An Analysis of the Costs for Reliable Messaging in Web/Grid Service Environments

Shrideep Pallickara, Geoffrey Fox, Beytullah Yildiz, Sangmi Lee Pallickara, Sima Patel and Damodar Yemme
(spallick, gcf, byildiz, leesangm, skpatel, dyemme)@indiana.edu
Community Grids Lab, Indiana University.

Abstract

As Web Services have matured the interactions that the services have between themselves have gotten increasingly complex and sophisticated. Web services can be composed easily from other services, and these services can be made to orchestrate with each other in dynamic fashion. As web services have become dominant in the Internet and Grid systems landscape, a need to ensure guaranteed delivery of interactions (encapsulated in messages) between services has become increasingly important. In this paper we describe our work with the WSRM specification. Here we describe our support of WSRM and also include an empirical evaluation of the various facets of this specification. We believe this would be very useful for system designers who intend to incorporate support for reliable messaging within their Grid applications.

1. Introduction

The emerging Web Services stack comprising XML – the lingua franca of the various standards, SOAP [1] and WSDL [2] have facilitated sophisticated interactions between services. WSDL describes message formats and message exchange patterns for services using XML. Interactions are facilitated through the exchange of SOAP messages. The use of XML throughout the Web Services stack of specifications allow interactions between services running on different platforms, containers, implemented in different languages, and over multiple transports.

It should be noted that more recently there has been an effort to factor the OGSI [3] functionality to comprise a set of independent Web Service specifications. These specifications align OGSI with the consensus emerging from the Web Services Architecture working group of the World Wide Web Consortium. The specifications that comprise the new proposed framework – the WS-Resource Framework (WSRF) [4] – can co-exist with other specifications in the Web Services area such as authentication, transactions, reliable messaging and addressing. The WSRF specification also includes WS-Notification [5] which models notifications using a topic based publish/subscribe mechanism. Similarly, the WS-GAF [6] effort in the United Kingdom provides a framework for building Grid applications using existing Web Services specifications while adhering to the principles of service-oriented architectures. The proposed solution demonstrates how issues like stateful interactions, logical resource naming, metadata, and lifetime management can be easily addressed using existing Web Services technologies.

As Web Services have matured the interactions that the services have between themselves have gotten increasingly complex and sophisticated. Web services can be composed easily from other services, and these services can be made to orchestrate with each other in dynamic fashion. Web services specifications have addressed issues such as security, trust, notifications, service descriptions, advertisements, discovery and invocations among others. These specifications can leverage, extend and interoperate with other specifications to facilitate incremental addition of features and capabilities. As web services have become dominant in the Internet and Grid systems landscape, a need to ensure guaranteed delivery of interactions (encapsulated in messages) between services has become increasingly important. This highly important and complex area was previously being addressed in the Web Services community using homegrown, proprietary, application specific solutions. It should be noted that the terms guaranteed delivery and reliable delivery tend to be used interchangeably to signify the same concept. Reliable delivery of messages is now a key component of the Web Services roadmap, with two promising, and competing, specifications in this area viz. WS-Reliability [7] from OASIS and WS-ReliableMessaging (hereafter WSRM) [8] from IBM and Microsoft among others. In this paper we provide an analysis of these specifications.

In this paper we describe our work with the WSRM specification. Here we describe our support of WSRM and also include an empirical evaluation of the various facets of this specification. We believe this would be very useful for system designers who intend to incorporate support for reliable messaging within their Grid applications. Here, we also note that we have recently finished incorporating support for the WS-Reliablity specification. The remainder of this paper is organized as follows. In section 2 we include a brief overview of the WSRM specification. In section 3 we include a description of our implementation strategy. We include empirical results from our implementation in section 4, with related work being described in section 5. Finally, in section 6 we outline our conclusions and future work.

2. WSRM

WSRM describes a protocol that facilitates the reliable delivery of messages between two web service endpoints in the presence of component, system or network failures. The specification outlines two distinct roles viz. a source and a sink. WSRM provides support for various delivery modes such as exactly-once and at least once. The delivery guarantees are valid over a group of messages, which is referred to as a sequence. Prior to a reliable exchange of messages between the source and a sink, a sequence needs to be established. Associated with this sequence is information regarding the source and sink, a unique identifier – typically a UUID, and policy information related to protocol elements and security related issues. Every message in WSRM is within the purview of sequence, within a sequence messages are assigned monotonically increasing message numbers. These message numbers allow one to keep track of problems, if any, in the intended message delivery at a sink. The message numbers facilitate the determination of out of order receipt of messages as well as message losses.

