Evaluating Web Services Based Implementations of GridRPC

Satoshi Shirasuna
Tokyo Institute of Technology
sirasuna@is.titech.ac.jp

Hidemoto Nakada
Tokyo Institute of Technology, and
National Institute of Advanced
Industrial Science and Technology
hide-nakada@aist.go.jp

Satoshi Matsuoka
Tokyo Institute of Technology, and
National Institute of Informatics
matsu@is.titech.ac.jp

Satoshi Sekiguchi
National Institute of Advanced
Industrial Science and Technology
s.sekiguchi@aist.go.jp

Abstract

GridRPC is a class of Grid middleware for scientific computing. Interoperability has been an important issue, because current GridRPC systems each employ its own protocol. Web services, where XML-based standards such as SOAP and WSDL are expected to see widespread use, could be the medium of interoperability; however, it is not clear if 1) XML-based schemas have sufficient expressive power for GridRPC, and 2) whether performance could be made sufficient. Our experiments indicate that the use of such technologies are more promising. than previously reported. Although a naive implementation of SOAP-based GridRPC has severe performance overhead, application of a series of optimizations improves performance. However, encoding of various features of GridRPC proved to be somewhat difficult due to WSDL limitations. The results show that GridRPC systems can be based on Web technologies, but there needs to be work to extend WSDL specifications, possibly impacting OGSA-based Grid services directions.

1. Introduction

GridRPC systems, such as Ninf[9] and NetSolve[3], provide high-level, easy-to-use task-parallel programming abstraction for the Grid. It hides the complexities in using various Grid services such as security, resource discovery, and scheduling; provides traditional parallel computing abstractions such as parallel tasking and flexible synchronization; as well as facilitate scientific computing features such as client-sever shared memory abstraction for large arrays, transparent client-side RPC API via dynamic server-side IDL management, and scientific IDL.

One primary R&D/standardization issues is interoperability amongst GridRPC systems, as well as with other services. Existing GridRPC systems employ private protocols, and although there have been efforts such as the NetSolve–Ninf bridge[8], they are not desirable solutions due to 1) only intersections of the features becoming available, 2) performance penalty of protocol translation, and 3) difficulty of supporting all combinations of GridRPC systems.

Meanwhile, Web services technologies with XML-based protocol standardization such as SOAP[2], WSDL[4] and UDDI[1], along with recent announcement and acceptance of the Open Grid Services Architecture (OGSA) make it an attractive medium for such standardization efforts. However it is not clear if 1) XML-based schemas have sufficient expressive power for GridRPC, as it embodies various mechanisms not present in traditional RPCs. For example, both NetSolve and Ninf "scientific IDL" allow automated shared memory, call-by-reference passing of array parameters while transferring only portions of arrays being used, and their server-side IDL management frees client users from dealing with managing IDLs to be up-to-date, etc. 2) Performance could be insufficient, since there are significant costs in XML handling. Previous results have been unoptimistic in this regard[5].

To investigate whether Web services are viable substrate for future GridRPC systems, we substituted various GridRPC components with SOAP and WSDL-based modules. Results are more promising than previously reported; although a naive implementation of SOAP-based GridRPC exhibits considerable performance overhead, series of optimization techniques improves performance to be competitive with the original binary transport. However, encoding various features of GridRPC as described above proved to

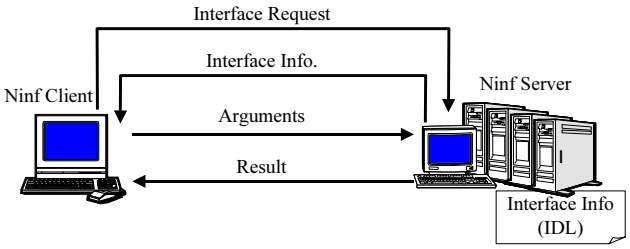

Figure 1. Ninf system architecture

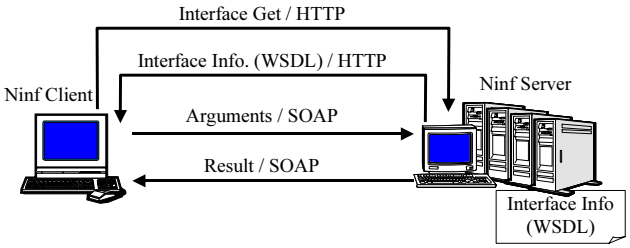

Figure 2. SOAP-based GridRPC system architecture

be imperfect, due to several inherent limitations of current WSDL, mandating small extensions to support scientific IDL. Overall GridRPC systems can benefit from Web services as the underlying transport and service descriptions, even in future OGSA-based Grid services.

