. In addition, we have not optimized the XML-PBIO conversion handlers. In the case of the 100Mbps link, for instance, data conversion takes more time than simply sending raw XML, for both the array data and the nested structs, the latter being more pronounced (as can be expected). In contrast, with the ADSL link with its peak bandwidth of about 1Mbps, XML-PBIO conversion has clear advantages, for both data sets, since this conversion is equivalent to compressing XML to about 1/4 of its original size. However, it is even more advantageous to compress XML using some standard compression methods, as evident from the fact that this method is the fastest for all cases. Note, however, that compression is not a suitable choice if the application at either end (client or server) produces or consumes data in binary form (as with typical large-scale web servers that use backend machines). In addition, if data must be transformed or otherwise operated on, XML-based data representations again require parsing, leading to substantial computational costs.

g) Comparison among the Modes of Operation.: Based on these measurements, it is possible to characterize the performance of each of the three modes of SOAP-bin operation discussed in Section I.

Figure 7 shows the overall costs incurred from the three modes of operation. It is evident that, for high bandwidth links, the differences in performance increase as higher size data are involved, whereas the costs over low bandwidth links are similar. This is because of the large delay introduced by slow links, which overshadows any smaller delays due to XML conversion at either end. This leads us to conclude that the high performance mode of operation should be used in "internal" communications between back-end servers, whereas the interoperable or compatible modes of operation should be used if the data is required to be in, or available as XML format.

C. SOAP-binQ: Sample Applications

We present three applications - a scientific application, an image application, and a commerical application , and describe a fourth, a visualization application, that are adapted to use SOAP-binQ.

h) Continuous Quality Management.: In all experiments, RTT is measured during each request using the same method proposed in [41] for RTT estimation, with future work planning to use more complex and effective estimators like those described in [42]. Specifically, in the current implementation, a client sends a timestamp to the server along with the message, and the server sends back the same timestamp along with the reply. The client then computes the difference to determine the RTT for that request. This RTT value is used to update the client's measure of the cumulative RTT value through exponential averaging, using $R = \alpha \cdot R + (1-\alpha) \cdot M$, where R is the current estimate of RTT, M the new estimate and α a value between 0 and 1. Most estimators use a value of 0.875. Note that this RTT value calculation also includes the time spent by the server to prepare the data. This can be rectified by the server setting the timestamp back by the time taken to prepare its response data.

The information given in the quality file is used by both the client and the server just before sending the message. Based on the estimated RTT value, the corresponding interval in the policy is selected and the appropriate message type is chosen for transmission. The current message is then copied onto the chosen type through a single copy and then sent for marshalling. Every time the RTT is estimated by the client, the server is informed of the new value during the next request. Note that this approach may cause SOAP-binQ to oscillate between two message types. This can happen when the larger message causes some undue delay, which forces the transport to choose a smaller message. This may in turn decrease the RTT, thus causing it to alternate between the two sizes. A simply history-based mechanism of RTT estimation is used to prevent this.

1) Image Application: We have created a real-time imaging code similar in structure to the Skyserver [2] application developed by researchers from Johns Hopkins University and Microsoft. The application consists of a collection of servers, each of them possessing a set of images collected by remote telescopes. One of these servers acts as the primary server, in that it is directly accessible by a web user. The client requests a specific image, along with a transformation that must be applied to it. Requests for images are directed at the web server, which then reroutes the request to an appropriate server among the collection of image servers.

The image application used in our experiments emulates this behavior, in that remote clients request images and transformations on these images from an image server. Transformations include routines like scaling, edge detection, etc. The image server receiving a request responds with the appropriate image, modified based on the quality file.

For the image server application, due to its wide area nature, we evaluate its behavior in response to changes in network conditions. The server and client are run on two PCs connected by a single-hop 100Mbps Ethernet link, each with a 2.2GHz Pentium IV processor and 512MB RAM, running Linux 2.4.18-27.7. The application starts with the client sending a request to the server for an image, identified by its filename, and an operation to be performed on it. In this case, it is edge detection on PPM images (Skyserver

