Pushing the SOAP Envelope With Web Services for Scientific Computing

Robert A. van Engelen*
Department of Computer Science and
School of Computational Science and Information Technology
Florida State University, Tallahassee, FL 32306-4530
engelen@cs.fsu.edu

Abstract

This paper investigates the usability, interoperability, and performance issues of SOAP/XML-based Web and Grid Services for scientific computing. Several key issues are addressed that are important for the deployment of high-performance and mission-critical SOAP/XML-based services. A successful deployment can be achieved by limiting the overhead of XML encoding through exploiting XML schema extensibility to define optimized XML data representations and by reducing message passing latencies through message chunking, compression, routing, and streaming.

1. Introduction

The Web Services framework [14] is gaining momentum as an approach to distributed computing by supporting the creation, deployment, and dynamic discovery of distributed applications. The popularity of Web Services can be contributed to the inevitable transition from people to software applications to access public, commercial, and government services on the Internet. This Web Services evolution is made possible in part by the adaptation of universally accepted standard protocols such as HTTP (Hyper Text Transfer Protocol) [13], XML (Extensible Markup Language) [23], SOAP (Simple Object Access Protocol) [21], WSDL (Web Services Description Language) [22], WSFL (Web Services Flow Language) [10] and UDDI (Universal Description, Discovery and Integration) [17].

The state-of-the-art Grid computing tools utilize the power of resource sharing with distributed applications on the Internet [6]. Grid applications enable researchers to solve large-scale problems in science and engineering. The Grid community recently embraced Web Services as a key technology supporting the shift to service-oriented Grid applications. A Grid Service is envisioned as "a (potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization" [7]. The Open Grid Services Architecture (OGSA) [8] leverages Web Services protocols and additional interfaces to manage Grid service lifetime, policies and credentials, and to provide support for notification as mandated by the OGSA specification.

This paper addresses the usability, interoperability, and performance aspects of Web and Grid Services for scientific computing. Usability is largely motivated by the fact that the Web Services framework provides universal access to numerous public and commercial services on the Internet. This creates new opportunities for Grid applications. Another important aspect of the framework is the high level of interoperability and service compositionality across platforms and programming languages. In addition to these usability and interoperability aspects, the framework also offers a practical infrastructure for Internet computing that includes protocols for enhanced routing capabilities such as multi-hop routing with intermediaries and the streaming of Direct Internet Message Encapsulation (DIME) attachments. Also available are Web Service security extensions such as HTTPS, WS-Security [12], SOAP digital signature [2], and GSI [3]. These enhancements are important to propel Web Services in performance-demanding environments, such as computational Grids.

It is widely perceived that Web and Grid Services can suffer severe performance penalties by adopting the XML-based SOAP protocol for transporting large volumes of data. Several papers have reported on the performance of scientific computing with Web Services, such as a performance investigation of SOAP for scientific applications that utilize large floating point arrays [4], a comparison of the latency of SOAP Web Services implementations [5], an evaluation of Web Services based implementations for

* Supported in part by NSF grants CCR-9904943, CCR-0105422, CCR-0208892, and DOE grant DEFG02-02ER25543.

GridRPC [16], and the design and implementation of the *gSOAP* toolkit for efficient SOAP/XML Web Services [18, 20]. However, these studies did not address the key issues that can affect performance the most. The key issues that may prevent a successful deployment are the overhead of XML encodings with SOAP/XML and the message passing latencies of HTTP over TCP/IP. These issues are addressed in this paper by exploiting XML schema extensibility for defining optimized XML representations for numerical data and by message passing optimizations such as chunking, compression, routing, and streaming media techniques.

This paper attempts to answer the following questions to help identify several key issues for deploying high-performance and mission-critical SOAP/XML-based services:

- How does SOAP/XML as a generic data representation compare to more specific XML representations for scientific data such as MathML and XSIL, for example?
- What is the performance impact of SOAP/XML Web Services compared to Java RMI?
- Can performance be improved using HTTP chunking and/or compression of SOAP/XML messages?
- What is the speedup of streaming SOAP/XML with DIME?
- Which alternative XML representations for numerical data are available and what is the impact on communication latency and performance?

