Benchmarking XML Processors for Applications in Grid Web Services

Michael R. Head*
Madhusudhan Govindaraju†
State University of New York (SUNY) at Binghamton
Robert van Engelen‡
Wei Zhang§
Department of Computer Science, Florida State University

Abstract

Web services based specifications have emerged as the underlying architecture for core grid services and standards, such as WSRF. XML is inextricably inter-twined with Web services based specifications, and as a result the design and implementation of XML processing tools plays a significant role in grid applications. These applications use XML in a wide variety of ways, including workflow specifications, WS-Security based documents, service descriptions in WSDL, and on-the-wire format in SOAP-based communication. The application characteristics also vary widely in the use of XML messages in their performance, memory, size, and processing requirements. Numerous XML processing tools exist today, each of which is optimized for specific features. To make the right decisions, grid application and middleware developers must thus understand the complex dependencies between XML features and the application. We propose a standard benchmark suite for quantifying, comparing, and contrasting the performance of XML processors under a wide range of representative use cases. The benchmarks are defined by a set of XML schemas and conforming documents. To demonstrate the utility of the benchmarks and to provide a snapshot of the current XML implementation landscape, we report the performance of many different XML implementations, on the benchmarks, and draw conclusions about their current performance characteristics. We also present a brief analysis on the current shortcomings and required critical design changes for multi-threaded XML processing tools to run efficiently on emerging multi-core architectures. [1]

Keywords: XML, Benchmarking, Multi-Core

1 Introduction

Over the past few years, designers of Web services have closely collaborated with the grid community to propose numerous XML-based protocol specifications to bridge the platform and programming language gap in heterogeneous wide-area systems. XML has many important features, including platform and language independence, flexibility, expressiveness, and extensibility. Thus, the combination of these characteristics with the interoperability trait of Web services is an attractive way to compose distributed applications. Additionally, the use of XML based protocols for security, routing, messaging, resource policies, workflows, events, and other tasks, provides an effective platform to build applications over computational grids [Berman et al. 2003; Foster and Kesselman 1998].

The recently adopted standards such as the Open Grid Services Architecture (OGSA) [Foster et al. 2005] and Web Services Resource Framework [WSRF 2004] define a set of standard interfaces and behaviors of grid services in terms of Web services based technologies. Some of the other important standards and specifications in the Web services space include Web Services Description Language (WSDL) [Christensen et al. 2001], SOAP (formerly, Simple Object Access Protocol) [Gudgin et al. 2003], Business Process Execution Language for Web Services (BPEL4WS) to orchestrate workflows, and WS-Security set of XML specifications. Additionally, many grid applications use well-defined XML schemas for the XML documents used in various parts of the application.

XML is a ubiquitous tree-oriented data representation language. A WSDL document is an XML based specification that provides a standard language to precisely specify all the

*email:mike@cs.binghamton.edu
†email:mgovinda@cs.binghamton.edu
‡email:engelen@scs.fsu.edu
§email:wzhang@scs.fsu.edu

[1]Supported in part by NSF grants IIS-0414981, CNS-0454298, BDI-0446224 and DOE Early Career Principal Investigator grant DEFG02-02ER25543.

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 ©2006 IEEE

information necessary for communication with a Web service, including the interface of the service, its location, the details of the data types it uses, and the list of communication protocols it supports. SOAP is the most widely used communication protocol for Web services, facilitating the exchange of XML-based structured information with HTTP widely used as the transport medium. Due to the heterogeneous nature of the grid infrastructure and the diverse characteristics of applications, the use of XML in SOAP makes it ideally suited to serve as the common standard communication protocol.

Various studies [Abu-Ghazaleh et al. 2004b; Chiu et al. 2002; Govindaraju et al. 2000], however, have shown that the use of XML can hinder performance. XML primarily uses UTF-8 as the representation format for data. Sending commonly used data structures via standard implementations of SOAP incurs severe performance overheads, making it difficult for applications to adopt Web services based grid middleware. Due to the widespread adoption of standards in Web services by the grid community, it is critically important to investigate the impact on performance for the kinds of XML documents used in grid applications. Several novel efforts to analyze the bottlenecks and address the performance at various stages of a Web services call stack have been discussed in the literature [Abu-Ghazaleh et al. 2004a; Abu-Ghazaleh et al. 2004c; Abu-Ghazaleh et al. 2004b; Chiu et al. 2002; Govindaraju et al. 2000; van Engelen 2004a; van Engelen and Gallivan 2002]. The flexibility and loose coupling of XML-based standards allows senders and receivers of XML documents to independently deploy selected optimizations, according to the communication patterns and data structures in use.

Some of the optimizations for XML toolkits (also referred to as *XML processors* in this paper) discussed in the literature include the following: (1) gSOAP parser [van Engelen 2004a] uses look-aside buffers to efficiently parse frequently encountered XML constructs; (2) the XML Pull Parser (XPP) [Slominski 2004] caches parsed strings to avoid multiple allocations of strings; (3) we earlier proposed a technique to enhance performance of parsing XML schemas by using schema-specific parsing along with *trie* data structures so that frequently used XML tags are parsed only once [Chiu et al. 2002; van Engelen 2004b]; (4) gSOAP uses a performance aware compiler to efficiently parse XML constructs that map to C/C++ types. It uses a single-pass schema-specific recursive-descent parser for XML decoding and dual pass encoding of the application's object graphs in XML [van Engelen and Gallivan 2002]; (5) The TDX parser [Zhang and van Engelen 2006] uses a table driven approach to combine parsing and validation into one pass to enhance the processing time for documents; (6) The VTD-XML [Zhang 2003] parser achieves performance improvement via incremental update, hardware acceleration, and native XML indexing.

The MetaData Catalog Service (MCS) [Singh et al. 2003] and the reference implementation of the WSRF specification, available from the Globus website [Globus Toolkit 2002], use the Axis [Axis Java 2002] toolkit to process XML documents. Our results show that for micro-benchmarks and data-structures commonly used in grid applications, Axis is not a good choice in terms of performance. However, as the architecture of the reference implementation of WSRF (and even Axis) is modular in nature and facilitates the use of specialized pluggable modules for various aspects of Web services, the results of the benchmark framework can be used to plug in specialized XML processing modules for each target application. Other significant efforts to implement the WSRF implementation that have a modular design include WSRF.NET [Humphrey and Wasson 2005] and WSRF-Python [Govindaraju et al. 2005].