In WSRM to facilitate error corrections a sink is expected to issue acknowledgements after the receipt of a message or a set of messages. This *acknowledgement interval* is typically negotiated during the creation of a sequence. The acknowledgement from a sink may cover a single message or a group of messages within a sequence. Upon receipt of this acknowledgement a source can determine which messages might have been lost in transit and proceed to retransmit the missed messages. Another protocol constant, the *retransmission interval* governs the pro-active retransmission of messages in the event that an acknowledgement has not been received within the elapsed time. In WSRM error corrections can also be initiated at the sink; this is done through the use of negative acknowledgements which identify the message numbers that have not been received at a sink. Since message numbers increase monotonically, if message numbers `1,2,3,4` and `8` have been received at a sink, this sink can easily conclude that it has not received message numbers `5,6` and `7`.

WSRM includes exchanges for the creation and termination of sequences, and also provides for notification and exchange of errors in processing between the endpoints involved in reliable delivery. The range of errors can vary from an inability to decipher a message's content to complex errors pertaining to violations in implied agreements between the interacting entities.

2.1 Specifications leveraged by WSRM

WSRM leverages other specifications such as WS-Addressing [9] and WS-Policy [10]. WS-Addressing is a way to abstract from the underlying transport infrastructures the addressing needs of an application. WS-Addressing incorporates support for end point references (EPRs) and message information (MI) headers. EPRs standardize the format for referencing (and passing around references to) both a Web service and Web service instances as well. The MI headers standardize information pertaining to message processing related to replies, faults, actions and the relationship to prior messages. This is especially useful in cases where there would be multiple dedicated entities dealing with these different cases. Besides, the use of WS-Addressing for describing the source and the sink, WSRM also leverages fault reporting headers to report problems in the processing messages. It is expected that every message within WSRM has a unique identifier, typically a UUID, which is carried within the Message-ID information header.

WSRM uses WS-Policy to exchange information regarding protocol constants such as acknowledgement intervals, retransmission intervals, exponential backoffs etc. An entity may specify these constants for a specific sequence or for a set of sequences. WS-Policy can also be used to convey security related information.

3. Implementation of the WSRM

In our implementation of the WSRM specification we considered SOAP to be the focal point of our implementation strategy. Since all control exchanges, messages and processing logic is encapsulated within SOAP messages this approach allows the creation of a WsProcessor which deals with SOAP messages. Most WS specifications are intended to be cascaded and work in tandem with each other: having a SOAP centric approach allows us to cope with such scenarios. Within the WSRM protocol there are two distinct roles viz. source and sink. The functionality associated with these roles is dealt with in two distinct instances of the WsProcessor. The WsProcessor contains just one method viz. **`processExchange(SOAPContext, direction)`** where **`SOAPContext`** simply encapsulates the SOAPMessage and the **`direction`** specifies whether the message was received over the network or from the application.

Included below is the definition of the **`processExchange()`** method. Using the **`SOAPContext`** it is possible for an entity to retrieve the javax.xml.SOAPMessage or the equivalent EnvelopeDocument (from XMLBeans). The logic related to the processing of messages is different depending on whether the message was received from the application or network. Exceptions thrown by this method are all checked exceptions and can be trapped using appropriate try-catch blocks. Depending on type of the exception that is thrown, either an appropriate SOAP Fault is constructed and routed to the relevant location or it triggers exception related processing at the node in question. A processor decides on processing a SOAP based on three parameters

- The contents of the WSA action attribute contained within the SOAP Header.
- The presence of specific schema elements in either the Body or Header of the SOAP Message.
- If the message has been received from the application or if it was received over the network.

If the WS processor does not know how to process a certain message, it throws an **`UnknownMessageException`** an example of this scenario is a WS-Eventing source node receiving a CreateSequence response from over the network. An **`IncorrectExchangeException`** is thrown if the WsProcessor instance should not have received a specific exchange. For example if a WSRM sink node receives a wsrm:Acknowledgement it would throw that particular exception. **`MessageFlowException`** and **`ProcessingException`**s are errors caused due problems with networking and processing a message respectively. Typically, when these exceptions occur unlike the previous exceptions processing related to the message within the handler/filter chain needs to be terminated immediately. **`ProcessingException`**s occur due to processing errors related to inability to locate protocol elements in message, incorrect schemas and no values being supplied for some elements. Included below is the definition of the **`processExchange()`** method.

```
public void
processExchange(SOAPContext soapContext,
                int direction)
throws UnknownExchangeException,
       IncorrectExchangeException,
       MessageFlowException,
      ProcessingException
```

As we mentioned earlier WSRM leverages two other specifications --- WS-Addressing and WS-Policy.