We are currently developing a new software package called *gNTL* (*gSOAP* Numerical Task Library) for efficient scientific computing with Web Services in C, C++, and Fortran (using a Fortran-to-C bridge). The design recommendations of the *gNTL* package are largely based on the results of the study presented in this paper. The package includes a set of XML schemas that provide alternative views of numerical data in XML, including sparse vectors, matrices, and arrays. The XML schemas provide SOAP/XML interoperability with other SOAP implementations such as Apache Axis (for Java), for example.

The remainder of this paper is organized as follows. Section 2 introduces Web and Grid Services toolkits. Section 3 discusses SOAP/XML for scientific computing. Section 4 compares performance results of HTTP chunking, compression, DIME streaming, and XML representations of numerical data. Section 5 discusses the observations and results. Finally, Section 6 concludes with a summary.

2. Web and Grid Services Toolkits

Currently more than seventy SOAP Web Services toolkits are available for a variety of programming languages and platforms, see e.g. [15]. Because production-quality scientific applications are written in Fortran[1], C, or C++, we will only mention the C/C++ SOAP toolkits. These SOAP Web Services toolkits and libraries for C++ are *Borland builder 6 with Delphi*, *easySOAP*, *eSOAP*, *gSOAP*, the *.NET framework*, *WASP for C++*, and *SQLData*.

The *gSOAP* toolkit is a platform-independent development environment for C and C++ Web Services [18]. Ease of use and performance were important design considerations in the development of the toolkit. In fact, the toolkit offers an easy-to-use RPC compiler that produces the stub and skeleton routines to integrate (existing) C and C++ applications into SOAP/XML Web Services. A unique aspect of the toolkit is that it automatically maps native C/C++ application data types to semantically equivalent XML types and vice versa. This enables direct SOAP/XML messaging with C/C++ applications on the Web. The overhead and memory usage of the run-time mapping to XML with *gSOAP*'s schema-optimized XML parsing techniques is low, which makes *gSOAP* attractive in high-performance environments and embedded systems.

The Globus Toolkit v3 [8] includes the *gSOAP* toolkit. *gSOAP* is used to implement C/C++ bindings for the OGSA-compliant Grid Services. The *gSOAP* toolkit is also used in the GridLab project [9] with the Globus Toolkit v2 and the GSI plug-in for *gSOAP* [3] and in the Harness [1] project for distributed computing. Many companies have shown an interest in *gSOAP*. For example, the *gSOAP* toolkit is integrated in the IBM alphaWorks Web Services Tool Kit for Mobile Devices [11].

3. Scientific Computing With SOAP/XML

SOAP is a protocol for remote procedure calling and messaging with XML-encoded application data. However, SOAP does not require the use of XML per se. In fact, SOAP supports binary data attachments and remotely referenced data such as objects provided by third parties that are produced or consumed at separate hosts, for example. These features can be exploited to create data portals on the Web. SOAP also specifies various usage scenarios, such as one-way message passing, single and multiple request-response invocations, and routing.

Consider for example the data transfer illustrated in Figure 1. Data is efficiently transfered from a client to a server with *gSOAP*'s streaming SOAP/XML and DIME. This form of routing streams the (binary) data encapsulated in DIME from a data source at the client side into a data repository at the server side. This data can be retrieved and stored on disk or processed dynamically by tasks running on separate machines, for example. Note that SOAP/XML with stream-

1 Fortran codes can be linked with C codes to provide SOAP interfaces.

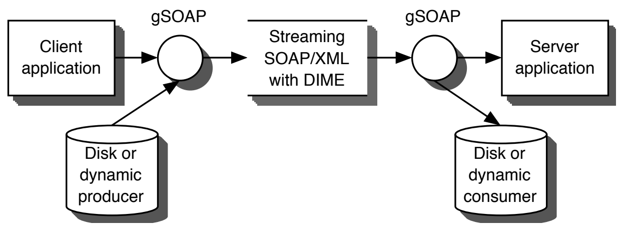


Figure 1. Example Client-Server Messaging With Streaming SOAP/XML and DIME

ing DIME file transfers are more flexible than current file transfer protocols such as FTP and GridFTP, because the SOAP/XML messages can be utilized to carry meta-data with the file transfer requests and responses. In addition, Web Services extensions, such as routing and security for example, are orthogonal which means that file transfers can be routed and protected.