It is important to compare, contrast, and evaluate different XML implementations, so that end-users can make informed decisions on which toolkit to use for their particular application. Specifically, the motivations for the design of a comprehensive performance evaluation framework for XML processors are:

- Grid applications place a wide range of requirements on the communication substrate and data formats. These requirements include low latency, high throughput communication, minimal memory footprint for improved caching efficiency, specialized handling of scientific data, and overlap of computation and communication by streaming XML messages via HTTP 1.1 protocol. These disparate requirements have led to a wide range of design and implementation choices. A comprehensive benchmark suite tailored for grid applications can aid in determining the XML (and Web services) toolkit that has the most optimized implementation for the class of grid applications under consideration.
- A wide range of implementations of XML Parsers is available [SoapWare.org 2001], including Xerces (DOM and SAX) [Xerces 2003], gSOAP-parser [van Engelen and Gallivan 2002], Piccolo [Oren 2002], Libxml [Veillard 1998], Expat [Clark 1998], kXML [Haustein 2000], XPP3 [Slominski 2004], VTD-XML [Zhang 2003], and Qt4 [Trolltech 1998]. Simple and straight forward implementations of XML parsing paradigms can result in a severe impact on performance. A comprehensive benchmark suite can help library developers identify and isolate the modules in their toolkits that need to be optimized. Ideally, toolkits will be designed to determine the data structures, use-cases, and communication patterns in the application code and have the ability to dynamically switch to the most optimized module for the use-case scenario.
- The reference implementation of the WSRF specification, available from the Globus Alliance web-

site [Globus Toolkit 2002], uses the Axis[Axis Java 2002] toolkit. The architecture of the reference implementation is modular in nature and facilitates the use of specialized pluggable modules for various aspects of Web services. The proposed framework will facilitate in the addition of application and feature specific modules to WSRF implementations. For example, the WSRF-C implementation can be enhanced by incorporating a Schema Specific Parser (SSP)[Chiu et al. 2002; van Engelen 2004b] that kicks-in when data conforming to known schemas is encountered; or a switch can be added to the serialization handler so that *differential serialization* [Abu-Ghazaleh et al. 2004b] is used for cases when similar content or structure of XML data is being repeatedly exchanged.

- The current set of Grid Web services tools are not tailored to utilize the capabilities for parallelism available in the emerging multi-core architectures. The lessons gained from the execution of the benchmarks will also provide insight into software design of toolkits and the possible changes required in the XML document structure itself, to aid in automatic detection of regions in the XML payload that can be processed in parallel.

With the reasons mentioned above as motivation, we have designed and developed a common standard XML benchmark suite for testing the performance and scalability of different XML toolkits, with a focus on data structures commonly used in grid services and applications. The SOAP community currently uses a set of well-known SOAP payloads and interfaces to test the *interoperability* of various toolkits [XMethods.com 2001]. Our work complements these efforts in that it aims to provide a standard set of workloads to test the various *features* and *performance characteristics* of XML implementations, rather than just the interoperability via the SOAP protocol. In designing these benchmarks, we draw on our experience in implementing and optimizing features of three different independent toolkits for Web services: gSOAP [van Engelen 2003; van Engelen 2004a; van Engelen and Gallivan 2002], XSOAP [Slominski et al. 2001; Chiu et al. 2002; Govindaraju et al. 2000; Slominski 2004], and bSOAP [Abu-Ghazaleh et al. 2004a; Abu-Ghazaleh et al. 2004c; Abu-Ghazaleh et al. 2004b].

Our benchmark suite provides grid middleware and application developers with working examples of XML features, and provides a common way of testing and assessing the performance of their specific implementation of these features. Another contribution is the snapshot it provides of the current performance of many popular XML implementations. This performance study provides insight into the relative strengths and weaknesses of different implementations under different usage scenarios, and demonstrates the utility of the benchmark suite. The benchmark suite and driver programs will be made publicly available from our website, and can be used to continuously compare the performance of available toolkits. The performance results in this paper show how effectively the benchmark suite can be used to select an appropriate XML toolkit for specific application needs.

Our benchmark framework will benefit both Web services developers and grid application programmers. Web services and grid middleware (library) developers can gain insights into the various factors and design choices that determine the performance of processing XML documents, thereby improving their ability to build better faster implementations. Application developers can use the benchmark suite to test and compare the performance of various aspects of different toolkits, and accordingly select the one that best suits their application's needs. We present performance results on many widely used toolkits including Xerces (DOM and SAX), gSOAP-parser, Piccolo, Libxml, Expat, XPP3, and Qt4.

The remainder of this paper is organized as follows. Section 2 describes the design of benchmarks in HPC. Section 3 provides the motivation, description and insights into the benchmark suite that we have designed. Section 4 describes our experimental setup and a representative set of performance results. We present a set of observations that can be drawn from our test results in Section 5. We present a simple analysis of XML toolkit design for multi-core architectures in Section 6. We discuss related work in Section 7 and end with pointers to future work in Section 8.

2 Benchmarks in HPC

Various benchmarks have been designed to test different features of HPC systems. These benchmarks can be broadly classified into two categories: low level probes and application based benchmarks [Chun et al. 2004].

Low-Level Probes: Benchmarks in this category are designed as probes to evaluate the performance of a system for fundamental operations. In recent months, the HPC Challenge Benchmark has been released by the DARPA HPCS program [Luszczek et al. 2005]. This benchmark is geared towards evaluating performance boundaries for future petascale computers. The components that the HPCC benchmarks are designed to stress are: LINPACK [Petitet et al. 2004] (CPU floating point performance), STREAM [McCalpin 1997] (memory subsystem and streaming performance), GUPS (Giga updates per sec) that stresses the communication fabric and protocol for short messages, and FFT stresses the bisection bandwidth of the system).

Representative Applications Based Benchmarks: these benchmarks capture the requirements of specific class of applications. The NAS Parallel Benchmarks (NPB) [Bailey et al. 1994], which originated from applications in computational fluid dynamics (CFD), are a set of programs de-

signed to compare the performance of parallel supercomputers. These benchmarks consist of three pseudo-applications and five kernels, including GridNPB3 [Frumkin and Wijngaart 2002], which includes serial and concurrent reference implementations of distributed applications in Fortran and Java. It also has a suite of benchmarks named Rapid Fire that test the capability of a grid infrastructure to manage and execute a large number of short lived processes. The Standard Performance Evaluation Corporation (SPEC) [SPEC 1992] corporation defines several popular benchmarks. These include Java Client/Server benchmark to measure the performance of J2EE application servers, speed of request handling capabilities of an NFS (Network File Server) system, and a suite for evaluation of the performance of parallel and distributed architectures. The ParkBench [Hey and Lancaster 2000] and SPLASH [Woo et al. 1995] benchmarks are also well known.

3 Design of the Benchmark Suite for XML Processing

Consistent with trends in HPC, we have divided the benchmark suite for XML processing tools into two categories: feature probes and application-class benchmarks. This section explains the rationale for each benchmark's design, and describes various optimizations that can be used to improve the performance of a toolkit for the features exercised by the benchmark. The benchmark suite is designed as a set of XML Schema documents along with example conforming documents, and a driver that reads trace data from local files and automates the testing process.

3.1 Feature Probes

These probe specific features of XML (and Web service toolkit) implementations such as toolkit overhead, processing of documents as required in serialization and deserialization in grid communication, management of arrays of various types, exercise of the buffering algorithms, handling of namespaces, scalability when dealing with coreferenced objects (multi-ref feature), and rate of handling typical SOAP messages.

3.1.1 Overhead

The *overhead* of the toolkit quantifies the minimum response time in processing an XML document. This measurement does not include costs associated with *cold start* or *warmup*, such as initialization costs due to loading of the necessary dynamic libraries or Java class files. Measurements are taken after the first few iterations.

The benchmark for this feature is a simple XML document that has a single element with no nested elements or attributes. The measured cost shows the minimum cost associated with memory allocation, de-allocation, and initialization of the parsers internal tables. This cost will be inherent to every use of the XML toolkit, and the results indicate which toolkit is best designed for extremely small XML documents.