2. Web Services and GridRPC

WWW and XML recently gave rise to "Web services", including the following technologies:

SOAP SOAP (Simple Object Access Protocol)[2] is a specification for message exchange in a distributed environment, independent of programming languages, or platforms. SOAP itself is a one-way, object-based wire transfer protocol. Although not mandated, HTTP is widely used for transport layer of SOAP messages.

WSDL WSDL (Web Service Definition Language)[4] is a specification for describing interface information for web services. WSDL itself has several bindings for Web services, but often used with SOAP as its IDL.

We investigate the usage of both standards in GridRPC system by replacing the various proprietary protocols and schemas components, while preserving the current client APIs and IDLs. Figure 1 shows the current architecture of the Ninf GridRPC system. A remote library is "gridified" by describing its interface and other information using the Ninf (scientific) IDL. Contrary to traditional RPCs where IDLs are managed on both the client and the server, and programmers must "program around" the generated skeletons from IDLs, in GridRPC all IDL management is on the server side, and the client can preserve the almost identical call interface without any skeleton generation, achieving distribution transparency. The underlying client library obtains interface information at run-time from the server in a two-phase RPC call sequence (similar to Java Jini). Here, a private "Ninf" protocol is employed to obtain interface information and perform wire transfer of GridRPC calls, latter based on XDR.

Figure 2 shows the Web-service based GridRPC architecture. The client retains the original API and the scientific IDL—however, underneath interface information is described using WSDL, translated from scientific IDL. The client performs a two-phase RPC call where it obtains the WSDL file using HTTP Get. SOAP is used to exchange parameters, meaning that call information such as library name and parameters themselves (including huge arrays) are encoded in a SOAP message. The result is again sent back in a SOAP response message.

Despite that the Web-service-based GridRPC system has various merits such as utilization of web services tools, interoperatbility and better compliance with OGSA, as well as being firewall friendly, some issues may hinder its usage:

Performance Degradation Performance penalty cased by XML could be significant. SOAP encoding may cause $\times 10$ ballooning factor of parameter size resulting in significant wire transfer overhead. Moreover, the cost of XML serialization/deserialization could be considerable or even dominant. [5] reports that SOAP-based RMI is slower than Java RMI by orders of magnitude.

Expressibility of SOAP and WSDL for GridRPC SOAP and WSDL target business applications, whereas GridRPC targets the requirements of scientific applications. Previous research showed that, features of GridRPC systems mentioned above is difficult to realize for CORBA[12]. It is not clear whether SOAP and WSDL are sufficiently expressible to support such features.

3. The 1st Prototype

As the first prototype, we implemented Ninf on Apache SOAP "naively". Apache SOAP is written in Java, consisting of a set of client libraries to allow client access to SOAP servers, and the Apache SOAP server that runs as a Servlet. We employed the Apache SOAP as the message layer, while preserving the client-side API for the original Ninf system(Figure 3). Since Apache SOAP does not

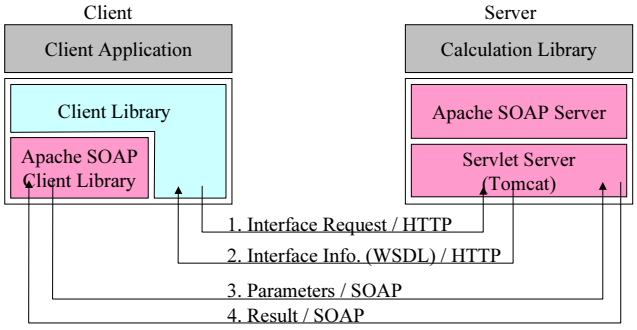

Figure 3. Ninf on Apache SOAP

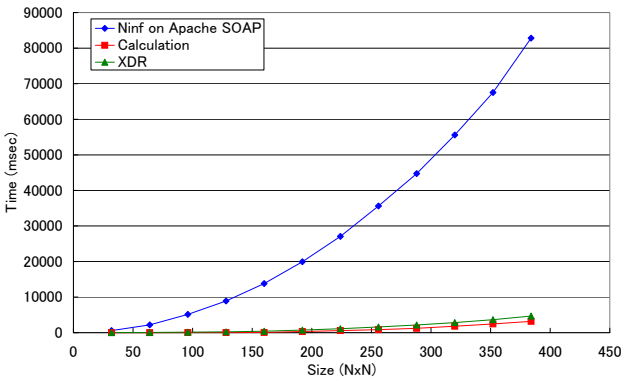

Figure 4. Ninf on Apache SOAP performance (LAN)

support WSDL we added a WSDL module for IDL management. We took steps to simplify the implementation to mainly evaluate the performance and Web services adequacy for GridRPC: for example, all the modules were implemented in Java, and due to the limitations of Apache SOAP, we did not implement some GridRPC features such as 1) multiple Out parameters, and 2) In/Out parameters, etc.