3.1 Architecture

Figure 1 provides a high-level view of the architecture of our implementation (open-source and available for download from http://www.naradabrokering.org) . Here the WSRM processor leverages capabilities available within processors related to other specifications such as WS-Addressing and WS-Policy. In fact, the first set of headers that need to be processed upon receipt of SOAP messages are those related to WS-Addressing. In the case of control exchanges, the semantic intent of the SOAP message is conveyed through the wsa:Action element in WS-Addressing. Similarly, the relationship between a response and a previously issued request is captured in the wsa:RelatesTo element.

While generating responses to a targeted web service, WS-Addressing rules need to be followed in dealing with the wsa:ReferenceProperties and wsa:ReferenceParameters element contained in a service's end point reference. Similarly responses, and faults are targeted to a web service or designated intermediaries based on the information encapsulated in other WS-Addressing elements such as wsa:ReplyTo and wsa:FaultTo elements.

The WS-Policy specification is used to deal with policy issues related to sequences. An entity may specify policy elements from an entire range of sequences. A stable storage is also available at every entity to store messages. It should be noted that an entity may be a source, sink or both for reliable delivery of messages. Our implementation has been tested with two relational databases viz. MySQL and PostgreSQL.

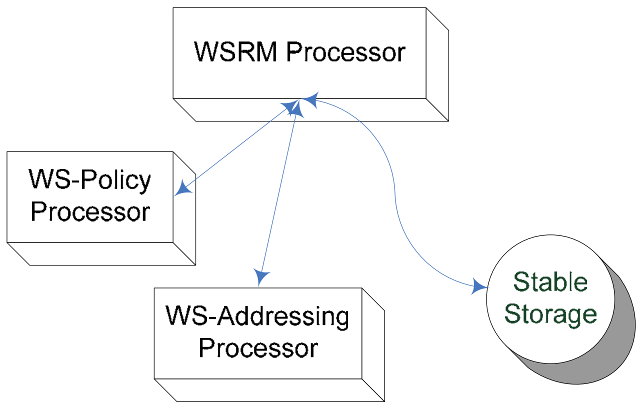

Figure 1: Overview of WSRM implementation

3.2 Rationale for the choice of XMLBeans

While implementing these specifications we were faced with an important decision regarding the choice of tool to use in processing the XML schema that aforementioned specifications conform to. In simple terms we were looking for a system that allowed us to process XML from within the Java domain. There were three main choices. First, we could use the AXIS wsdl2java compiler. Issues (in versionAxis 1.2) related to wsdl2java's support for schemas have been documented in Ref [11]. Specifically, the problems related to support for complex schema types, XML valaidation and serialization issues.

The second approach was to use the JAXB specification (a specification from Sun to deal with XML and Java data-bindings). JAXB though better than what is generated using Axis' *wsdl2java* still does not provide complete support for the XML Schema. We looked at both the JAXB reference implementation from Sun and JaxMe from Apache (which is an open source implementation of JAXB).

The final approach involves utilizing tools which focus on complete schema support. Here there were two candidates –– XMLBeans and Castor –– which provide good support for XML Schemas. We settled on XMLBeans because of two reasons. First, it is an open source effort. Though originally developed by BEA it was contributed by BEA to the Apache Software Foundation. Second, in our opinion it provides the best and most complete support for the XML schema of all the tools currently available. It allows us to validate instance documents and also facilitates simple but sophisticated navigation through XML documents. The XML generated by the corresponding Java classes is true XML which conforms to (and can be validated against) the original schema.

4. Performance Measurements

We now include performance measurements from our experiments. These experiments were performed on a 3.5 GHz Pentium IV machine with Sun's 1.4.2 Java Virtual Machine. For each measurement we performed the experiment 100 times. An outlier removal program was used to remove outliers, if any, in the result set. For each run we also tracked the memory utilization. This was done by simply recording the memory utilization prior to the invocation of a specific operation and after the invocation. In some cases this calculation resulted in a negative utilization because of garbage collection (via the Java garbage collector thread) in the intervening period. We have measured several relevant performance aspects of our implementation. We now proceed to discuss each of this in detail. A synopsis of our results is also available in a separate table (Table 1) for the reader's perusal.

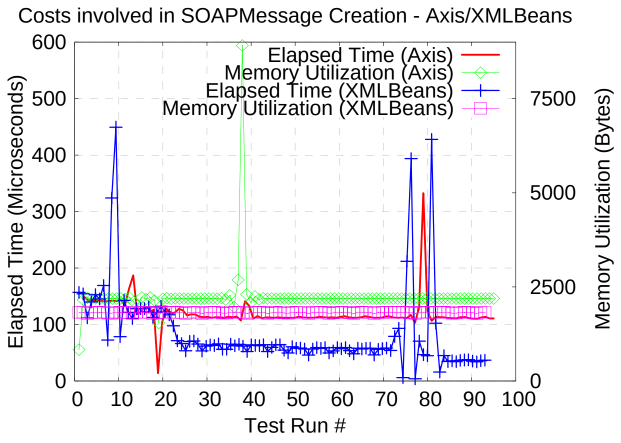

Figure 2: Costs of SOAP creation - Axis/XML Beans

Figure 1 depicts the costs involved in the creation of SOAP messages in Axis and XMLBeans. The SOAP message in Axis is an instance of javax.xml.soap.SOAPMessage while the one in XMLBeans is based on the class which is derived from the SOAP schema. Both these versions are important since though the specification has been implemented with XMLBeans, during deployments in containers it needs to be conversions need to be made to the javax.xml.soap.SOAPMessage representation.