SOAP and WSDL are a protocols that exploit the richness of the XML schema specification standard for defining XML content. XML schemas define the structure of XML documents by defining document layout and content constraints. The schema notation is amenable to meta-level data translation and interpretation. In fact, SOAP toolkits already translate WSDL service descriptions into service objects and client proxies by inspecting the WSDL schema types parts to determine the parameter marshaling requirements. The SOAP RPC and (to a limited extend) the SOAP literal encoding styles provide a rich set of data types that can be utilized for XML encoding, including primitive types (XSD types), enumerations, bitmasks, arrays, lists, and records (structs). Arbitrary data types with application-specific semantics can be added by XML schema extension and restriction. Schema extension and restriction introduces an object-oriented approach to data modeling with XML.

In contrast, many XML protocols for scientific data are based on DTDs rather than XML schemas and lack XML namespace and extensibility properties. The expressiveness of these protocols is restricted to a set of pre-defined data types offered by the protocol. For example, XSIL (Extensible Scientific Interchange Language) [25] and XDMF (Extensible Data Model and Format) [24] both provide encodings for primitive and aggregate types in XML. XSIL concentrates on data structures with a Java-dependent XML format. The XSIL, XDMF, MathML, and BioML XML representations can be seamlessly blended into SOAP/XML. However, the application-specific XSIL and XDMF representations have a limited set of data types and offer no means to exploit schema extension to define optimized and application-specific XML layouts.

The XML schema standard provides a basis for programming language bindings for XML as demonstrated by *gSOAP* and the .NET framework, for example. *gSOAP* generates XML schemas at compile time and automatically performs data mappings at run time. The toolkit guarantees the preservation of the logical structure of graph-based data which is difficult to achieve with hand-written conversion routines (often referred to as *wrappers*). Scientific applications typically deal with large symbolic or numerical data sets. Symbolic data can be text-based or graph-based, such as symbolic expressions. Graph-based data can be represented in XML using SOAP RPC encoding of nodes as “structs” and co-referenced nodes as “multi-referenced objects” [19]. The use of XML graph nodes to representing symbolic expressions in XML can be based on existing XML standards, such as MathML and CML (Chemical Markup Language).

Flat data structures such as numerical vectors and matrices can be represented in many different forms of XML, including sparse SOAP arrays, UTF8-encoded strings, and hexadecimal and base64 binary encodings. Flat data structures don’t necessarily require a generic SOAP array representation with values represented in decimal notation. The loss of floating point precision and the encoding overhead of floating point numbers in decimal often prohibits the exchange of numerical data in plain text form.

Table 1 list the storage requirements of 32-bit, 64-bit, and 128-bit floating point values in XML. Hexadecimal notation requires a pair of octets to encode each octet of the internal binary floating point representation, UTF8 encoding treats the internal representation of floats as sequences of unicode characters represented in UTF8, and base64 requires 4 octets to encode each sequence of 3 octets. UTF8 encoding represents float point values as strings in XML, which means that the string contents have to be stored in XML CDATA sections to prevent interference with XML markup. In addition, most floating point values will be encoded with the full 24 UTF8 characters. As a result, much of the space gains of UTF8 encoding will be lost. Base64 is more attractive because of the low overhead of the encoding scheme.

To save space and preserve precision, a numerical vec-

FP Prec.	Decim.	Hex	UTF8	Base64
32 bit	1–14	8	1–6	8
64 bit	1–24	16	2–12	12
128 bit	1–40	32	4–24	24

Table 1. XML Storage Requirements (in Octets) of One Floating Point Value in Decimal Notation Compared to Hexadecimal, UTF8, and Base64 Binary Encodings.

```
class xsd__base64Binary // built-in base64Binary XSD type
{ unsigned char *__ptr; // pointer to block of memory
  int __size;           // size of memory block
};
class ns__int32: public xsd__base64Binary
{ bool normalized = true;   // initially true when recv'd
  void set_int32(int*,int); // set data and normalize
  int *get_int32();         // get de-normalized data
};
class ns__fp32: public xsd__base64Binary
{ bool normalized = true;   // initially true when recv'd
  void set_fp32(float*,int);// set data and normalize
  float *get_fp32();        // get de-normalized data
};
```