3.1.2 Buffering

Since XML toolkits primarily deal with data in ASCII, they make extensive use of string operations, including search for specific sentinel characters, convert binary types to string formats, and incremental run-time allocation of strings. The default implementations of these features can often result in a performance penalty. The parsing and storage of frequently encountered XML constructs can be optimized via *look-aside* buffering schemes. gSOAP reuses the memory allocated for storing attribute name/value pairs to improve performance of parsing XML. This is particularly effective in parsing the `xsi:type` attribute which may be present in every XML element of the SOAP payload. Similarly, XPP3 caches parsed strings and avoids multiple allocations of strings for processing XML input with values that repeat frequently, such as in the case of arrays.

The benchmark for this feature is exercised by XML documents representing SOAP-encoded arrays of various sizes and primitive types. Managing the repeated occurrences of *xsi:type* for each element of the array tests the buffering algorithm of the XML toolkit. As described in Section 3.2, grid applications typically exchange arrays of various types, and are directly affected by this feature of the toolkit they employ.

3.1.3 Managing Namespace-qualified Elements

The primary purpose of namespaces is to distinguish between identical names of elements, attributes, and tags that appear in an XML document. The extensive use of namespaces in XML documents makes it critical to evaluate the implementation of this feature. Each namespace is associated with a URI. A specialized attribute *xmlns* is used by tags to point to a fully qualified name. In a typical XML document, there are usually a few *xmlns* attributes but a large number of references to these attributes. The standard implementation of namespaces involves the use of a stack to store namespace prefixes and associated URIs. The performance limitation of the stack implementation stems from the repeated comparison operations that are needed in this implementation module. An optimization to manage namespaces is to use one table lookup to determine a corresponding internal namespace prefix of the *xmlns* attribute. The table should be

populated with information obtained from the XML schema of the document being processed. In this scheme, the stack just records the translated prefixes to provide efficient matching of qualified tags. This results in reduction of the amount of storage and number of comparisons of prefixes.

The benchmark consists of XML documents in grid applications with plenty of *xmlns* bindings, such as those that are generated as a result of applying the canonicalization algorithm [W3C]. The canonicalization algorithm defines a standard form for an XML document, meant to guaranty bit-wise comparisons for logically equivalent documents. We chose canonicalized forms of example WS-Security standard documents for the benchmark. Another benchmark that tests this feature is the XML representation of nested data structures such as linked lists, wherein several tags and element names are identical. This forces a toolkit to apply its namespace resolution algorithms to correctly resolve all the names according to their namespaces.

3.1.4 Object Graphs and Co-Referenced Objects

An important requirement for Web services based grid applications is that data structures and object graphs be consistently stored and manipulated [van Engelen et al. 2006]. SOAP-RPC 1.1 encoding provides multi-referencing to serialize (cyclic) object/data graphs, wherein multi-ref accessors are placed at the end of a message, so that all multi-references are *forward pointing*. Object copying or pointer back-patching must be used by an XML processor for each forward pointing edge to complete the edge references in the partially instantiated object graph. The SOAP 1.2 RPC encoding format is more natural, and allows both forward and back edges, but no constraints are given to avoid object copying or back-patching. This design is analogous to the use of pointers and references in many programming languages to refer to one instance of an object from multiple locations.

When a streaming parser, such as Simple API for XML (SAX) [Xerces 2003] or XPP [Slominski 2004], is used, a co-referenced object can only be deserialized after the parser has processed the multi-ref objects at the end of the message. Even though the DOM model is simple to use for such cases, it imposes a performance penalty as the entire message has to be stored in memory. Our performance tests show that in the widely used Apache Axis toolkit, *every* object in the graph is serialized with `id` and `href` using an inefficient non-scalable run-time algorithm.

In Java toolkits, if the common approach of using the *equals()* method is invoked, instead of *IdentityHashMap*, to compare all objects to check for co-references, decoding an XML document representing a graph can result in an $O(n^2)$ serialization algorithm, hurting the scalability of the application. The gSOAP toolkit uses generated routines to decode the XML document and reconstruct the original data structure graph. The parser takes special care in handling the `id` and `ref` attributes to instantiate pointers, using pointer back-patching and object copying when required. When the data structure is reconstructed, temporarily unresolved forward references are kept in a hash table keyed with the `id` values. When the target objects of the references have been parsed and the data is allocated in memory, the unresolved references are back-patched.

The workloads that we have designed for this benchmark consist of XML representation of a graph of nodes, and an array of strings of various sizes, wherein some of the array elements are identical. A conforming toolkit needs to test for co-references for each node and element. Even though the use of a hash-table is efficient, for large arrays it may result in overflow chains, and the lookup may not always be in constant time.

3.1.5 Processing SOAP Messages

SOAP is the most widely used communication protocol in Web services based grid middleware. A SOAP message is formally specified as an XML infoset, which is an abstract description of the contents of the message. XML is the most commonly used on-the-wire representation of the infoset. A wide range of SOAP implementations, developed in various programming languages using different XML parsers, are available today. As a result, it is important to collect and analyze performance statistics for processing of XML messages that are generated as part of on-the-wire format of SOAP communication.

Our benchmark consists of SOAP messages for arrays of different data types and sizes that are commonly used in grid applications. The data types include floats, integers, doubles, strings, base64 encodings, and structs with few primitives. The size of the array for various payloads vary from a few elements to 100,000 elements, as we do not expect the SOAP protocol to be used for larger message sizes.

3.2 Application Class Benchmarks

The second set of benchmarks in our framework is application-oriented and captures typical XML messages in different classes of grid applications. The analysis of the these applications running on the grid infrastructure based on Web services will provide more insight into what new metrics and core kernel benchmarks need to be added to the suite for a more robust and well designed benchmark suite. Initial set of applications that we have considered include information service components, replica location services, resource management services, security components, and data grid services. In this paper we present results with example payloads of workflow documents, XML messages sent via the SOAP protocol in the MetaData Catalog Service (MCS),

application schemas such as HapMap [HapMap 2003] and BioMedical Applications, Mesh Interface Objects (MIOs) used in scientific computing, events stream used in applications such as Linked Environments for Atmospheric Discovery LEAD [LEAD Events 2003] project.

3.2.1 Workflow Documents

Grid workflows have emerged as critical tools to facilitate in the development of complex scientific applications. Workflows allow the integration of legacy code and Web services from various organizations, developed in different languages, into a a single distributed applications [Gannon et al. 2004]. There are many scientific workflow systems tailored for use in grid computing applications, many of which use XML based representations to specify the workflows [Slominski 2005].

We have curently added two sets of workflow documents: (1) example workflow documents from the Kepler [Kepler 2003] project for scentific applications. The Kepler project's goal is to provide an open source scientific workflow system to efficiently execute workflows using emerging Grid-based approaches; (2) example workflow documents currently used in for the LEAD application, which is used for creating an integrated, scalable cyberinfrastructure for mesoscale meteorology research and education. As our benchmark suite will be publicly available for download and use, we expect to add new workflow documents from other frameworks in the near future.