Figure 3 depicts the costs associated with converting an XMLBeans representation into the equivalent javax.xml.soap.SOAPMessage instance. Figure 4 depicts the cost associated with converting a javax.xml.soap.SOAPMessage instance into an equivalent XML Beans SOAP Envelope instance. In these figures, ssome of the spikes coincide with the garbage collection (as evidenced by the dips in memory utilization) process, It is our conjecture that some of these spikes are related to scheduling of threads on windows XP, and the interaction of threads in Axis and XML Beans. Repeated experiments revealed the same pattern.

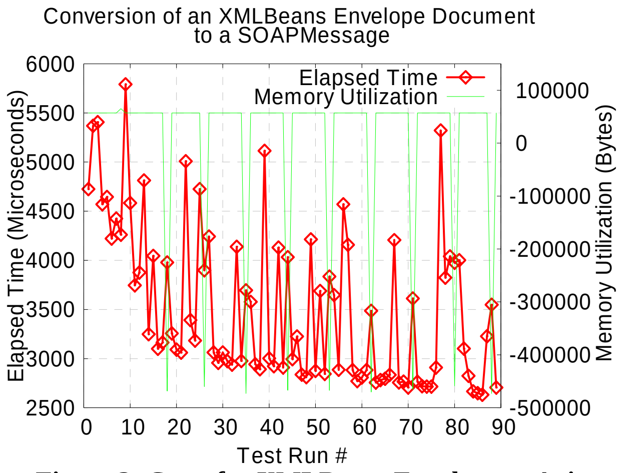

Figure 3: Costs for XMLBeans Envelope to Axis SOAP conversion

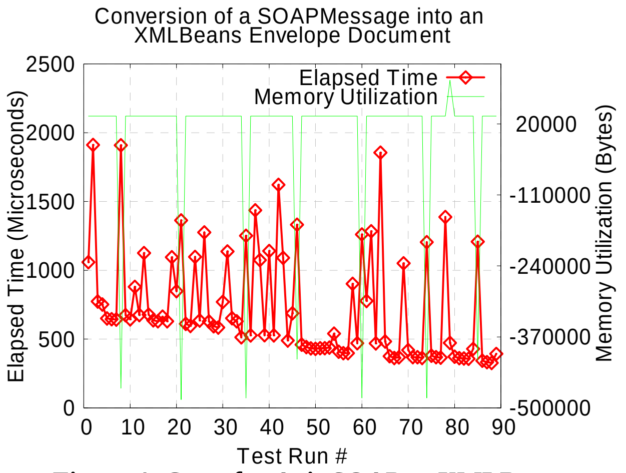

Figure 4: Costs for Axis SOAP to XMLBeans Envelope Conversion

Figure 5 depicts the costs associated with the creation of endpoint references (also referred to as EPRs) in WS-Addressing (also referred to as WSA). EPRs facilitate the targeting of web services involved in interactions and are central to all specifications that leverage WS-Addressing.

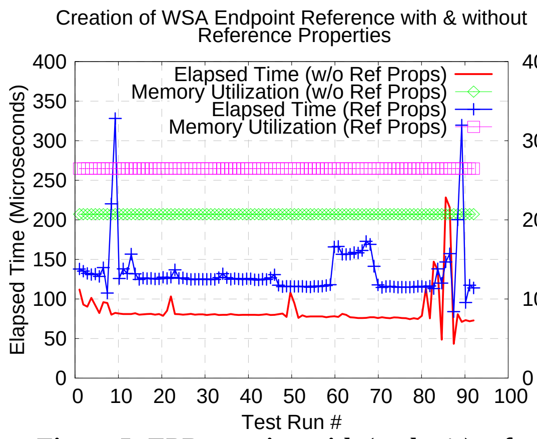

Figure 5: EPR creation with (and w/o) reference properties

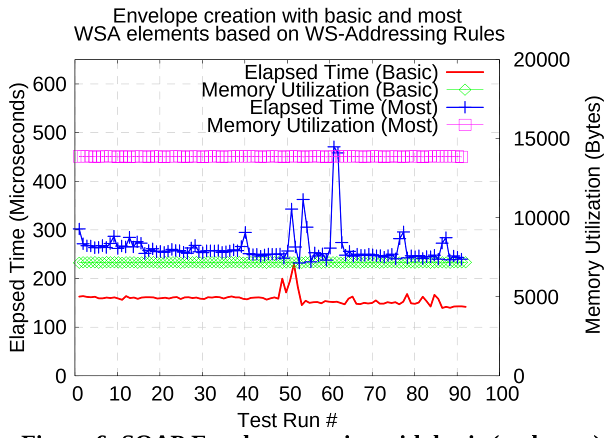

Figure 6: SOAP Envelope creation with basic (and most) WSA elements based on WSA rules