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.genivia.com/gntl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="http://www.types.org"
  xmlns="http://www.w3.org/2001/XMLSchema">
 <simpleType name="int32">
  <restriction base="xsd:base64Binary"/>
 </simpleType>
 <simpleType name="fp32">
  <restriction base="xsd:base64Binary"/>
 </simpleType>
</schema>
```

(b)

Figure 2. Base64 and Derived int32 and fp32 gSOAP Specifications (a) and XML Schema (b)

tor can be encoded with a SOAP array of base64-encoded IEEE 754 floating point values. However, space requirements can be further reduced by encoding the entire vector in base64. An appropriate XML schema can be defined for these base64-encoded arrays using the generic base64 binary XSD type as the base type. The *gSOAP* specification of the int and float arrays derived from the base64 XSD type is shown in Figure 2.

The specification exploits inheritance to produce the necessary schema extensions for the int and float array types. Getter and setter methods are added to access the binary representation and to (de-)normalize the data to big endian integers or IEEE 754 floats, for example, to ensure interoperability of the XML representations with applications running on different platforms. The run-time encoded base64 XML content carries the schema type with the data, so the recipient can determine the numerical array encoding used and apply de-normalization when necessary.

4. Latency and Performance

The choice of XML encoding for the data of a SOAP/XML Web Service application can have a significant impact on the communication latency and overall performance of the service. The XML representation of floating point arrays, for example, can be as large as ten times the original binary representation. However, the total message length is not always the determining factor of performance but the time to convert data in XML (such as floats to decimal notation and back) is, because it introduces a significant overhead [4]. This overhead is also apparent in the linear systems solver service that utilizes SOAP arrays of floats [18].

To determine the impact of XML encoding and HTTP transport on the performance of a service, we used the well-known magic squares algorithm to build a "MagicSquares" Web Service with *gSOAP* that produces 32-bit integer matrices. Part of the *gSOAP* specification of the MagicSquares service is shown in Figure 3. This algorithm is fast and linear in the size of the matrix. Because the algorithm is fast, it enables us to investigate the XML encoding/decoding and communication latencies rather than the compute latency of the service.

The service specification includes the definition of XML base64-encoded matrices in *gSOAP*. The definitions for SOAP-array matrices and binary DIME attachments that were used in the performance tests are similar (not shown). The *gSOAP* service method implementation uses the magic squares algorithm shown in Figure 4

We measured the impact of HTTP chunking, compression, base64 encoding, and DIME on the performance of the service. HTTP chunking provides a form of streaming. Chunking can be used to omit the extra pre-serialization phase that *gSOAP* uses to determine HTTP content length. We also measured the overhead of real-time HTTP compression to transfer SOAP arrays with *gSOAP*. The results are shown in Figure 5. The response times of the MagicSquares service with HTTP chunking and base64 encoding is very low compared to non-chunked and chunked+compressed SOAP array transfers for varying matrix ranks (rank=100 corresponds to an array of 100 arrays of 100 integers). The experiments were conducted with two 550 MHz dual P3 machines with Red Hat Linux and a 100BaseT LAN.

Real-time compression appears to be very expensive but also very effective because it reduces the message length to a size that is comparable to the size of the compressed form of the binary data (which is about 20KB for a 100 by 100 matrix with 32-bit integers). The gzip compression rate

```
struct matrix           // matrix is a base64 type
{ unsigned char *__ptr; // pointer to memory block
  int __size;           // size of block
};
int ns__magic(int n, struct matrix *q);
```

Figure 3. gSOAP MagicSquares Web Service Specification With Base64-Encoded Matrices