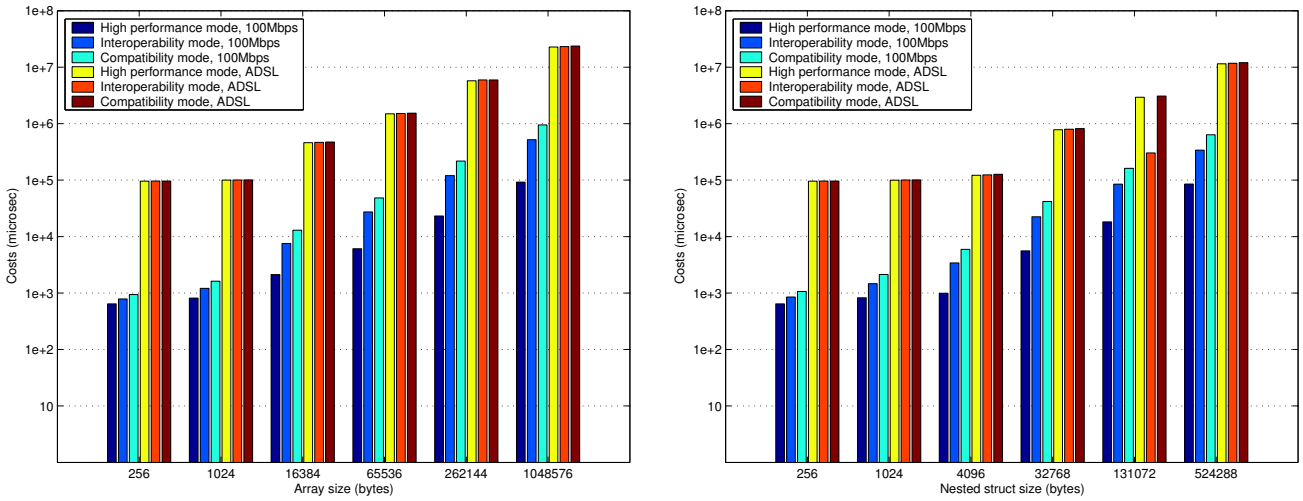

Fig. 7. Comparison of overall costs in high performance, interoperable and compatibility modes over 100Mbps and ADSL links for (a) arrays and (b) nested structs.

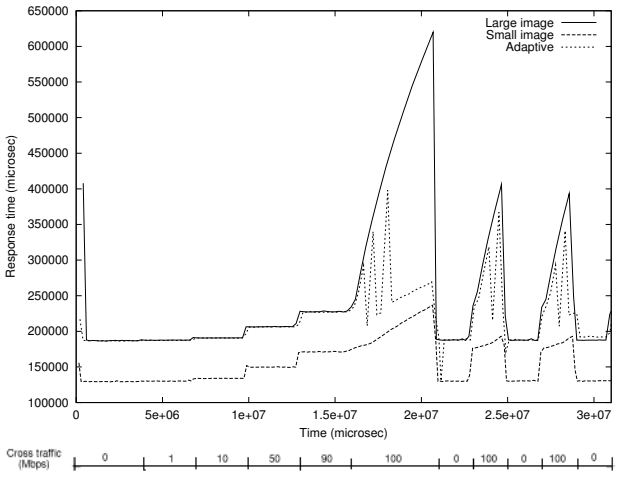

Fig. 8. Response times for Imaging Application.

uses JPEG images, for which we are currently implementing suitable data transformations). The PPM images are 640x480 pixels in resolution, with 3 bytes per pixel, one for each color value. Hence, the ideal response is close to 1MB in size. This results in high response times, especially for slow network connections.

For this application, the quality file is written to allow the server to resize the output image to 320x240 resolution when response times are high. In practice, more than two image sizes and/or additional ways of reducing response data would be used, but these experiments use only two image sizes, since the intent is simply to demonstrate the utility of this functionality offered by SOAP-binQ. To emulate network variations, cross-traffic is introduced using the IPerf tool *iperf*, which sends UDP packets at varying speeds. The resulting response times are shown in Figure 8.

As evident from the measurements in Figure 8, runtime quality management enables the application to send higher resolution images in good conditions, but once the response time increases further than that specified in the policy, it changes to sending lower resolution images. When conditions improve, it reverts to the original image sizes. As a result, the adaptative method's performance lies 'between' the performance attained for large vs. small image files.