3.2.2 MetaData Catalog Service

The Metadata Catalog Service (MCS) [Singh et al. 2003] runs on top of a Web service that provides functionality to store and retrieve descriptive information (metadata) about logical data items. MCS has been developed as part of the Grid Physics Network (GriPhyN) project, with an overall aim of supporting large-scale scientific experiments. MCS is a classical example of a system that uses XML communication between clients and the Grid service, via the SOAP implementation of Axis [Axis Java 2002]. The performance study reported in [Singh et al. 2003] shows that the Web service overhead causes an average performance drop by a factor of 4.8. We used the MCS schema to generate compliant XML documents of various sizes to study the XML toolkit that is most ideally suited to address the performance bottleneck reported by the MCS authors.

3.2.3 Human Genome Project

The International HapMap project aims to develop a *haplotype* map of the human genome [HapMap 2003]. The schemas are used to describe the common patterns in human DNA sequence variation. It is expected that grid computing solutions will play a significant role in the human genome project. Our benchmark suite consists of synthetic workloads that are compliant with the schemas for HapMap, to determine the toolkit that performs best for this project.

3.2.4 Mesh Interface Objects

Mesh interface objects (MIO) structures are of the form (int, int, double), where the two integers represent a mesh coordinate and the double represents a field value. This data structure is often used by scientific components on the grid. MIOs are used in communication between two Partial Differential Equation (PDE) solvers in different domains. An example usage is in a climate model that ties together an atmospheric simulator with an ocean circulation simulator [Barron et al. 1994]. Another example is a fluid simulation that is coupled with a solids structure code, as is done in some industrial process modeling [Illinca et al. 1997]. Our benchmark framework consists of MIO payloads that test the scalability of the XML parser, as the number of MIOs is varied from a few to 100,000 elements.

3.2.5 Event Streams

WS-Notification and WS-Eventing have emerged as the standard XML-based specifications for asynchronous notifications to interested listeners. They define the message exchange formats along with the baseline set of operations required by producers and consumers of events. These event specifications provide a de-coupled communication medium for grid applications. Typical uses of events include monitoring, debugging, and reporting occurrences such as a successful creation of a remote file. Notification services is also an integral part of services described in the WSRF specification [WSRF 2004].

We have defined two types of events. First, a simple event data structure as a struct with three data members: an integer (sequence number), a double (time stamp) and a string to store the event message. This definition provides both simplicity and flexibility. The string can be used to store small values such as a *url* for GridFTP transfer, or a long string requesting resource properties from a WSRF service. Second, we have included XML documents conforming to the WS-Notification and Eventing schemas to conform to the requirements of many existing and emerging grid applications that are expected to use these specifications.

Our benchmark driver can be configured to choose the size of the elements in the events schema that accurately reflects the needs of events in the application of interest, and accordingly decide the best toolkit to process event streams.

3.2.6 WS-Security Documents

The WS-Security suite of security specifications address a broad range of issues concerning protection of messages exchanged in a Web services environment. This model brings together formerly incompatible technologies such as Kerberos and public key infrastructure. The broad set of specifications include authentication, authorization, privacy, trust, delegation, integrity, auditing, and confidentiality. The OGSA Security Working Group, whose charge is to address the grid security requirements, has declared that the OGSA security architecture will leverage the Web services security foundations published in the WS-Security specifications [Nagaratnam and Humphrey 2003]. Our benchmark suite consists of example documents from the WS-security specifications. A unique feature of these documents is the large number of namespaces for most of the elements.

Additionally, we have also included sample XML documents used by scientists at the National BioMedical Computation Resource (NBCR), who are building an end-to-end Web services architecture for Bio-Medical applications [Krishnan et al. 2005].

4 Representative Performance Results

The Linux test environment consisted of one dual core machine, with an Intel(R) Pentium(R) D CPU 3.00GHz with 256MB PC4200 RAM and a 7200 RPM 80GB SATA-2 drive running the i386 edition of Ubuntu Linux 5.10 ("breezy") with the 2.6.12 kernel compiled for i686 SMP processors. All C and C++ based parsers were compiled with gcc/g++ version 4.0.2. All Java-based parsers were compiled and run with the Sun Java 5 SDK, version "1.5.0_06". The C#-based parser is from the implementation from System.Xml in Mono version 1.1.8.3. The version of the other parsers presented are as follows: expat 1.95.8, gsoap 2.7.0d, libxml2 2.6.21, piccolo 1.0.4, xerces-c 2.6.0, xerces-j 1.4.4, and xpp3 1.1.3_6.

Figure 1 shows overhead incurred by various toolkits. Among the toolkits we tested, gSOAP-parser has the least overhead of 5.5 μs, and Expat's overhead at 14 μs is the next best. The Mono-Reader (developed in C#) parser, which is a light-weight pull-model based parser, has the least overhead (33 μs) among non C/C++ parsers. Both Mono-DOM and XPP3 have an overhead of approximately 60 μs. These two have the next lowest overhead among non-C/C++ parsers. Note that the Xerces implementations in both Java and C have relatively high overheads. Libxml, Piccolo, Qt4, perform better than Xerces, but have an overhead more than 1millisecond.

In Figure 2, we chose two grid applications (Workflow and

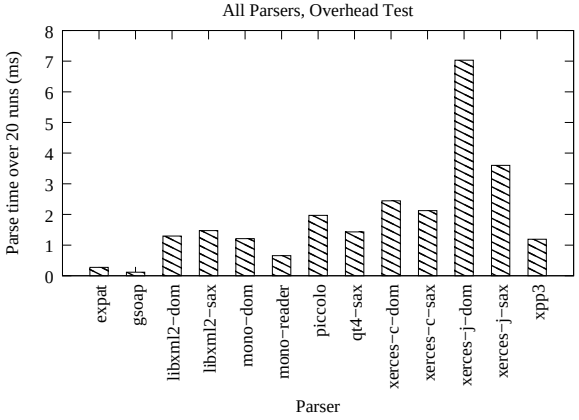

Figure 1: The overhead associated with each parser. *We run a tiny XML file through each parser 20 times and measure the parse time. Because the XML file is so small, this effectively measures each parser's setup and cleanup time.* gSOAP*'s overhead is the lowest at 110 μs.* Xerces-J-DOM*'s overhead is twice that of* Xerces-J-SAX *at 7029 μs.*

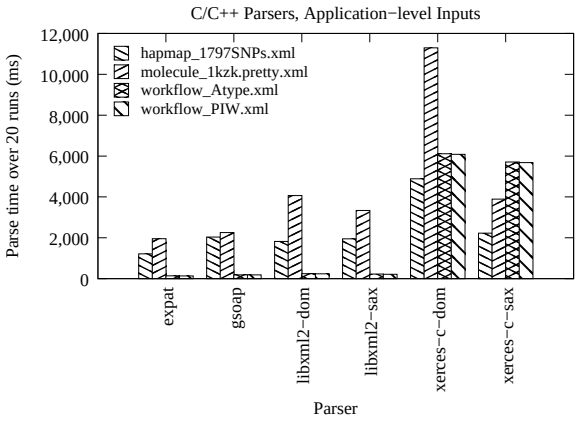

Figure 2: Performance of C/C++-based parsers on some large grid applications. *Files sizes range from 277KBytes (workflow_PIW.xml) to 4.9MBytes (hapmap_1797SNPs.xml) and are parsed 20 times in succession. All parsers processed the HapMap file in approximately 2s, with the exception of* Xerces-C-DOM*, which took about 5s.*