```
int ns__magic(struct soap *soap, int n, struct matrix *q)
{ int i, j, k, l, v;
  int *a = (int*)soap_malloc(soap, sizeof(int)*n*n);
  for (i=0; i<n*n; i++) a[i] = 0;
  i = 0; j = (n-1)/2; a[n*i+j] = 1;
  for (v=2; v<=n*n; v++)
  { if (i<1) k = n-1; else k = i-1;
    if (j<1) l = n-1; else l = j-1;
    if (a[n*k+l]) i = (i+1)%n; else { i = k; j = l; }
    a[n*i+j] = v;
  }
  q->__ptr = (unsigned char*)a;
  q->__size = sizeof(int)*n*n;
  return SOAP_OK;
}
```

Figure 4. gSOAP MagicSquares Web Service Method

of XML containing SOAP arrays is very high as is shown in Figure 6, where the compression $rate = (1 - \frac{c}{\ell})\%$, ℓ denotes the length of the original message, and c denotes the length of the compressed message. Compression is useful when the network bandwidth is limited. However, some experimentation with low-bandwidth networks such as a phone line did not yield any performance gains with compressed HTTP transfers because v.90 modems already apply compression.

Figure 7 **(a)** shows the response times of the MagicSquares server using chunked SOAP messages with base64 encoding and SOAP/XML with DIME over the LAN. The startup overhead indicates the network latency to establish a socket connection with the service. *gSOAP* supports

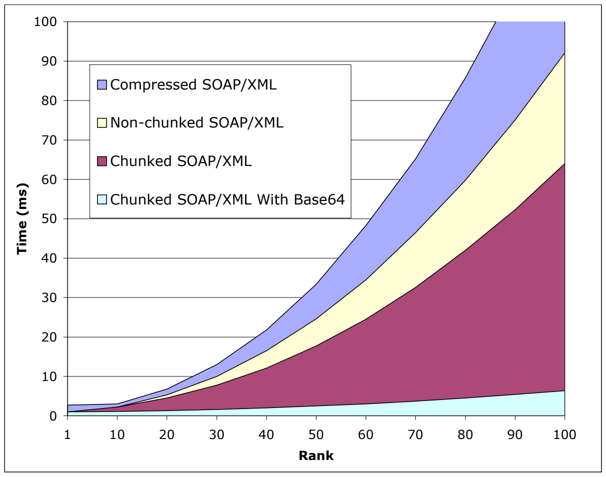


Figure 5. Response Times of the MagicSquares Service With Base64 and Chunked/Compressed SOAP Arrays for Varying Matrix Rank

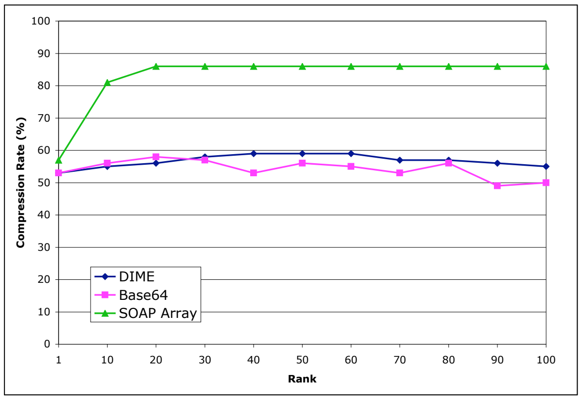


Figure 6. Compression Rates of SOAP Arrays, Base64, and DIME-Encoded Matrices Returned by the SOAP/XML MagicSquares Server for Varying Matrix Rank

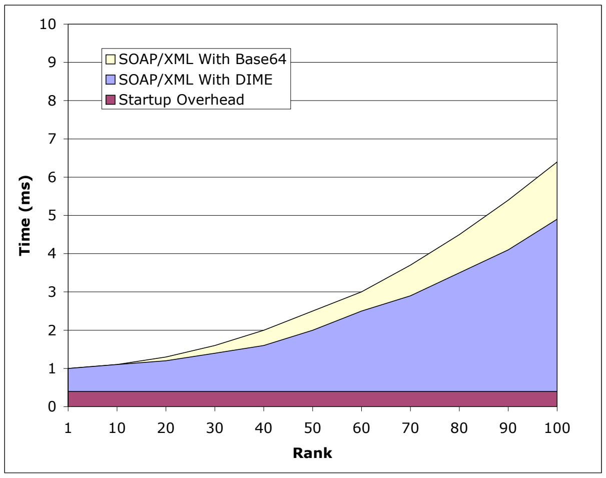
HTTP keep-alive, which means that the startup time can be avoided using HTTP keep-alive connections after establishing a connection with a service. In addition, *gSOAP* disables the Nagle algorithm to reduce the connection startup latency (`TCP_NODELAY`).

Figure 7 **(b)** shows the response times of a Java implementation of the MagicSquares server using Java RMI over the LAN (JDK 1.3 with JIT). The startup overhead is the time it takes for the Java RMI lookup operation to establish a connection to the server. The SOAP/XML service response is almost twice as fast with HTTP chunking and DIME compared to the Java RMI implementation.

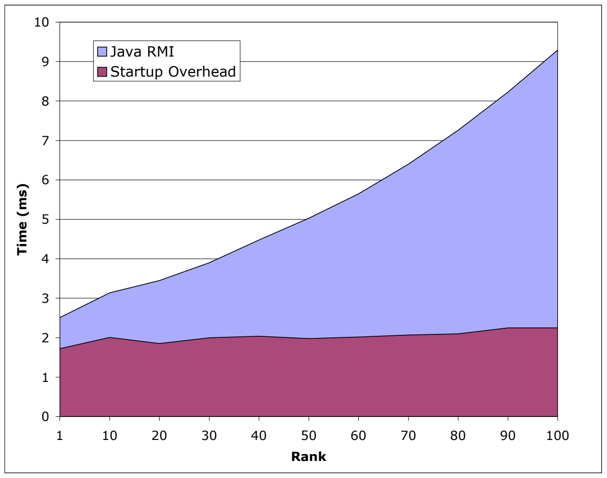
5. Discussion

This section briefly summarizes the results and discusses these findings in the context of scientific computing with Web and Grid services.

File transfers File transfer protocols such as FTP and GridFTP are commonly used to exchange large static data sets. However, SOAP/XML with streaming DIME is more flexible than these file transfer protocols. Firstly, the SOAP/XML request and response messages can be utilized to carry meta-data such as application-specific information items. Secondly, multiple files can be streamed at once. Thirdly, streaming DIME supports the exchange of dynamic data (producer and consumer type applications). Fi-



(a)



(b)

Figure 7. Response Times of MagicSquares Servers With Base64 and DIME-Encoded Matrices (a) and Java RMI (b) for Varying Matrix Rank

nally, the industry standard WS routing and WS security protocols are orthogonal to SOAP/XML/DIME which means that file transfers based on Web Services can be routed and protected.