One point to be noted here is that higher response times need not be caused by network congestion alone. They may also be due to the data-dependent nature of application behavior, i.e., the application itself may cause the delays if it spends extra time in preparing the data. This may cause SOAP-binQ to respond inappropriately. There are multiple ways in which issues like these may be addressed. As shown in our work on dynamic system monitoring[43], dynamic feedback from network protocols and/or about other system resources can more precisely identify the causes of performance degradation. In addition, to prevent frequent oscillation and as explained in Section IV-C, the use of history-based mechanisms can further help offset such effects.

2) Scientific Application: Collaborative scientific applications can generate and consume substantial amounts of data [9], [10]. The example used in this paper is from the domain of molecular dynamics, where the application models the behavior of the bonds between atoms within a molecule over time. It consists of a "bond server" that constructs a graph, where the vertices represent the atoms and the edges represent bonds. This data is available for a sequence of timesteps. Such a graph is constructed for every timestep and sent to a remote client for processing/display. The size corresponding to each of the timesteps for the response data is about 4KB. The SOAP-binQ quality file is formulated such that the server sends collective data corresponding to as many timestamps (between 1 and 4) in its response, as indicated by available network resources[2].

Experiments with this application are performed on the same machines as the previous application, but across an ADSL link, with UDP cross-traffic introduced during the course of the experiment. The intent is to emulate a situation

[2]An alternative approach is to eliminate unnecessary application data using user-defined data filters[43].

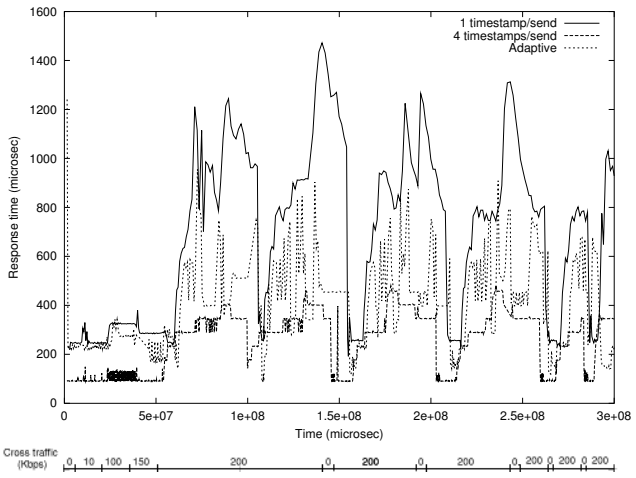

Fig. 9. Response times for Molecular Dynamics Application.

in which a science end user interacts from her home with a simulation running on a server farm.

Figure 9 shows the response times for various policies, varying with time. In one case, the server sends four timesteps per request, immaterial of the network conditions, and in the other, it sends one timestep per request. In the third case, an adaptive policy, which sends 1-4 timesteps for every request, depending on the cross-traffic, is used. Again, the changes in latency, in response to variations in network conditions, are minimized. The quality file used for the experiment guarantees that the response time never exceeds 900 μsec, and at the same time, it does not allow the network to be under-utilized - the response time seldom goes below 200 μsec. This can be made more vs. less restrictive dynamically, as mentioned in Section III-B.0.d.

3) Commercial Application: A third application emulates the operational information systems used by airlines, transport companies (e.g, FedEx), and others. In this application, information is continuously produced, entered in a large, memory-resident data set, business rules are applied to it, and resultant data is shared with end users. In the specific scenario used here, flight and passenger information is collected and distributed, and excerpts of such information are shared with relevant parties, such as flight caterers. The client, in that case, requests specific detail about the meals to be served, and the server responds with such detail.

Experiments assume an operational information system that uses PBIO in its core communications, but must convert catering information to XML for interoperability purposes. Measurements compare the transport of XML vs. PBIO data to end users, using the ADSL link. As seen in Table I, the smaller data sizes used by PBIO result in improvements compared to the direct use of XML, requiring a PBIO 'plug-in' at the client side. Improvements would be more substantial if conversion to XML was not performed at all, directly transmitting selected PBIO data from the core system infrastructure to plug-in capable end users. This in fact, is the way in which PBIO-based communications are used by one of our collaborators, Delta Technologies [3].

