In Proc. IEEE Conf. on CEC/EEE, 2008

WSDL-D: A Flexible Web Service Invocation Mechanism for Large Datasets*

Mark Wiley Aihua Wu Jianwen Su
Department of Comuter Science
UC Santa Barbara
{wileym66, ahwu, su}@cs.ucsb.edu

Abstract

WSDL web services are built around the request-reply framework, requiring service invocation to be bundled together with all relevant data in a single message. Inefficiency becomes evident as web service providers begin to offer more robust services that require massive datasets (e.g., multimedia and scientific data). Under the WSDL standards, these hefty datasets must be ported to an appropriate message format and transferred in their entirety upon each service invocation or response. Significant gains in service flexibility and performance can be made simply by separating invocation messages from their datasets. Such a separation ultimately grants service consumers the ability to pass parameter datasets from third party hosts, to maintain dataset parameters on the service provider host for use with future service invocations, and to provide datasets in a variety of different formats. In this paper, we develop a service invocation mechanism, called WSDL-D, to support this separation of service invocation from parameter datasets.

1. Introduction

The development of technologies for the Web makes it easy to share data and other resources. WSDL allows the resources to be published as (stateless) Web services. In scientific workflows, algorithms (often in the form of web services) and datasets (usually large in size) are frequently published and shared [3, 4, 5, 6], third party data management (e.g., salesforce.com) is becoming a new and interesting model for data management and IT operations for businesses. As service orientation (SOA) is being more widely adopted, more applications attempt to assemble these resources for their needs (ownerships, management overhead, etc.). Several interesting problems arise from this context; a solution would require the separation of dataset transmissions from service invocations that are beyond WSDL capabilities. In this paper, we develop an extension of WSDL to address such problems.

WSDL was designed based on the motivation to loosen coupling of interoperating software components in a simple framework. WSDL services are built around the request-reply framework that requires service invocation (response) messages to include all relevant data. For non-data-intensive (lightweight) web services (especially those utilized to enable dynamic website content), the technique is quite appropriate.

Conceptually, the request-response interface is clean and simple, but the approach is not scalable for large datasets. Specifically, inefficiency becomes evident for services that require massive datasets (e.g., product-customer databases, multimedia and scientific data). This is because that under the current WSDL and related standards, these hefty datasets must be ported to appropriate message formats and transferred in their entirety upon each service invocation or response. For example, in an e-business, management of product catalogs and customer records may be outsourced to one vendor (D) and the key functionalities of an online store front may be provided by another provider (S). If the user U wants to invoke a service by S with U's (large) dataset managed at its contractor D, the dataset has to be downloaded from D to U's site first, packaged into an XML message form, and then sent along with the service invocation message to S. Moreover, if the service of S is repeatedly invoked, the dataset has to be downloaded every time the service is invoked. This is a significant waste of resources. Such situations when the dataset and service request are located in different sites occur naturally and often. In e-science, algorithms for computing a scientific phenomenon may be available as services by one team and the services (algorithms) may require a dataset existing elsewhere (e.g., from USGS).

* Supported in part by NSF grants ISI-0415195 and CNS-0613998.

Significant improvements in service flexibility and performance stand to be made simply by separating invocation and response messages from their respective datasets. Such a separation not only grants service consumers the ability to pass parameter datasets from third party hosts, but also to maintain dataset parameters on the service provider host for use with future service invocations, and to provide datasets in a variety of different formats.

In this paper, we develop a dataset-friendly service invocation mechanism called WSDL-D ('D' for "data") to support the separation of service invocation and response from parameter datasets. In WSDL-D, input parameters (datasets) of a service invocation are not required to be sent with the invocation message, instead, an input dataset can be fetched by a service provider, sent later by the requester, or the dataset from a previous invocation request can simply be reused. Similarly, an output dataset can be pushed to the requester asynchronously, or fetched by the requester.

The paper makes the following contributions: (*i*) technical design of WSDL-D, including the methods of parameter (input/output) passing; in particular, WSDL-D extends and is compatible with WSDL, and (*ii*) experimental evaluation of a prototype implementation confirming that the potential gains in service efficiency dwarf the small amount of overhead associated with the separation of datasets from their respective invocation or response messages.

This paper is organized as follows. Section 2 illustrates application scenarios to motivate the dataset problems in WSDL. Section 3 presents details of the design of WSDL-D. Section 4 discusses the prototype implementation. Section 5 includes experimental evaluation. Section 6 concludes the paper.