For the server, we employ the Apache SOAP server itself as the GridRPC server. Gridified libraries are registered with Apache SOAP server, along with the WSDL file describing its interface. For the client, we use the Apache SOAP library as the underlying message transport. The GridRPC call initially transfers the URL signature of the library, with which the server does the appropriate lookup and the obtained IDL in WSDL format is sent back to the client, which uses the information to serialize the parameters appropriately, and sent as a SOAP message. The server then deserializes the parameter, invokes the library, and then serializes and sends back the result, again as a SOAP message using the Apache SOAP library. The calls is performed while preserving the GridRPC API, and thus the use of Apache SOAP is largely transparent to the user.

We then evaluated the performance of Ninf on Apache SOAP by gridifying a simple matrix multiply library. Although a simple library, we felt that it serves as an appropriate benchmarking yardstick for two reasons: 1) it largely embodies ratio of complexity between communication vs. computation (for n-by-n matrix it is $O(n^2)$ vs. $O(n^3)$) that is representative of dense matrix application kernels, and 2) for sparse matrices computation is more dominant, which will mostly hide the GridRPC overhead irrespective of implementation, while communication dominant applications such as fluid dynamics, Grid computing itself may not be feasible.

As a testbed, we employed nodes of the PrestoII cluster in our laboratory at Titech. There are 64 nodes of dual Pentium III 800MHz w/640MB memory, with DEC 21140AF interconnected with a full-bandwidth 100Base-T switch. Relevant software consists of Linux 2.2.19, IBM Java 1.3.0, and Jakarta Tomcat 3.2.3. In evaluations of LAN environment, both the client and server are on the same LAN. For evaluations of WAN environment, we employed a node at AIST as a client, and a node of PrestoII cluster as a server. The client is a Sun Ultra-Enterprise machine with SPARC 333MHz $\times 6$ and 960MB memory. Relevant software consists of Solaris 5.7, Sun Java 1.3.0, and Jakarta Tomcat 3.2.3. Figures 4, 5 show the result in LAN environment and WAN environment respectively. The horizontal axis denotes the array size n, and the vertical axis shows the execution time. We also compare the execution time of a XDR-based GridRPC system we implemented in Java, matching closely the software structure of the original C-based Ninf implementation. We easily observe that our naive implementation of Ninf on Apache SOAP is terribly inefficient, with orders of magnitude increase in GridRPC execution time compared to the XDR-based implementation. This is consistent with the previous, related reports [5].

Detailed performance analysis revealed that the overhead is largely due to inefficient handling of large XML data by Apache SOAP which uses the DOM[6] parser for deserialization of SOAP messages. The DOM parser needs to receive the entire XML data before any analysis, as it will initially construct in memory a DOM object tree that represents the entire XML structure subject to parsing. The drawbacks include 1) we cannot deserialize a SOAP message until we receive the entire message, and 2) more seriously, space will be grossly inefficient because the entire DOM tree needs to be in memory; in fact this even consumes more space compared to the original textual XML representation by several factors.

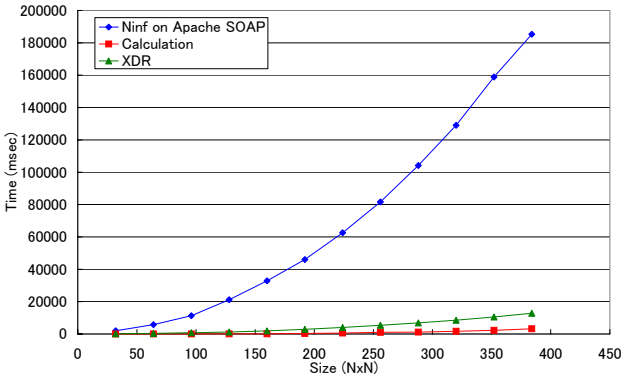

Figure 5. Ninf on Apache SOAP performance (WAN)

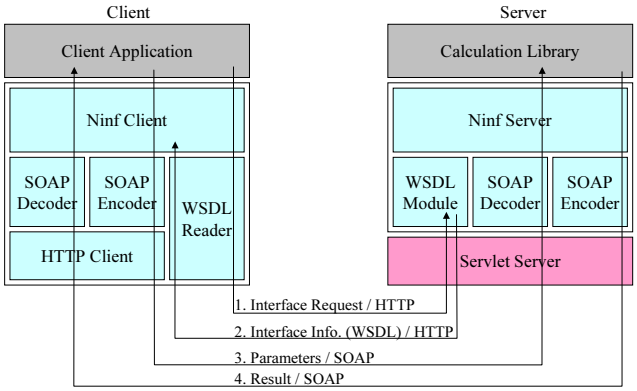

Figure 6. Ninf-on-SOAP architecture

4. Ninf-on-SOAP (2nd Prototype)

To solve the performance inadequacy, we designed and implemented a 2nd prototype system called Ninf-on-SOAP(Figure 6), which embodies customized SOAP serializer/deserializer tuned for speed. Also, it supports other features of Ninf GridRPC API including multiple Out parameters, and In/Out parameters. As is with the 1st prototype, all modules are implemented in Java.