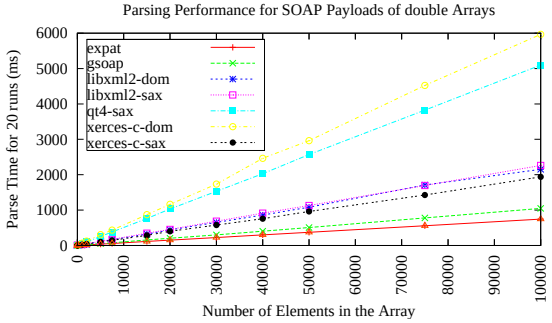

Figure 3: Scalability of C/C++-based parsers over arrays of doubles in SOAP payloads. *Here the parsers are fed XML documents containing SOAP-serialized arrays of doubles.* Expat *leads the group parsing a document containing 100,000 doubles 20 times in 744ms.* Xerces-C-DOM *generates a DOM each parse, and performs the same task in 5,965ms.*

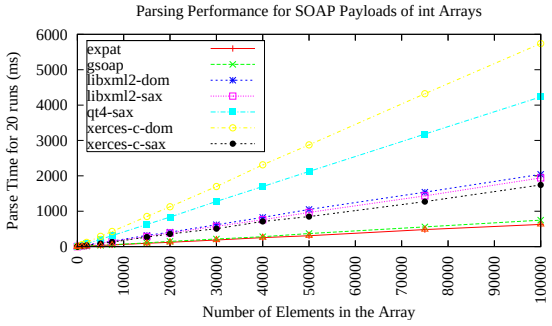

Figure 4: Scalability of C/C++-based parsers over arrays of integers in SOAP payloads. *Similar to Figure 3, we test each parser against a set of XML documents containing SOAP-serialized arrays of varying size. In contrast to Figure 3, all parsers improve when handling integers versus doubles, though* gSOAP *and* Qt4-SAX *both improve more than the others.*

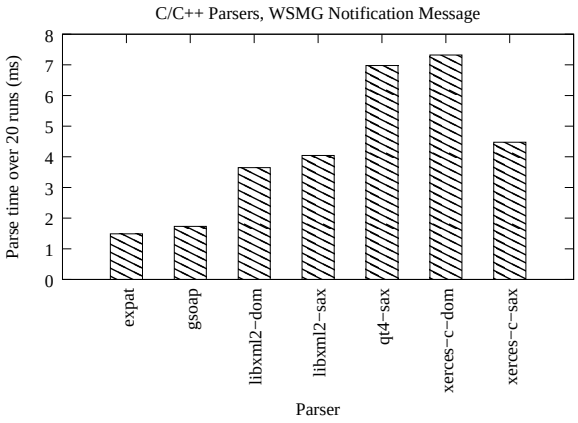

Figure 5: C/C++-based parsers using WS-MG notification messages. *Again,* Expat *is the best performing parser, processing the WSMG notification message 20 times in 1.48ms.* Xerces-C-Dom *tops the chart at 7.31 ms.*

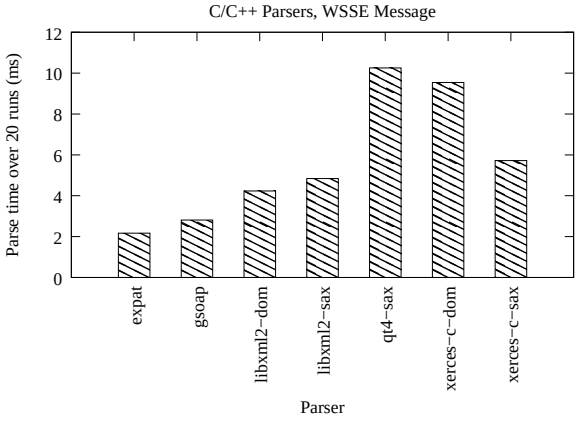

Figure 6: C/C++-based parsers using WSSE security messages. *The results are similar to those of Figure 5, except that* Qt4-SAX *performs the worst at 10.2 ms.*

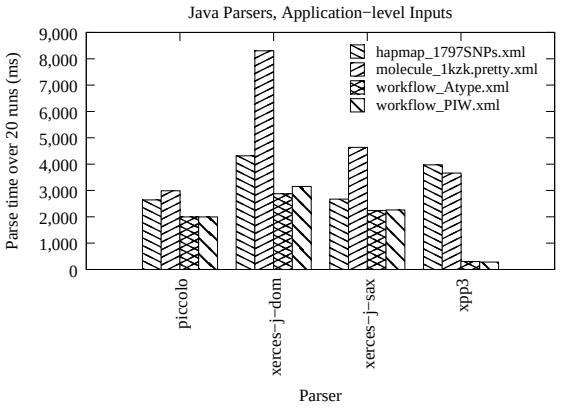

Figure 7: Performance of Java-based parsers on some large grid applications. *This is the same test as shown in figure 2, using Java-based parsers. There is some interesting variability here.* XPP3 *handles the workflow tests in roughly 10% the time of the other parsers, but is squarly in the middle of the group for the HapMap and Molecule tests.*

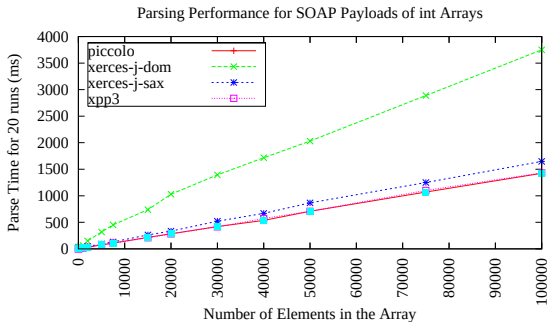

Figure 8: Scalability of Java-based parsers over arrays of integers in SOAP payloads . *The same test as figure 4 for parsers written in Java. We see that* Piccolo *and* XPP3 *are equivalent here.*

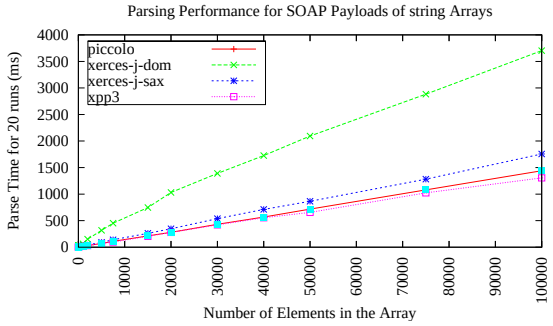

Figure 9: Scalability of Java-based parsers over arrays of strings in SOAP payloads. *Similar to the other SOAP payload tests, here the elements in the arrays are text strings, as opposed to textual representations of numbers.*