TABLE I
EVENT RATES FOR AIRLINE APPLICATION

	Size	Event rate (events per sec)
SOAP	3898 bytes	10.15
SOAP-bin	860 bytes	13.76
Native PBIO	860 bytes	14.06
SOAP (compressed XML)	1264 bytes	13.17

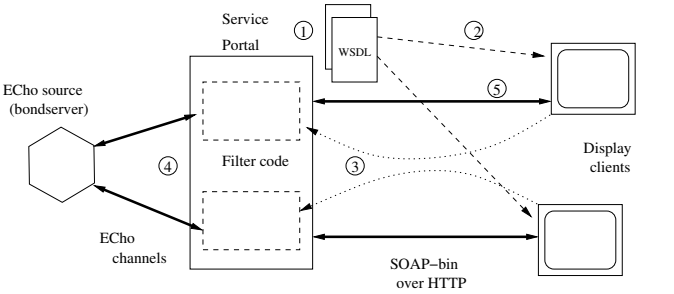

Fig. 10. Remote Visualization Application - Architecture.

4) Experiments with Remote Visualization: One of the key claims of our work is that the use of binary vs. XML data with SOAP substantially broadens the utility of the SOAP standard. As a concrete demonstration, we have developed a remote visualization application with SOAP-bin middleware. The application is based on the one discussed in Section IV-C.2, where the positions of the atoms and bonds in a molecule are obtained by the client, which are then displayed. The database on the server's end stores data in the raw format, and the display expects data in SVG format [44], which is just an XML document. The overall architecture of this framework is as shown in Figure 10.

The display client is connected to the service portal through a HTTP connection. The service portal acts as a sink for the 'ECho'[3] event source that generates bond data. The portal thus has an 'ECho' bondserver as a backend. The service portal (1) advertises its services through a set of WSDL files. These are obtained by the display clients (2), which then construct the appropriate request (3), with filter code and the desired output format. Data arriving from the bondserver (4) is then modified by the filter code, providing the output in the desired format, which is then sent back to the client (5) as the response. The client can dynamically change the filter code and the output format desired. This has been successfully implemented, thus providing more flexibility to the client in obtaining the data it needs. Measurements over two Linux machines similar to the one used in microbenchmarks, connected by a 100Mbps link shows a response time of about $2400\mu s$ for a data size of 16Kbytes, indicating a response time low enough for visualization purposes.

V. CONCLUSIONS AND FUTURE WORK

This paper describes the a software architecture for an efficient realization of the SOAP protocol. The goal is to

[3]ECho is our own implementation of a publish/subscribe, event-based communication system, focused on large-data applications[45].

make SOAP more broadly useful for end user applications, particularly targeting large-data applications [3], [9]. The key idea is to use SOAP's XML-based parameter data as meta-information, while actual data exchanges between clients and servers utilize binary representations of such data. The SOAP-bin protocol presented in this paper has substantial performance advantages compared to SOAP, due to the reduced data sizes for binary vs. XML data and due to reductions in processing overheads for these two alternative data representations.

A generalization of SOAP-bin, termed SOAP-binQ, further extends this protocol by associating runtime quality management functions with SOAP parameters. The idea is to use application-specific data manipulations, such as data down-sampling, to adjust data volumes to available clients. The adaptive behavior implemented by SOAP-binQ further expands the range of applications that can operate with the SOAP protocol. Continuous quality management is particularly important in resource-constrained environments, like international Internet connections, for instance.

The implementation of SOAP-bin and of SOAP-binQ presented in this paper uses a C-based realization of SOAP. It also uses an efficient binary structured data format suitable for communications across heterogeneous machines developed in our previous research (i.e., PBIO [14]). These choices affect the specific performance results presented in this paper, but similar results would be attained with alternative implementation approaches, including with Java-based implementations. In fact, in other work, we have already implemented XML-binary-Java translators, which permit us to efficiently transform structured XML data (i.e., schemas) to corresponding Java classes and vice versa [13].

While use of a binary format in place of a text standard may be questioned, it must be noted that this approach is not very different from various content types of binary nature (like JPEG, GIF, etc.) being used in conjunction with text-based HTML. With the growing trend seen in web services, it would only be prudent to use binary messages along with XML.

Currently, Soap-binQ quality handlers manipulate only binary data. In future work, we will generalize handlers to be able to manipulate XML data, binary data, or both. In addition, our current implementation installs handlers statically, at compile-time. In other work [45], we have already developed the technologies necessary to install binary handlers at runtime, using dynamic binary code generation techniques and/or using code repositories.