Data encodings SOAP and WSDL are protocols that exploit the richness of the XML schema specification standard for defining XML content. The XML schema extension and restriction features introduces an object-oriented approach to data modeling with XML. It enables the definition of derived data types to optimize the exchange of application-specific data formats such as numerical arrays. The *gNTL* library for *gSOAP* (*gNTL* is currently under development) is based on this powerful concept and offers alternative XML views of application-specific data.

DIME attachments provide the most efficient means to exchange raw binary data. While binary data transfers are often best utilized for platform-independent streaming media types such as images and sound, binary numerical data exchange is platform dependent due to the differences between big- and little-endian architectures and internal floating point representations. The results presented in this paper demonstrated that the base64 encoding is almost as efficient as DIME transfers of data sets (550MHz P3, Linux RH, and 100BaseT LAN). Base64 has the advantage that individual data items (ints and floats) can be encoded within the SOAP/XML message contents such as SOAP arrays, for example.

HTTP keep-alive A client can request a persistent connection with a service using HTTP keep-alive. However, the service may drop the connection after a while (to enforce a fairness policy) thereby forcing the client to reconnect. The start-up time to establish a connection between a *gSOAP* client and service is relatively low and less than 0.5 ms (550MHz P3, Linux RH, and 100BaseT LAN). HTTP keep-alive is enabled in *gSOAP* with the command `soap_set_omode(soap, SOAP_IO_KEEPALIVE)`.

HTTP chunking Either a client or a service or both can utilize HTTP chunked transfers. HTTP chunking is a simple form of streaming, because the chunked SOAP/XML messages are not buffered to determine the HTTP content length header. In fact, in contrast to other SOAP implementations, *gSOAP* does not buffer a message to determine the HTTP content length header, but uses a two-phase serialization scheme consisting of a message length counting phase followed by the transmission of the message [18]. This scheme is implemented by serializing the message twice. HTTP chunking limits the message transmission to only one phase, which is faster. In addition, *gSOAP* implements a form of latency hiding with chunked transfers, where communication is overlapped with the serialization computations. HTTP chunking is enabled in *gSOAP* with the command `soap_set_omode(soap, SOAP_IO_CHUNK)`

HTTP compression Compression should be used sparingly. The overhead of the Zlib compression algorithm outweighs the benefit of the message size reduction to improve bandwidth utilization, for example. Most modems and some network routers already compress TCP/IP packets on the fly, which eliminates the need to compress SOAP/XML messages. However, compression is still useful to store data in compressed XML using *gSOAP*'s data serializers. HTTP gzip compression is enabled in *gSOAP* with the command `soap_set_omode(soap, SOAP_ENC_ZLIB)`

6. Conclusions