4.1. Ninf-on-SOAP Server

The Ninf-on-SOAP server runs as a Servlet, and the global runtime architecture is largely the same as the Ninf-on-Aparche SOAP. Individual system components, are however very different:

WSDL module Upon GridRPC invocation from the client, the WSDL module reads the registered WSDL file corresponding to the client-URL-designated gridified library. In order to parse&analyze the WSDL file, a Java library called WSDL4J is utilized. The WSDL file will have been generated from the Ninf scientific IDL using the new version of the Ninf IDL compiler, and will contain the following information required on the server side: a) Classname that embodies the gridified method (library), b) Gridified method (library) name, c) Name, type information, and input/output mode of each parameter, and d) Ordinal position of each parameter.

SOAP deserializer This module deserializes the SOAP message sent by the client. The SOAP message is assumed to comply with the IDL information denoted by the corresponding WSDL—otherwise an error would occur For XML analysis, we employ the SAX[10] parser, which is event-driven and does not need to maintain the entire XML data in memory. Also, SAX allows on-the-fly serialization/deserialization of SOAP messages, a property will shall exploit later for optimization. Each deserialized parameter is placed into a vector, where the order of placement is determined by the information embedded within the WSDL.

The Invoker The Invoker executes the gridified library. In the prototype implementation, the executed library is a single Java method, but in general it could be any "gridified" C, C++, or Fortran functions with arbitrary call-by-reference array parameters.

The SOAP Serializer The SOAP serializer serializes into XML the returned result of the gridified Java method, as well as the Out and In/Out parameters from the parameter vector, and sends them back to the client as SOAP messages.

4.2. Ninf-on-SOAP Client

Ninf-on-SOAP client library retains the original simplicity and the small size of the original Ninf client. When a GridRPC call is made, the client sends the URL designation of the library to the server, and the server in turn sends back the WSDL file specifying the interface information, that has been compiled from the Ninf scientific IDL. The client uses the WSDL information to serialize the In and In/Out parameters, and sends them as SOAP messages. After the computation is done of the server, the client receives the return Out and In/Out parameters and deserializes them, again according to the WSDL interface information.

As for handling of SOAP as the underlying messaging layer and WSDL as IDL, much of the structure is similar to that for the server side. The SOAP parser is built using SAX as is with the server.

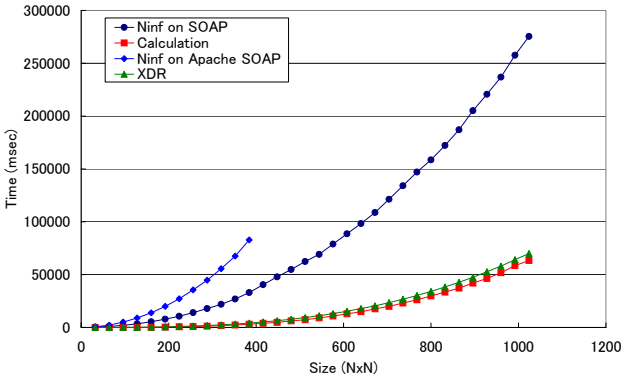

Figure 7. Performance of Ninf-on-SOAP (LAN)

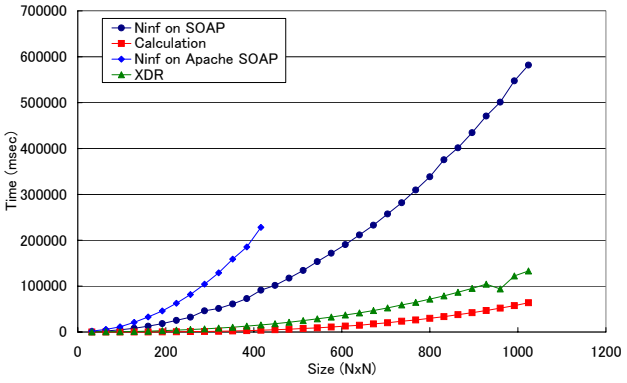

Figure 8. Performance of Ninf-on-SOAP (WAN)