Our immediate future work will realize some of the improvements of SOAP-bin and SOAP-binQ described above, most importantly focusing on the ability to dynamically define and re-define quality management. We are also pursuing the use of SOAP-binQ with more complex end user applications, the near-term goal being its use of interactive remote visualization in scientitic codes like the SmartPointer application described in [5]. The intent is to leverage SOAP's promise of interoperability for high end codes.

REFERENCES

[1] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, "Simple object access protocol SOAP 1.1," http://www.w3.org/TR/SOAP/.

[2] A. Szalay, J. Gray, A. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg, "The SDSS skyserver - public access to the sloan digital sky server data," http://skyserver.sdss.org/en/sdss/skyserver/.

[3] Van Oleson, Karsten Schwan, Greg Eisenhauer, Beth Plale, and Calton Pu, "Operational information systems: An example from the airline industry," in *First Workshop on Industrial Experiences with Systems Software*, October 2000.

[4] Ada Gavrilovska, Karsten Schwan, and Van Oleson, "A practical approach for 'zero' downtime in an operational information system," in *International Conference on Distributed Computing Systems (ICDCS-2002)*, July 2002.

[5] Matthew Wolf, Zhongtang Cai, Weiyun Huang, and Karsten Schwan, "Smartpointers: Personalized scientific data portals in your hand," in *Proceedings of SuperComputing 2002*, November 2002.

[6] Greg Eisenhauer, Fabian Bustamente, and Karsten Schwan, "A middleware toolkit for client-initiated service specialization," in *Proceedings of the PODC Middleware Symposium*, July 2000.

[7] "Network for earthquake engineering simulation," http://www.nees.org.

[8] Yuan Chen, Karsten Schwan, and David Rosen, "Java mirrors: Building blocks for remote interaction," in *2002 International Parallel Distributed Processing Symposium(IPDPS 2002)*, April 2002.

[9] Beth Plale, Volker Elling, Greg Eisenhauer, Karsten Schwan, Davis King, and Vernard Martin, "Realizing distributed computational laboratories," *The International Journal of Parallel and Distributed Systems and Networks*, vol. 2, no. 3, 1999.

[10] Peter A. Dinda and David R. O'Hallaron, "An extensible toolkit for resource prediction in distributed systems," Tech. Rep., School of Computer Science, Carnegie Mellon University, 1999.

[11] Ahmed Gheith and Karsten Schwan, "Kernel support for multiweight objects, invocations, and atomicity in real-time multiprocessor applications," *ACM Transactions on Computer Systems (TOCS)*, vol. 11, no. 1, pp. 33–72, February 1993.

[12] Graham Hamilton, Michael L. Powell, and James J. Mitchell, "Subcontract: A flexible base for distributed programming," in *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, 1993.

[13] Patrick Widener, Greg Eisenhauer, Karsten Schwan, and Fabian E. Bustamante, "Open metadata formats: Efficient XML-based communication for high-performance computing," *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, , no. 5, pp. 315–324, 2002.

[14] Greg Eisenhauer, Fabian E. Bustamante, and Karsten Schwan, "Native data representation: An efficient wire format for high performance computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 12, Dec. 2002.

[15] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier, "Cluster-based scalable network services," in *Proceedings of the 16th ACM Symposium on Operating System Principles*, October 1997.

[16] Sun Microsystems Inc., "RPC: Remote procedure call protocol specification version 2," RFC-1057.

[17] J. Clark, "Expat - XML parser toolkit," http://www.jclark.com/xml/expat.html.

[18] Carsten Isert and Karsten Schwan, "Acds: Adapting computational data streams for high performance," in *2000 International Parallel Distributed Processing Symposium (IPDPS 2000)*, 2000.

[19] Qi He and Karsten Schwan, "IQ-RUDP: Coordinating application adaptation with network transport," in *Proceedings of the 11th International Symposium on High Performance Distributed Computing*, 2002.

[20] Fabián E. Bustamante, Greg Eisenhauer, Patrick Widener, Karsten Schwan, and Calton Pu, "Active streams: An approach to adaptive distributed systems," in *Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[21] Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley, "Investigating the limits of SOAP performance for scientific computing," in *Proceedings of The Eleventh International Symposium on High Performance Distributed Computing*, 2002.