Figure 6 depicts the costs associated with creation of a SOAP envelope targeted to a specific EPR based on WS-Addressing rules under two specific cases. In the first case we include only the most basic WSA elements while in the second case we include most of the WSA elements.

Figure 7 depicts the costs associated with parsing WSA headers. This operation is typically the precursor to any processing since the WSA elements indicate not only the semantic intent (wsa:Action) but also the context (wsa:Relates, wsa:MessageID) and also where errors need to be issued to in case there are problems. The spikes in the processing time all coincide with garbage collection times (as evidenced by the dips in memory utilizations). Figure 8 depicts the cost associated with the creation of a SOAP Envelope describing a WSRM fault based on the rules outlined in both the WSRM and WS-addressing specifications. Though some of the spikes coincide with the memory utilization dips (hence garbage collection) it is not clear to us why we see these spikes in the elapsed times. Repeated tests have revealed similar patterns.

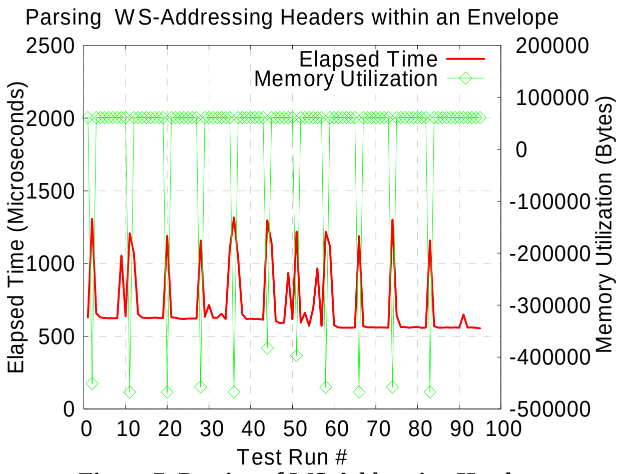

Figure 7: Parsing of WS-Addressing Headers

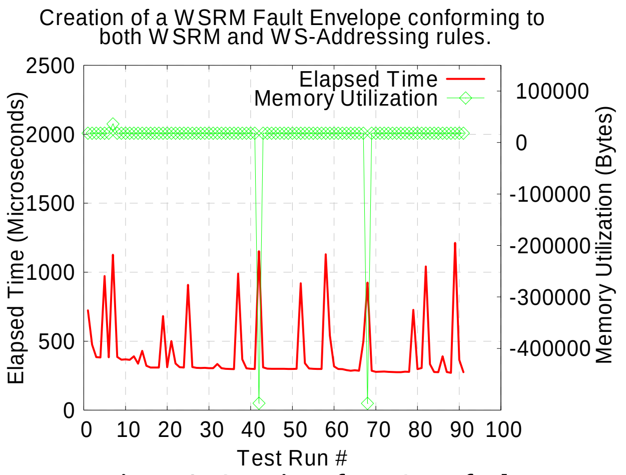

Figure 8: Creation of a WSRM fault

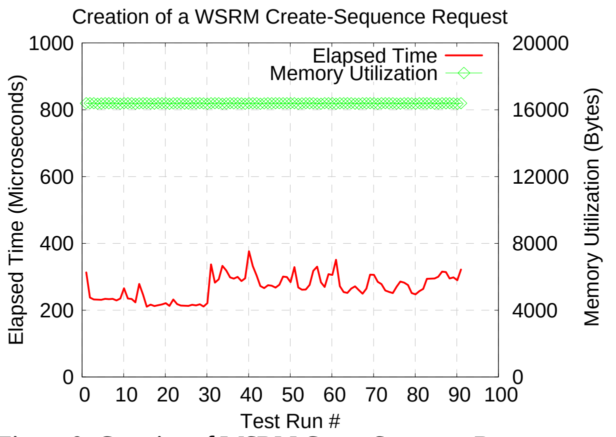

Figure 9: Creation of WSRM CreateSequence Requests

Figure 9 depicts the cost associated with the creation of WSRM CreateSequence request while Figure 10 depicts the cost associated with the generation of the corresponding response. These SOAP message envelopes are created based on the rules outlined in both the WS-Addressing and WSRM specifications.