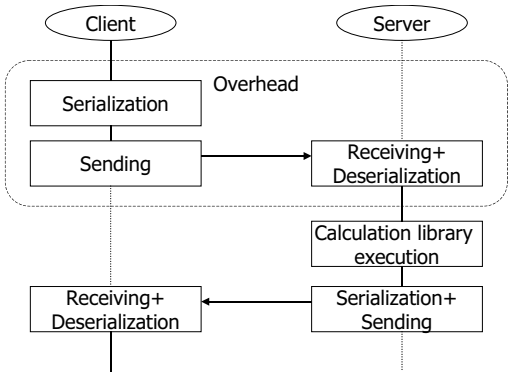

Figure 9. Ninf-on-SOAP execution flow

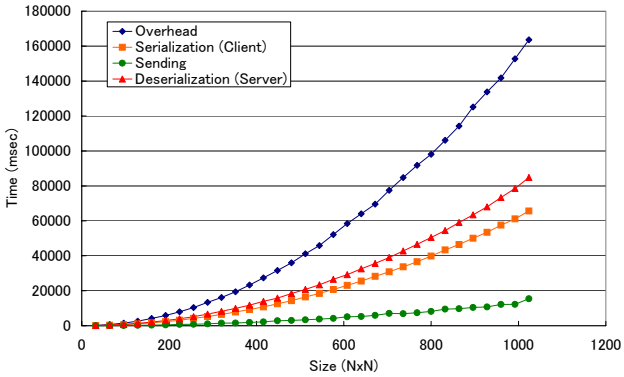

Figure 10. Detailed overhead analysis (LAN)

5. Benchmarking and Optimizations

We measured the performance of Ninf-on-SOAP under the same environment as in Section 3. Figures 7, 8 show the results plus the performance of the 1st prototype (Ninf on Apache SOAP) for comparison. Although the performance is significantly improved, the overhead is still high compared to the XDR version.

We performed detailed analysis of where time is being spent along the execution flow for Ninf-on-SOAP on a GridRPC call (Figure 9). The dotted rectangle enclosure indicates the overhead prior to server-side computation, consisting of serialization, wire transfer, and deserialization phases. Figure 10 indicates that serialization/deserialization are major sources of overhead, while wire transfer is very small under the 100Base-T environment, despite nearly an order of magnitude increase in message size. Even in WAN environment (Figure 11), the cost of serialization/deserialization is big although wire transfer occupies the biggest part of the overhead because of the increase in message size.

We next perform a series of optimizations, and investigate whether performance overhead is inherent to using Web services, or that with clever implementation techniques, we can remove much of the overhead.

5.1. Optimization 1: HTTP Content-Length Elimination

One significant overhead is that serialization, wire transfer, and deserialization are done in sequence and not overlapped (Figure 9). This is because Content-Length header field is required on a HTTP Post request, and thus the client must construct the entire SOAP message in memory first in order to calculate the message length. However, since SOAP messages are XML-based, HTTP servers could determine the end of the message by counting pairs of XML tags, provided that the XML is syntactically correct. Since GridRPC systems generate the SOAP messages and as such

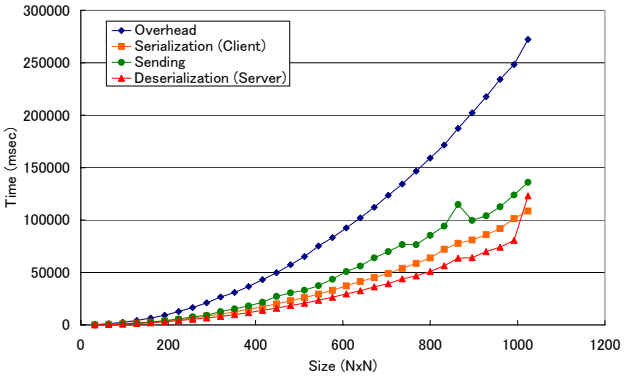

Figure 11. Detailed overhead analysis (WAN)

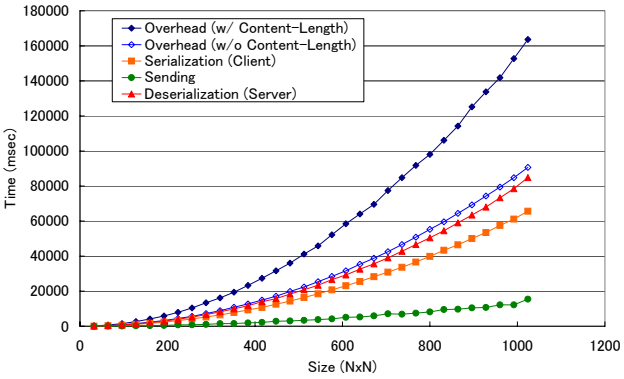

Figure 12. HTTP Content-Length overhead (LAN)