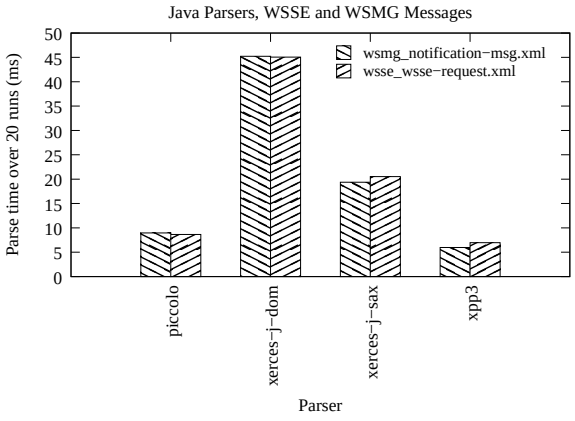

Figure 10: Java-based parsing of WSMG notification messages. *The same tests as shown in figures 5 and 6, applied to Java-based parsers. Here* Piccolo *and* XPP3 *again show much better performance than either* Xerces-J-DOM *or* Xerces-J-SAX. XPP3 *parses the message in 30% of the time it took* Xerces-J-SAX*, which has a similar programming model.*

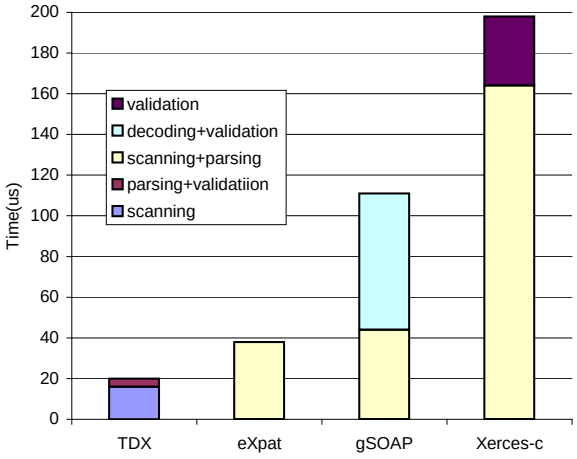

Figure 11: TDX parser vs. other C/C++-based parsers decoding a SOAP payload containing an array of strings. *TDX combines validation with parsing. It's table-driven design enables it to perform parsing and validation in less than half the time that it takes expat to parse without validation.*

HapMap), with different payloads ranging from 277KB to 4.9MB. We found that apart from Xerces-c-DOM, the rest of the parsers were able to execute the benchmark within 2 seconds. Depending on the exact performance needs of the application, one among Expat, gSOAP, and Libxml can be used for C/C++ based middleware for these applications.

Figure 3 and Figure 4 compare the performance of C/C++ based toolkits for arrays of doubles and integers respectively. The payloads consist of XML documents generated by serialization according to the SOAP protocol. The size of the arrays was varied to 100,000 elements, which we believe is the upper limit for usage via SOAP-based communication. Figure 3 shows that the Expat toolkit performs the best (744 ms, for 20 iterations of size 100,000), while the Xerces-C-DOM toolkit is orders of magnitude slower and does not scale well. For the same array sizes, due to conversion to ASCII format, the payload for array of integers is less than that of array of doubles, even though the underlying tree structure is the same. So, the parsers perform better for array of integers, as can be seen in Figure 4. In particular, gSOAP and Qt4-SAX show marked improvement.

In Figures 5 and 6, we present results when toolkits parse typical XML payloads for WS-Messenger [Huang et al. 2006] and WS-Security documents that are small in size but contain lots of namespace qualifications. The surprising results we see in these two graphs is that Qt4-SAX performs worse than Xerces-DOM for WS-Security documents. Expat, again, is the best toolkit for these kinds of XML messages, slightly outperforming gSOAP.

We present performance of Java-based parsers in Figure 7. Interestingly, XPP3 handles the smaller workflow documents better than other parsers, while Piccolo performs

best for larger sized documents used in applications for HapMap (hapmap_1797SNs.xml) and biomedical projects (molecule_1kzk_pretty.xml).

Figure 8 shows that either of Piccolo or XPP could be used in Java-based grid frameworks for handling XML payloads generated from SOAP representation of integer arrays. The results differ from the case when Piccolo and XPP handle arrays of complex types (structs) and do not have similar performance, as shown in Figure 7. The performance for arrays of strings in Figure 9 presents the same conclusions as the case with array of integers in Figure 8.

As opposed to the large size application messages used in Figure 7, for smaller size XML documents represented by WS-Notification and WS-Security documents in Figure 10, Piccolo and XPP3 have comparable performance, while the the Xerces-Java toolkit performs poorly.

TDX combines parsing and validation together, so parsing and validation cannot be separated. TDX scans and tokenizes the XML message in a separate stage, so scanning together with parsing was measured. The other parsers tested combine parsing with scanning. The result is shown in Figure 11: TDX scans, parses, and validates in much less time than it takes any other parser to even scan and parse.

5 Recommendations

- If low overhead is desired, for example when very small documents need to be processed, then the gSOAP-parser and Expat are the ideal choices for C/C++ frameworks. For Java or C# based toolkits, the Mono-reader should be used. XPP3 and Mono-DOM also have very low overheads, and should be preferred over Piccolo, Libxml2, and Qt4. The Xerces toolkit performs the worst among the toolkits we tested, and should be avoided for applications where overhead is critical.
- Xerces has a modular design and provides a great deal of flexibility for users to add their modules and mappings. As a result it is a popular choice for many applications. So, in C/C++ toolkits, if Xerces has to be used, our results show that the SAX implementation should be used, rather than the DOM model. The DOM model has a prohibitive overhead for arrays of scientific data such as doubles, floats, and integers. If performance and scalability are important, and array sizes beyond 10,000 need to be parsed, then gSOAP-parser and Expat should be employed.
- The Buffering algorithms and management of namespaces are exercised extensively for processing array of strings. Among the C/C++ parsers, we note that gSOAP and Expat are comparable. Due to the look-aside buffering scheme and optimizations for handling namespaces in gSOAP, it performs well for large array sizes. As with arrays of doubles and integers, Xerces-DOM should not be used for processing large arrays of strings.
- For Java-based frameworks, Piccolo and XPP3 have comparable performance, and out-perform the Xerces-Java implementation. Again, if Xerces-Java has to be used, then for performance, its SAX model should be used instead of DOM.
- XPP3 performs the best among Java toolkits for processing documents with complex types, such as some Kepler based workflow examples, whose sizes are a few hundred KBs. However, once the size exceeds one MB (Biomedical and Genome XML documents), Piccolo outperforms other toolkits.
- The MCS toolkit should use XPP3 or Piccolo to parse XML messages sent between the clients and the MCS server. C/C++ based clients, should use gSOAP or Expat to connect to MCS. These choices, instead of using the currently employed Axis toolkit, will significantly reduce the Web services overhead (factor of 4.8) that was reported in the MCS performance results [Singh et al. 2003].
- Pluggable modules should be incorporated into the communication medium of the reference WSRF-Java implementation, so that Axis toolkit based processing can be replaced by the efficient libraries of Piccolo or XPP3. These toolkits perform better than the other Java toolkits for WS-Notification documents, arrays of primitives and complex types, and WS-Security documents.
- The model used by the TDX parser has promise for high-performance XML processing needs, as it efficiently combines validation and scanning in one step. However, it is only applicable when the schema for the XML document to be processed is known in advance.