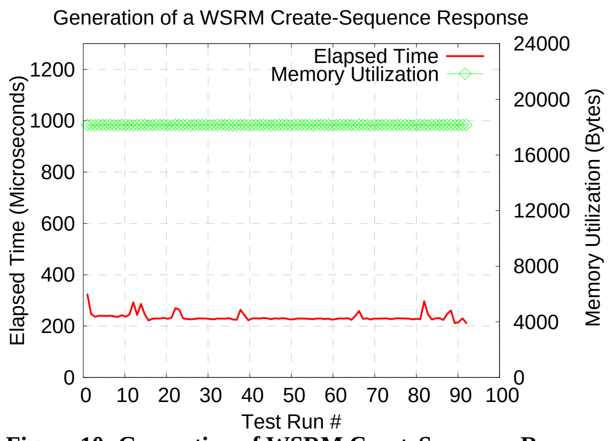

Figure 10: Generation of WSRM CreateSequence Response

Figure 11 and Figure 12 depict the costs associated with the creation of a WSRM Sequence element and the addition of this element to a SOAP Envelope. Typically, during the reliable messaging process a wsrm:Sequence element containing the appropriate sequence identifier and an incremented message number is added to the application SOAP message.

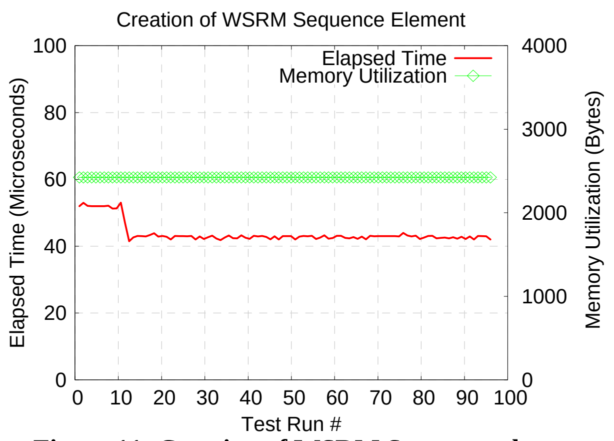

Figure 11: Creation of WSRM Sequence element

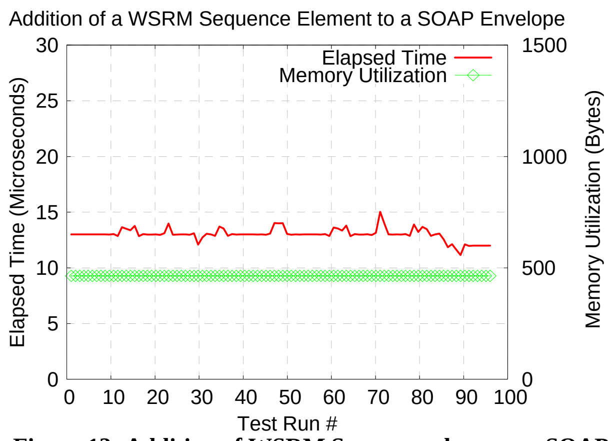

Figure 12: Addition of WSRM Sequence element to SOAP

Figure 13 depicts the costs associated with the creation of a WSRM sequence acknowledgement, which contains acknowledgements for a range of messages. Figure 14 outlines the costs involved in the creation of a WSRM Terminate Sequence request envelope based on the rules outlined in WS-Addressing and WSRM.