[22] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon, "Requirements for and evaluation of RMI protocols for scientific computing," in *Proceedings of Supercomputing Conference*, 2000.

[23] S. Shirasuna, S. Matsuoka, H. Nakada, and S. Sekiguchi, "Evaluating web services based implementations of GridRPC," in *Proceedings of the 11th International Symposium on High Performance Distributed Computing (HPDC 11)*, 2002.

[24] V. Krishnaswamy, D. Walther, S. Bhola, E. Bommaiah, G. Riley, B. Topol, and M. Ahamad, "Efficient implementations of java RMI," in *4th USENIX Conference on Object-Oriented Technologies and Systems*, 1998.
[25] Jason Maassen, Thilo Kielmann, and Henri E. Bal, "Efficient replicated method invocation in Java," in *Java Grande*, 2000, pp. 88–96.
[26] J. Maassen, R. Nieuwpoort, R. Veldema, H. E. Bal, T. Kielmann, C. J. H. Jacobs, and R. F. H. Hofman, "Efficient Java RMI for parallel programming," *Programming Languages and Systems*, vol. 23, no. 6, pp. 747–775, 2001.
[27] Christian Nester, Michael Philippsen, and Bernhard Haumacher, "A more efficient RMI for java," in *Java Grande*, 1999, pp. 152–159.
[28] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley, "Design of an XML based interoperable RMI system : SoapRMI C++/Java 1.1," in *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2001.
[29] Aleksander Slominski, "Home page of XML pull parser," http://www.extreme.indiana.edu/xgws/xsoap/xpp/.
[30] John Ibbotson, "SOAP version 1.2 usage scenarios (w3c working draft 26 june 2002)," http://www.w3.org/TR/xmlp-scenarios/.
[31] Y. Krishnamurthy, V. Kachroo, D. A. Karr, C. Rodrigues, J. P. Loyall, R. Schantz, and Schmidt DC, "Integration of QoS-enabled distributed object computing middleware for developing next-generation distributed applications," in *ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, 2001.
[32] Ahmed Gheith and Karsten Schwan, "Chaos-arc – kernel support for atomic transactions in real-time applications," *ACM Transactions on Computer Systems*, Apr. 1993.
[33] K.H. Kim, "Object structures for real-time systems and simulators," *IEEE Computer*, August 1997.
[34] R. Koster, A. Black, J. Huang, J. Walpole, and C. Pu, "Infopipes for composing distributed information flows," in *Proceedings of the ACM Multimedia Doctoral Symposium*, October 2001.
[35] "Quality objects - QuO," http://quo.bbn.com/.
[36] Sun Microsystems Inc., "Java remote method invocation RMI," http://java.sun.com/products/jdk/rmi/.
[37] Alex Graveley, "Making SOAP with SOUP," http://lwn.net/2001/features/OLS/pdf/pdf/soup.pdf.
[38] Christian Poellabauer, Karsten Schwan, and Richard West, "Lightweight kernel/user communications for real-time and multimedia applications," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2001.
[39] Daniela Rosu, Karsten Schwan, Sudhakar Yalamanchili, and Rakesh Jha, "On adaptive resource allocation for complex real-time application," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, December 1997.
[40] Fabian Bustamante, Greg Eisenhauer, Karsten Schwan, and Patrick Widener, "Efficient wire formats for high performance computing," in *Proc. of Supercomputing 2000 (SC 2000)*, November 2000.
[41] Arpanet working group requests for comment, "Transmission control protocol specification," RFC-793.
[42] Van Jacobson and Michael J. Karels, "Congestion avoidance and control," in *ACM Sigcomm '88 Symposium*, 1988.
[43] S. Agarwala, C. Poellabauer, J. Kong, K. Schwan, and M. Wolf, "Resource-aware stream management with the customizable dproc distributed monitoring mechanisms," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.
[44] John Bowler et al., "Scalable vector graphics SVG 1.0 specification," http://www.w3.org/TR/SVG/.
[45] Greg Eisenhauer, "The ECho event delivery system," http://www.cc.gatech.edu/systems/projects/ECho/Eisenhauer02EED.pdf.