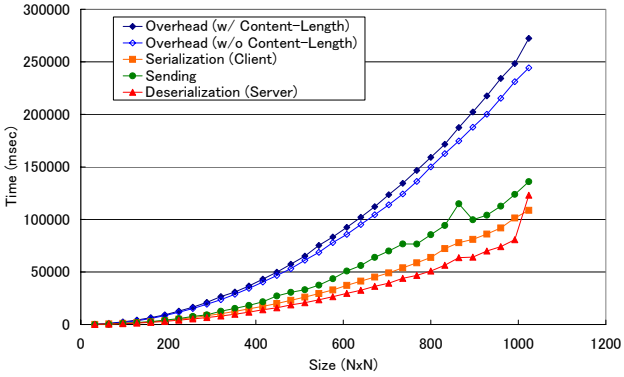

Figure 13. HTTP Content-Length overhead (WAN)

malformed XMLs do not happen, we can safely omit the field and effectively overlap and pipeline server side serialization, wire transfer, and server-side deserialization (despite that it is a slight violation of the RFC).

Figure 12 shows that, the overhead is roughly the sum of serialization and deserialization time in LAN environment, whereas with optimization it decreases to be roughly half (about 55%). The overhead is also reduced in WAN environment (Figure 13) even though the decrease is not apparent as in LAN environment because wire transfer occupies the big part of the overhead. This demonstrates that such overlapping & pipelining are quite effective for using SOAP as the GridRPC transport

We are considering ways to comply with RFC while retaining performance. One approach is to employ a very fast two-pass algorithm such as employed by [13], which only counts the message length on the first pass. The 2nd approach is to roughly estimate the length of XML representation per each field whose length may be variable (such as numerical values), and padding with whitespaces when the textual representation is shorter. The 3rd approach is to employ Chunked Transfer Coding supported by HTTP 1.1, and separate a big message into a small chunks, each with its own size indicator. Evaluation of these methods is a future work.

5.2. Optimization 2: Base64 Encoding

SOAP offers semi-binary encoding of parameters via Base64 encoding. Here, we observe the effect of such encoding instead of sending the array as a collection of element data. Note that Ninf standardizes binary encodings of array data, but we are not necessarily straightforwardly sending the memory image of individual arrays, but rather will have to perform various pack/unpacking as mandated by various specifications in the scientific IDL.

Figures 14, 15 show that, by applying Base64 encoding, we obtain approximately 75% overhead reduction both in LAN environment and WAN environment, primarily due to elimination of parsing overhead in serialization and deserialization thanks to substantial decrease in the number of XML tags.

One drawback of this approach is that we lose the information of each item of arrays in wire format. However, treating an array as a big binary data is acceptable in the most of scientific computing. One alternative is to use gzipped transfer supported by HTTP 1.1, but the cost of serialization/deserialization would not be reduced.

5.3. Putting them all together

Figures 16, 17 compare the performance when optimizations are applied individually and also in combination,

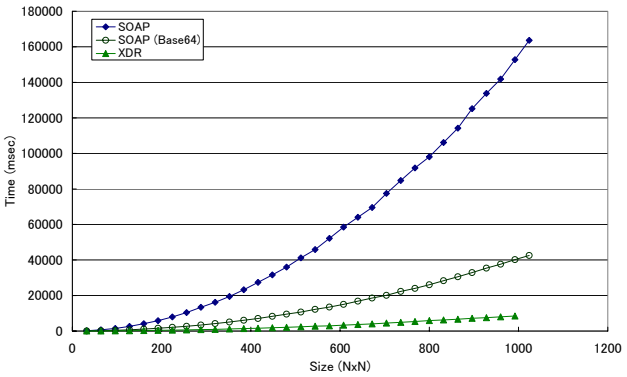

Figure 14. Overheads of different encodings (LAN)

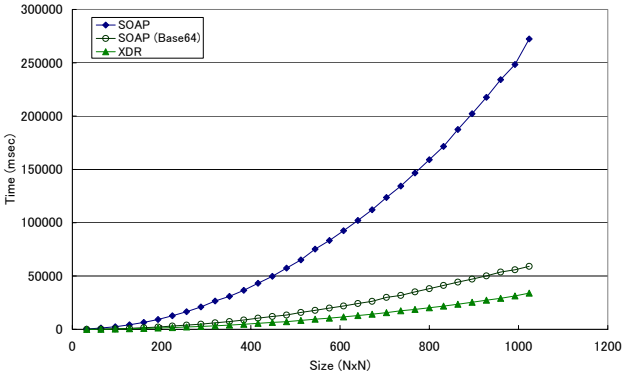

Figure 15. Overheads of different encodings (WAN)

illustrating that Ninf-on-SOAP and optimizations applied thereof improves performance significantly over the naive Ninf on Apache SOAP. In fact, compared to the XDR implementation performance is almost competitive; thus, for less communication-intensive, typical GridRPC application, we expect the optimized Ninf-on-SOAP to be quite usable.