This paper discussed the use of SOAP/XML Web and Grid Services for scientific computing. SOAP and WSDL are based on extensible schemas which allows the selection of alternative XML layouts for application data depending on on interoperability and performance goals. Performance testing with HTTP chunking, compression, binary DIME attachments, and XML representations such as SOAP Arrays and base64 shows that significant performance gains with SOAP/XML Web Services are possible, surpassing the performance of Java RMI, for example.

References

[1] Beck et al. HARNESS: A next generation distributed virtual machine. *Future Generation Computer Systems*, 15, 1999.

[2] A. Brown et al. SOAP security extensions: Digital signature. Technical report, W3C, 2001. http://www.w3.org/TR/SOAP-dsig.

[3] M. Cafaro, D. Lezzi, and R. van Engelen. Secure web services with globus GSI and gSOAP. In *submitted to EUROPAR2003*, 2003. http://sara.unile.it/~cafaro/gsi-plugin.html.

[4] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of SOAP performance for scientific computing. In *proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing*, 2002.

[5] D. Davis and M. Parashar. Latency performance of SOAP implementations. In *2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002.

[6] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1998.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the Grid: An Open Grid Services Architecture for distributed system integration. Technical report, the Globus project, 2002. http://www.globus.org/research/papers/ogsa.pdf.

[9] GridLab. The gridlab project. http://www.gridlab.org.

[10] IBM. WSFL specification, 2001. http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[11] IBM alphaWorks. Web services tool kit for mobile devices, 2002. http://www.alphaworks.ibm.com/tech/wstkMD.

[12] IBM and Microsoft. WS-Security specification. Technical report, IBM and Microsoft, 2002. http://msdn.microsoft.com/ws-security.

[13] IETF. HTTP 1.1 specification rfc2616. www.ietf.org/rfc/rfc2616.txt.

[14] H. Kreger. Web services conceptual architecture WSCA 1.0. Technical report, IBM, 2001.

[15] P. Kulchenko. SOAP::Lite for Perl. http://www.soaplite.com.

[16] S. Shirasuna, H. Nakada, and S. Sekiguchi. Evaluating web services based implementations of GridRPC. In *IEEE International Symposium on High Performance Distributed Computing HPDC-11*, page 237, 2002.

[17] UDDI. The universal description, discovery, and integration (UDDI) specification. http://www.uddi.org/specification.html.

[18] R. van Engelen and K. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002.

[19] R. van Engelen, K. Gallivan, G. Gupta, and G. Cybenko. XML-RPC agents for distributed scientific computing. In *IMACS'2000 Conference*, Lausanne, Switzerland, 2000.

[20] R. van Engelen, G. Gupta, and S. Pant. Developing web services for C and C++. *IEEE Internet Computing*, pages 53–61, March 2003.

[21] W3C. SOAP specification. http://www.w3.org/TR/SOAP.

[22] W3C. WSDL specification. http://www.w3.org/TR/wsdl.

[23] W3C. XML specification. http://www.w3.org/XML/Core.

[24] XDMF. Extensible data model and format. http://www.arl.hpc.mil/ice/.

[25] XSIL. Extensible scientific interchange language. http://www.cacr.caltech.edu/SDA/xsil/.