6 Design for Multi-core architectures

For efficient use of multi-core architectures, it is important for XML toolkits to minimize the cost of synchronization, multi-thread overhead, and use of mutex. With the currently used sequential-access formats of XML documents, if the document is not pre-scanned, the parser threads need to determine their starting points by moving a *cursor* over the document. The cursor may be controlled by one thread, or cooperatively. However, moving the cursor is a costly sequential operation that must follow XML syntax rules and handle local namespace bindings. Scanning XML is a significant component in the entire parsing process.

Amdahl's law suggests a high ratio of parsing/decoding time over XML scanning is needed to get reasonable speedups. In our earlier work on designing a table driven parser [Zhang and van Engelen 2006] (whose performance is shown in Figure 11), the breakdown in scanning, parsing, and deserializa-

tion overhead with TDX parsing is reported and compared to other XML parsers. The analysis shows that scanning can be three times slower than parsing. From Amdahl's law we see that 14% speedup can be gained with two threads, and 23% with four threads.

An issue with SAX parsing is its inherent event-based processing mode, as a result, parallel threads will not be helpful. It is possible to populate a DOM tree in parallel and gain some speedup, however, the subsequent traversal of the tree by a single thread will be slow. Another approach is to use a read-ahead thread that caches portions of the file ahead of the single-threaded parser.

To make effective use of multi-core architectures, we recommend the following: (1) pre-scanning of the document, to combine parsing with decoding, is essential to decide how to subdivide tasks to the parser threads; (2) random access should be added as a feature in XML documents (e.g. via attributes at the top level element) to aid in avoiding the cost of sequential scanning to determine starting point for each thread; (3) schema developers should specify a set of guidelines for processing instructions, in the XML document itself, to enable high performance processing under multiple threads.

7 Related Work

Several general XML benchmarking programs exist [Chilingaryan 2003; DevSphere 2000]. The XML Benchmark [Chilingaryan 2003] tests a number of parsers against arbitrary XML documents, but it does not provide a set of sample input files important for grid applications. The XML Parsing Benchmark [DevSphere 2000] tests only two different Java-based parsers, and again is not tailored to the needs of grid application developers.

The XMark project [Schmidt et al. 2001] has designed an XML benchmark suite to examine the performance of XML repositories, such as relational databases, for a wide range of queries that are typical of real-world application scenarios. This benchmark effectively compares different implementations of XML databases with queries that test specific primitives of the query processor and storage attributes. Another complementary effort is the SOAPFix [Kohlhoff and Steele 2004] project that studies applicability of SOAP for realistic business computing with data obtained from the Australian Stock Exchange.

To test the interoperability of various SOAP toolkits, the SOAP community uses a set compliant payloads for an "echo" operation of primitives, arrays of primitives, and structs [XMethods.com 2001]. Our new benchmark suite complements this effort, as it includes some of these payloads to test the performance, along with the compliance to standards. Our benchmark suite also includes many grid specific feature and application payloads.

Previously, we also developed a SOAP benchmark for grid services ourselves [Head et al. 2005]. Our new suite focuses specifically on the parsing component that applications embed, rather than the entire SOAP serialization infrastructure provided by SOAP toolkits.

8 Conclusions and Future Work

A critical component that is missing in the Grid Web services landscape is the lack of fundamental metrics and micro-benchmarks for Web services based grid middleware that can provide insights on performance limitations, bottlenecks, and opportunities for optimizations. The main thrust of this paper is the development of a comprehensive set of well-designed feature- and application-based benchmarks for Grid Web services. This framework will help evaluate and provide a road-map for the evolution of the architecture and design of grid middleware. It will also provide insights to various performance aspects of Web services based grid middleware and facilitate in its adoption by a wider scientific community. In the near future we plan to evaluate the performance of the emerging Axis2 toolkit for C++ and the role of XML toolkits for memory constrained applications such as hand-held and embedded devices.

References

ABU-GHAZALEH, N., GOVINDARAJU, M., AND LEWIS, M. J. 2004. Optimizing performance of web services with chunk-overlaying and pipelined-send. *Proceedings of the International Conference on Internet Computing (ICIC)* (June), 482–485.

ABU-GHAZALEH, N., LEWIS, M. J., AND GOVINDARAJU, M. 2004. Differential serialization for optimized soap performance. *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13)* (June), 55–64.

ABU-GHAZALEH, N., LEWIS, M. J., AND GOVINDARAJU, M. 2004. Performance of Dynamic Resizing of Message Fields for Differential Serialization of SOAP Messages. *Proceedings of the International Symposium on Web Services and Applications* (June), 783–789.

AXIS JAVA, 2002. The Apache Project. http://ws.apache.org/axis/.

BAILEY, D., BARSZCZ, E., BARTON, J., BROWNING, D., CARTER, R., DAGUM, L., FATOOHI, R., FINEBERG, S., FREDERICKSON, P., LASINSKI, T., SCHREIBER,

R., SIMON, H., VENKATAKRISHNAN, V., AND WEERATUNGA, S., 1994. The NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

BARRON, E. J., BATTISTI, D. S., BOVILLE, B. A., BRYAN, K., CARRIER, G. F., CESS, R. D., DAVIS, R. E., GHIL, M., HALL, M. M., KARL, T. R., KIEHL, J. T., MARTINSON, D. G., PARKINSON, C. L., SALTZMAN, B., AND TURCO, R. P. 1994. Global ocean-atmosphere- land system (GOALS) for predicting seasonal-to-interannual climate. National Academy Press, Washington, D.C.

BERMAN, F., FOX, G., AND HEY, T. 2003. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley.

CHILINGARYAN, S. A., 2003. XML benchmark. http://xmlbench.sourceforge.net/.

CHIU, K., GOVINDARAJU, M., AND BRAMLEY, R. 2002. Investigating the Limits of SOAP Performance for Scientific Computing. In *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing*, 246–254.

CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S., 2001. Web Services Description Language (WSDL) 1.1, March. http://www.w3.org/TR/wsdl.

CHUN, G., DAIL, H., CASANOVA, H., AND SNAVEL, A. 2004. Benchmark probes for grid assessmen. In *In Proceedings of the High-Performance Grid Computing Workshop*.

CLARK, J., 1998. The expat xml parser. http://expat.sourceforge.net/.

DEVSPHERE, 2000. The XML parsing benchmark. http://www.devsphere.com/xml/benchmark/.

FOSTER, I., AND KESSELMAN, C. 1998. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann.

FOSTER, I., KISHIMOTO, H., SAVVA, A., BERRY, D., DJAOUI, A., GRIMSHAW, A., HORN, B., MACIEL, F., SIEBENLIST, F., SUBRAMANIAM, R., TREADWELL, J., AND REICH, J. V. 2005. The open grid services architecture, version 1.0. *Global Grid Forum* (January). http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.

FRUMKIN, M., AND WIJNGAART, R. F. V. D. 2002. Nas grid benchmarks: A tool for grid space exploration. *Cluster Computing 5*, 3.

GANNON, D., KRISHNAN, S., FANG, L., KANDASWAMY, G., SIMMHAN, Y., , AND SLOMINSKI, A. 2004. On building parallel and grid applications: Component technology and distributed services. In *CLADE 2004, Challenges of Large Applications in Distributed Environments. IEEE Computer Society Press*.

GLOBUS TOOLKIT, 2002. Globus Alliance. http://www-unix.globus.org/toolkit/downloads/.