To compare our implementation with other fast SOAP implementations, we compare our results with the performance of XSOAP[7], a new fast SOAP-based RMI developed at Indiana University. Because XSOAP does not yet support 2-dimensional arrays, we used a 1-dimensional array to express matrix data. The result shows that performance of XSOAP is nearly equal to our 2nd prototype system, Ninf-on-SOAP (not optimized).

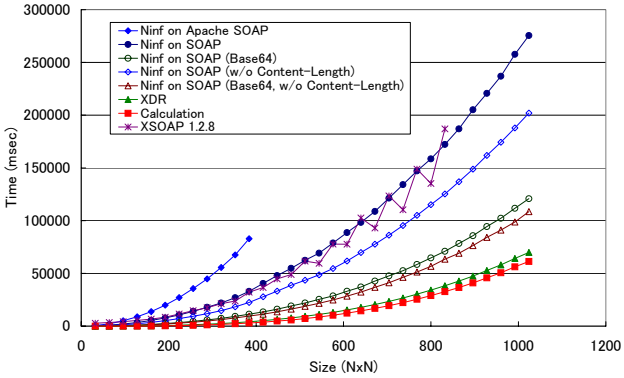

Figure 16. Performance of various versions (LAN)

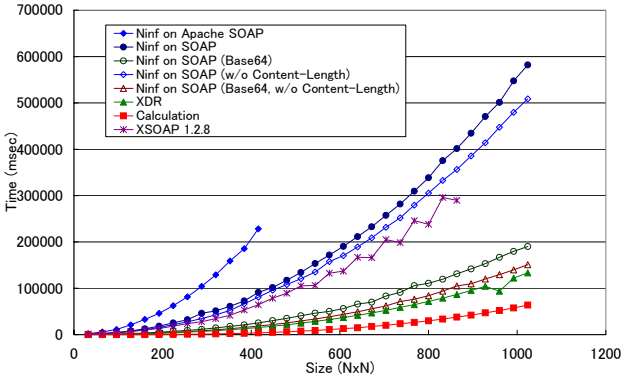

Figure 17. Performance of various versions (WAN)

6. SOAP/WSDL Expressibility

Here we qualitatively assess whether SOAP and WSDL are sufficiently expressible for scientific IDL features of GridRPC. Figure 18 is an example of Ninf IDL for a matrix multiply function call, and Figure 19 is a part of WSDL converted from the Ninf IDL. Since the Ninf system is designed for scientific computing, the supported data type is tailored for such a purpose; for example, the data types are limited to scalars and their multi-dimensional arrays. On the other hand, there are special provisions in the IDL for scientific applications, such as support for expressions involving input arguments to compute array size. Therefore, there are several features that are not expressible by current specifications of SOAP and WSDL, the target of which is business applications. We will discuss representative examples below.

```
Define dmmul(mode_in int n,
             mode_in double A[n][n],
             mode_in double B[n][n],
             mode_out double C[n][n])
"... description ..."
Required "libxxx.o"
Calls "C" dmmul(n,A,B,C);
```

Figure 18. Ninf IDL

```
<message name="DMMulInput">
  <part name="n"
        type="xsd:int"/>
  <part name="A"
        type="tns:ArrayOfArrayOfdouble"/>
  <part name="B"
        type="tns:ArrayOfArrayOfdouble"/>
</message>

<message name="DMMulOutput">
  <part name="C"
        type="tns:ArrayOfArrayOfdouble"/>
</message>

<portType name="MatrixType">
  <operation name="dmmul"
             parameterOrder="n A B C">
    <input message="tns:DMMulInput"/>
    <output message="tns:DMMulOutput"/>
  </operation>
</portType>
```

Figure 19. Core part of WSDL

Array Size Specification For scientific computing, various array information such as size depend on some call parameters (typically one or combination of scalar parameters). As a simple case, consider a matrix multiply function call for matrices of size n, $C = AB$, denoted as `dmmul(n, A, B, C)`, and when gridified becomes `gridrpc_call("dmmul", n, A, B, C)`. Even under local, sequential setting the size parameter `n` is required since arrays are passed as reference and the Fortran/C typesystem doesn't embody size information. But this alone is insufficient for an RPC system, since it must know at the call site a) n denotes such information, b) A and B has to be sent to the server, and c) C must be returned. In GridRPC, such information (not expressible in business IDLs such as CORBA) is denoted in scientific IDL. WSDL however currently lacks the ability to directly express array sizes or such dependencies.

Subarray, strides of array Likewise, given some numerical library, it is typical for only portions of arrays, such as subarrays and array strides, are accessed, even if the entire array is seemingly passed by references. Although SOAP supports partially transmitted arrays and sparse arrays, and would be possible to express such subarrays and array strides, WSDL does not embody any specifications of such data types.