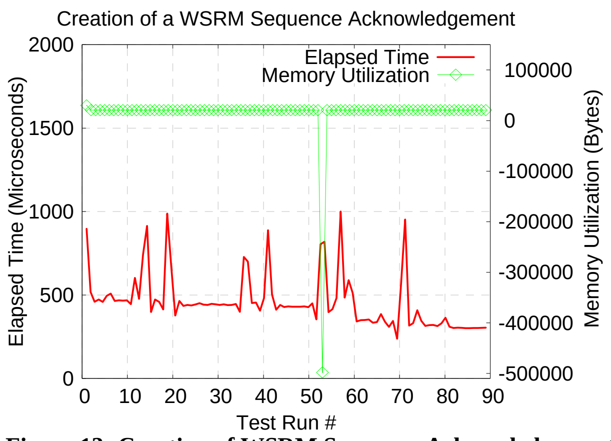

Figure 13: Creation of WSRM Sequence Acknowledgement

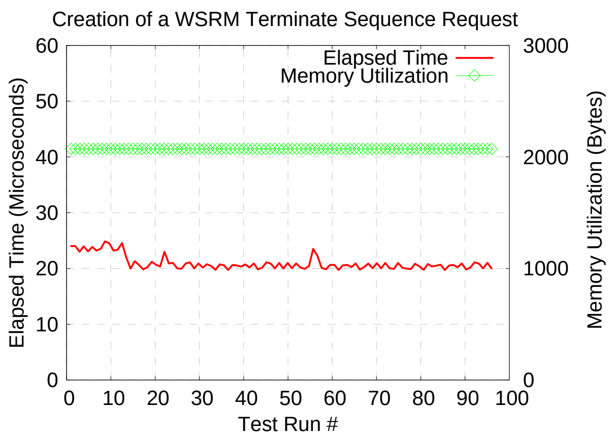

Figure 14: Creation of WSRM Terminate Sequence Request

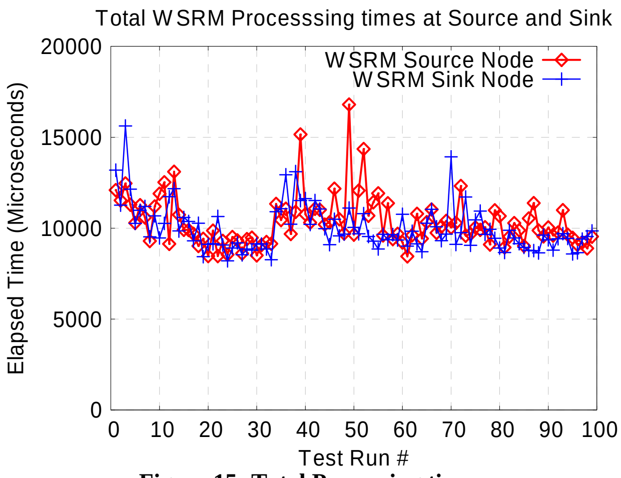

Figure 15: Total Processing times

Figure 15 depicts the total processing times at a WSRM source and sink. This includes the times for storage of message to stable storage at both source and sink. For MySQL this cost is typically between 4-6millsecond for message sizes 100B-10KB.

Table 1: Summary of results (All results in Microseconds)

Operation	Mean	Standard Deviation	Standard Error	Number of Outliers	Min Value	Max Value	Memory Utilization (Bytes)
Create an XMLBeans based Envelope Document	121.29	25.77	2.65	6	110	333	2192
Create an Axis based SOAPMessage	85.76	79.36	8.22	7	34	540	1824
Convert an EnvelopeDocument to a SOAPMessage	3503.81	758.48	80.85	12	2632	5406	57152
Convert SOAPMessage to EnvelopeDocument	730.08	392.35	41.58	11	327	1911	34424
Create a WS-Addressing EPR (Contains just a URL address)	84.61	25.61	2.67	8	72	301	2072
Create a WS-Addressing EPR (Contains WSA ReferenceProperties)	133.13	35.64	3.71	8	114	354	2648
Create a WSE SubscribeRequest	2716.98	975.79	101.73	8	1382	5418	76360
Create an Envelope targeted to a specific WSA EPR	157.98	12.19	1.27	8	140	219	7184
Create an Envelope targeted to a specific WSA EPR with most WSA message information headers	263.20	35.73	3.74	9	240	471	13880
Parse an EnvelopeDocument to retrieve Wsa Message Info Headers	711.74	231.61	23.76	5	555	1317	61024
Create a Wsrm Fault	413.80	239.17	25.07	9	271	1212	18096
Create a Wsrm SequenceRequest	268.95	37.93	3.97	9	212	374	16392
Create a Wsrm SequenceResponse	234.97	17.40	1.81	8	212	324	18160
Create a Wsrm SequenceDocument	43.8125	2.99	0.30	4	42	53	2424
Add a WsrmSequenceDocument to an existing envelope. (Contains sequence identifier and message number)	13.01	0.57	0.05	4	11	15	464
Create a WSRM SequenceAcknowledgement based on a set of message numbers	461.17	172.40	18.27	11	301	1043	20624
Create a WSRM TerminateSequence	20.95	1.30	0.13	4	20	25	2072

5. Related Work

The problem of reliable delivery [12] and ordering [13, 14] in traditional group based systems with process crashes has been extensively studied. The approaches normally have employed the primary partition model [15], which allows the system to partition under the assumption that there would be a unique partition which could make decisions on behalf of the system as a whole, without risk of contradictions arising in the other partitions and also during partition mergers. This virtual synchrony model, adopted in Isis [16], works well for problems such as propagating updates to replicated sites. Systems such as Horus [17] and Transis [18] manage minority partitions (by having variants of the virtual synchrony model) and can handle concurrent views in different partitions.