GOVINDARAJU, M., SLOMINSKI, A., CHOPPELLA, V., BRAMLEY, R., AND GANNON, D. 2000. Requirements for and Evaluation of RMI Protocols for Scientific Computing. In *Proceedings of SuperComputing 2000*.

GOVINDARAJU, M., LEWIS, M., CHIU, K., ENGELEN, R., LANG, S., AND JACKSON, K. 2005. Web services performance aspects. In *The Proceedings of GlobusWorld*.

GUDGIN, M., HADLEY, M., MENDELSOHN, N., MOREAU, J.-J., CANON, AND NIELSEN, H. F., 2003. Simple object access protocol 1.1, June. http://www.w3.org/TR/SOAP.

HAPMAP, 2003. International HapMap Project. http://www.hapmap.org/abouthapmap.html.

HAUSTEIN, S., 2000. kxml pull parser, July. http://kxml.sourceforge.net/.

HEAD, M. R., GOVINDARAJU, M., SLOMINSKI, A., LIU, P., ABU-GHAZALEH, N., VAN ENGELEN, R., CHIU, K., AND LEWIS, M. J. 2005. A benchmark suite for soap-based communication in grid web services. In *SC—05 (Supercomputing): International Conference for High Performance Computing, Networking, and Storage*. http://grid.cs.binghamton.edu/projects/soap_bench/.

HEY, T., AND LANCASTER, D. 2000. The Development of ParkBench and Performance Prediction. *In the International Journal of High Performance Computing Applications 14*, 3, 205–215.

HUANG, Y., SLOMINSKI, A., HERATH, C., AND GANNON, D. 2006. Ws-messenger: A web services based messaging system for service-oriented grid computing. In *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid06)*. http://www.extreme.indiana.edu/xgws/messenger/.

HUMPHREY, M., AND WASSON, G. 2005. Architectural foundations of wsrf.net. *International Journal of Web Services Research 2*, 2 (April-June), 83–97.

ILLINCA, F., HETU, J.-F., AND BRAMLEY, R., 1997. Simulation of 3-d mold-filling and solidification processes on distributed memory parallel architectures, November. Proceedings of International Mechanical Engineering Congress & Exposition.

KEPLER, 2003. The Kepler Project. http://www.kepler-project.org/.

KOHLHOFF, C., AND STEELE, R. 2004. Evaluating SOAP for High Performance Applications in Capital Markets.

Journal of Computer Systems, Science, and Engineering 63, 4 (July), (241–251).

KRISHNAN, S., BALDRIDGE, K., GREENBERG, J., STEARN, B., AND BHATIA, K. 2005. An end-to-end web services-based infrastructure for biomedical applications. In *In Grid 2005, 6th IEEE/ACM International Workshop on Grid Computing*.

LEAD EVENTS, 2003. Indiana University Extreme Computing Laboratory. http://www.extreme.indiana.edu/xgws/messenger/.

LUSZCZEK, P., DONGARRA, J., KOESTER, D., RABENSEIFNER, R., LUCAS, B., KEPNER, J., MCCALPIN, J., BAILEY, D., AND TAKAHASHI, D., 2005. Introduction to the HPC Challenge Benchmark Suite, March. http://icl.cs.utk.edu/hpcc/pubs/index.htm.

MCCALPIN, J. D., 1997. STREAM: Sustainable Memory Bandwidth in High Performance Computers, June. http://www.cs.virginia.edu/stream.

NAGARATNAM, N., AND HUMPHREY, M., 2003. Open grid service architecture security working group (ogsa-sec-wg). http://www.cs.virginia.edu/ humphrey/ogsa-sec-wg/.

OREN, Y., 2002. Piccolo XML Parser for Java, March. http://piccolo.sourceforge.net/.

PETITET, A., WHALEY, R. C., DONGARRA, J., AND CLEAR, A. 2004. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. Tech. rep., Innovative Computing Lab, University of Tennessee, January. http://www.netlib.org/benchmark/hpl/.

SCHMIDT, A. R., WAAS, F., KERSTEN, M. L., D. FLORESCU, I. M., CAREY, M. J., AND BUSSE, R. 2001. The xml benchmark project. Tech. rep., Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April.

SINGH, G., BHARATHI, S., CHERVENAK, A., DEELMAN, E., KESSELMAN, C., MAHOHAR, M., PAIL, S., AND PEARLMAN, L. 2003. A metadata catalog service for data intensive applications. *Proceedings of Supercomputing* (November).

SLOMINSKI, A., GOVINDARAJU, M., GANNON, D., AND BRAMLEY, R. 2001. Design of an XML based Interoperable RMI System : SoapRMI C++/Java 1.1. In *Proceedings of PDPTA*, 1661–1667.

SLOMINSKI, A., 2004. XSOAP Toolkit. http://www.extreme.indiana.edu/xgws/.

SLOMINSKI, A., 2005. Scientific workflows survey. http://www.extreme.indiana.edu/swf-survey/.

SOAPWARE.ORG, 2001. The Leading Directory for SOAP 1.1 Developers, May. http://www.soapware.org/directory/4/implementations.

SPEC, 1992. The SPEC Benchmarks. http://www.specbench.org.

TROLLTECH, 1998. Qt C++ Application Development Framework, October. http://www.trolltech.com/products/qt/.

VAN ENGELEN, R. A., AND GALLIVAN, K. 2002. The gsoap toolkit for web services and peer-to-peer computing networks. In *The Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, 128–135.

VAN ENGELEN, R., ZHANG, W., AND GOVINDARAJU, M. 2006. Toward remote object coherence with compiled object serialization for distributed computing with xml web services. In *in the proceedings of Compilers for Parallel Computing (CPC)*, 441–455.

VAN ENGELEN, R. 2003. Pushing the SOAP envelope with Web services for scientific computing. In *proceedings of the International Conference on Web Services (ICWS)*, 346–352.

VAN ENGELEN, R. 2004. Code generation techniques for developing light-weight efficient XML Web services for embedded devices. In *proceedings of 9th ACM Symposium on Applied Computing SAC 2004*.

VAN ENGELEN, R., 2004. Constructing finite state automata for high performance xml web services.

VEILLARD, D., 1998. The XML C Parser and toolkit of Gnome, February. http://xmlsoft.org/.

W3C. Canonical XML. http://www.w3.org/TR/xml-c14n.

WOO, S. C., OHARA, M., TORRIE, E., SINGH, J. P., AND GUPTA, A. 1995. The SPLASH 2 Programs: Characterization and Methodological Considerations. *In Proceedings of the 22nd International Symposium on Computer Architecture* (June).

WSRF, 2004. Web services resource framework 1.2, December. http://www.oasis-open.org/committees/wsrf/.

XERCES, 2003. Xerces XML Parser, September. http://xerces.apache.org/.

XMETHODS.COM, 2001. SOAPBuilders Interoperability Lab. http://www.xmethods.com/ilab/ .

ZHANG, W., AND VAN ENGELEN, R. 2006. TDX: a high-performance table-driven xml parser. In *The Proceedings of the ACM SouthEast Conference*, 726–731.

ZHANG, J., 2003. Virtual Token Descriptor (VTD) XML Parser. http://vtd-xml.sourceforge.net/.