Interoperability and usability WSDL can describe the parameter order with the parameterOrder attribute within an operation. Ninf-on-SOAP uses this attribute to denote the order of parameters. However, because parameterOrder attribute is optional, Ninf-on-SOAP cannot decide the order if it encounters some WSDL lacking parameterOrder. Currently, Ninf-on-SOAP handles such a case assuming that the order is the same as it appears in the WSDL file, and that input parameters are followed by output parameters.

There are other interoperability issues, such as WSDL facilitating multiple ways to specify the same datatype, such as arrays. Since complete implementations will have to deal with all such different representations, assuring interoperability will be somewhat a tedious task.

7. Related Work

There have been numerous work on efficient RPC implementation in the '80s, but they predate the primary focus of our work—Grid(RPC) vs. XML, SOAP, WSDL, etc.—and their results are not directly applicable. [12] performs qualitative and quantitative comparisons of GridRPC systems with CORBA. There, the wire speed between the most efficient implementations were similar, and the issue was with ease of use of GridRPC-style client interface management and expressibility of scientific IDLs compared to CORBA IDLs. [5] compares different incarnations of RMI protocols, and shows that their implementation of SOAP-based RMI is slower than Java RMI or Nexus RMI by an order of magnitude. They only measure bytearray transfer efficiency and does not discuss GridRPC/scientific IDL-like properties. As a followup work [11] describes a SOAP-based RMI system, SoapRMI, implemented using their own fast XML parser implementation called the XML Pull Parser. Again, the focus is on general RMI. [14] proposes a XML-based Grid system, which uses XML (actually similar to but not WDSL) only for interface information description, but employs binary-based data format for data exchange for performance efficiency; they do not discuss scientific IDL properties, nor interoperability issues by sticking within XML standards.

8. Conclusion

We investigated whether GridRPC could be formidably be implemented using Web services, in particular SOAP as the underlying wire protocol, and WSDL as the system-level IDL while retaining the ease-of-use of current GridRPC. We found that naive implementations would result in significant overhead, but on-the-fly, pipelined serializaton/deserialization, coupled with elimination of XML tags by the use of base64 encoding, results in significant speedup to be almost competitive with binary wire protocol implementations. However, some features of Ninf scientific IDL was difficult to express with WSDL, namely dependencies between scalar parameters and the actual array size, array stride, subarray specifications, etc. Small extensions to the WSDL will likely resolve this, however.

For future work, we construct a fast XML parsing algorithm especially tailored for SOAP and GridRPC serialization/deserialization, along with using dynamic compilation techniques for dynamic IDL management at call sites.

References

[1] UDDI Technical White Paper. Technical report, uddi.org, June 2000.

[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. rystyk Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1, W3C Note, May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[3] H. Casanova and J. Dongarra. Applying NetSolve's network-enabled server. *IEEE Computational Science & Engineering*, 5(3):57–67, July/Sept. 1998.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, W3C Note, March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[5] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon. Requirements for and Evaluation of RMI Protocols for Scientific Computing. In *Proc. of SuperComputing 2000*, November 2000.

[6] A. L. Hors, P. L. Hegaret, L. Wood, G. Nicol, J. Robie, and M. Champion. Document Object Model (DOM) Level 2 Core Specification Version 1.0, W3C Recommendation, November 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113.

[7] Indiana University. XSOAP toolkit (aka SoapRMI). http://www.extreme.indiana.edu/soap/.

[8] H. Nakada and S. S. Satoshi Matsuoka. Bridging Ninf and NetSolve, 1997.

[9] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi. Utilizing the Metaserver Architecture in the Ninf Global Computing System. In *High-Performance Computing and Networking '98, LNCS 1401*, pages 607–616, 1998.

[10] Simple API for XML (SAX). http://www.saxproject.org/.

[11] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley. Design of an XML based Interoperable RMI System: SoapRMI C++/Java 1.1. In *Proc. of The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, June 2001.

[12] T. Suzumura, T. Nakagawa, S. Matsuoka, H. Nakada, and S. Sekiguchi. Are Global Computing Systems Useful? Comparison of Client-server Global Computing Systems Ninf, NetSolve Versus CORBA. In *Proc. of 14th IEEE Intl. Parallel & Distributed Processing Symp.*, pages 547–556. IEEE Computer Society Press, 2000.

[13] R. A. van Engelen and K. A. Gallivan. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In *Proc. of IEEE CC Grid Conference 2002*, May 2002.

[14] P. Widener, G. Eisenhauer, and K. Schwan. Open Metadata Formats: Efficient XML-Based Communication for High performance Computing. In *Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing-10 (HPDC-10)*, pages 371–380, August 2001.