We now discuss related work in the read of publish/subscribe systems. NaradaBrokering [19, 20] facilitates delivery of events to interested entities in the presence of node and link failures. Furthermore, entities are able to retrieve any events that were issued during an entity's absence (either due to failures or an intentional disconnect). The scheme withstands failures of the entire broker network and does not require a stable storage at every entity. DACE [21] introduces a failure model, for the strongly decoupled nature of pub/sub systems. This model tolerates crash failures and partitioning, while not relying on consistent views being shared by the members. The Gryphon [22] system uses knowledge and curiosity streams to determine gaps in intended delivery sequences. This scheme requires persistent storage at every publishing site and meets the delivery guarantees as long as the intended recipient stays connected in the presence of intermediate broker and link failures

Message queuing products (MQSeries) [23] leverage the store-and-forward approach where the queues are statically pre-configured to forward messages from one queue to another. The Fault Tolerant CORBA (FT-

CORBA) [24] specification from the OMG defines interfaces, policies and services that increase reliability and dependability in CORBA applications. The fault tolerance scheme used in FT-CORBA is based on entity redundancy [25], specifically the replication of CORBA objects.

In the area of Web Services, the WS-Reliability specification from Sun and Oracle includes support for more or less the same set of capabilities as in WS-Reliable Messaging. We have implemented this WS-Reliability specification and will be releasing it to the open source community in the near future.

6. Conclusions and Future Work

In this paper we presented details about our implementation of the WS-ReliableMessaging specification. We also included empirical results from our implementation. The results demonstrate that WS-ReliableMessaging introduces acceptable overheads while ensuring the reliable delivery of SOAP messages between two web service endpoints.

References

[1] M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[2] Web Services Description Language (WSDL) 1.1 http://www.w3.org/TR/wsdl

[3] The Open Grid Services Infrastructure (OGSI). http://www.gridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf

[4] The Web Services Resource Framework (WSRF) http://www.globus.org/wsrf/

[5] Web Services Notification http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[6] Savas Parastatidis, Jim Webber, Paul Watson, Thomas Rischbeck. A Grid Application Framework based on Web Services Specifications and Practices. CS-TR-825, School of Computing Science, University of Newcastle upon Tyne, UK, Jan 2004.

[7] Web Services Reliable Messaging TC WS-Reliability. http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf

[8] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf

[9] Web Services Addressing (WSAddressing) ftp://www6.software.ibm.com/software/developer/library/wsadd200403.pdf

[10] Web Services Policy Framework (WS-Policy). IBM, BEA, Microsoft and SAP. http://www-128.ibm.com/developerworks/library/specification/ws-polfram/

[11] Kevin Gibbs, Brian D Goodman, IBM Elias Torres. Create Web services using Apache Axis and Castor. IBM Developer Works. http://www-106.ibm.com/developerworks/webservices/library/ws-castor/.

[12] Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Ithaca, NY-14853, May 1994.

[13] Kenneth Birman. A response to Cheriton and Skeen's criticism of causal and totally ordered communication. Technical Report TR 93-1390, Cornell University, Ithaca, NY 14853, October 1993.

[14] Kenneth Birman and Keith Marzullo. The role of order in distributed programs. Technical Report TR 89-1001, Cornell University, Ithaca, NY 14853, 1989.

[15] Aleta Ricciardi, Andre Schiper, and Kenneth Birman. Understanding partitions and the "no partition" assumption. In Proceedings of the Fourth Workshop on Future Trends of Distributed Systems, Lisbon, Portugal, September 1993.

[16] Kenneth Birman. Replication and Fault tolerance in the ISIS system. In Proceedings of the10th ACM Symposium on Operating Systems Principles, pages 79–86, Orcas Island, WA USA, 1985.

[17] R Renesse, K Birman, and S Maffeis. Horus: A flexible group communication system. In Communications of the ACM, volume 39(4). April 1996.

[18] D Dolev and D Malki. The Transis approach to high-availability cluster communication. In Communications of the ACM, vol 39(4). April 1996.

[19] The NaradaBrokering System http://www.naradabrokering.org

[20] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference. 2003.

[21] Romain Boichat Effective Multicast programming in Large Scale Distributed Systems. Concurrency: Practice and Experience, 2000.

[22] Sumeer Bhola, Robert E. Strom, Saurabh Bagchi, Yuanyuan Zhao, Joshua S. Auerbach: Exactly-once Delivery in a Content-based Publish-Subscribe System. DSN 2002: 7-16

[23] The IBM WebSphere MQ Family. http://www-3.ibm.com/software/integration/mqfamily/

[24] Object Management Group, Fault Tolerant CORBA Specification. OMG Document orbos/99-12-08 edition, December 1999.

[25] Object Management Group, Fault Tolerant CORBA Using Entity Redundancy RFP. OMG Document orbos/98-04-01 edition, April 1998.