2. Motivations and Problem Statement

WSDL service invocations involve two parties and the service patterns are limited to four types: one-way, request-response, solicit-response, and notification. A typical service invocation consists of two steps: (1) a request message from the *client* (requester) to the *service* (provider), and (2) a response message from the service to the client. This works well in applications when the size of data used is small. Problems arise when datasets are (very) large. In this section, we illustrate with an application scenario that WSDL service invocations are undesirable.

Fig. 1 shows an application for a retail business that stores its sales and other data in a Data Center (D). The Business Operations (O) and the Research (R) departments make replenishments orders and design strategies, resp. An outsourced service (A) is used for

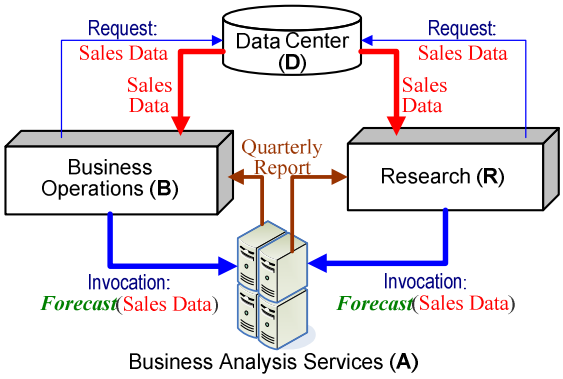

Figure 1: A business application scenario

performing data analysis tasks. In making replenishment decisions, O uses an analysis report that is generated by the *Forecast* service provided by A. Thus O downloads the *Sales Data* (large in size) from D and invokes the *Forecast* service at A. Similarly, R also needs the *Quarterly Report* for its decisions. In Fig. 1, thickness of lines roughly indicates the message size.

Three key deficiencies of WSDL web services exist in this scenario. First, a duplicate *Sale Data* transfer takes place during the invocation of *Forecast* by B: D to B and B to A. Also, *Sales Data* must be wrapped into an appropriate XML invocation message. It seems unnecessary to transfer the dataset twice. It is desirable for A to download *Sales Data* from D directly to avoid the extra transfer.

The second shortcoming of WSDL services exposed in the scenario is the wasteful disposal of datasets once the service provider has finished processing a given operation. In the scenario, R wants to get the same *Quarterly Report*. Under the WDSL limitations, R's invocation of *Forecast* resembles B's. However, *Sales Data* was already downloaded from D and sent to A via B's invocation. It is desirable to avoid R's upload. Also, the *Forecast* service does not need to run on the same input twice. It could be much more efficient to reuse that resultant data than to repeat the redundant service invocation.

The third weakness demonstrated by the scenario is the superfluous wrapping and unwrapping of datasets not readily formatted for WSDL service invocation. With WSDL services (in particular those utilizing SOAP messages), the invocation and all parameter data must be converted to an XML message. *Sales Data* could be in binary format, the file must first be encoded in base64 binary in order to invoke *Forecast*. This wrapping not only adds additional computational overhead, but also augments the size of the invocation message as base64 binary takes up more space than the original binary data. Once the invocation message reaches the web service, computational overhead is

		Push	Pull	Use
Client	*Service Input*	valid	—	—
	Service Output	—	valid	—
Server	*Service Input*	—	valid	valid
	Service output	valid	—	—

Figure 2: Valid Actor-Action Pairs

again incurred as the data must be decoded before the server can process it.

These scenarios illustrate the weaknesses of WSDL when dealing with large datasets. In order to overcome these weaknesses, one might consider the following questions to explore alternative invocation mechanisms for large input parameters (and/or output result):

- When input data isn't stored at the client but in a data center, or when output data is destined for a third party, how should the service invocation be handled?
- Can a service provider actively fetch data from the client? Similarly, can the client "pull" the result from service provider?
- Can one reuse input data in subsequent invocations?
- How can one efficiently deal with data in formats not supported by XML (e.g., a binary file, CSV)?

In the remainder of this paper, we develop a framework WSDL-D to augment WSDL with flexible service invocations so that large datasets can be dealt with effectively and efficiently.

3. WSDL-D

In this section, we present the details of WSDL-D, including parameter passing methods, communication protocols between service provider and requester, and WSDL syntax augmentation.

3.1. Overview

There are three primary features in WSDL-D. The first is the ability to pass parameter datasets hosted on third party servers. As third party data hosting solutions evolve, it becomes more and more likely that datasets used by web service consumers will not be hosted on the consumer's machine. Section 2 illustrates the problem (especially for large datasets) as the web service consumer must first acquire the dataset before being able to invoke the service.

The second improvement by WSDL-D is the ability to reuse previous datasets. By enabling storage of parameter datasets on the web service provider, WSDL-D allows multiple service operations to share the same dataset and avoid the consumer to resend the dataset. This has great potential to save time as web service consumers are no longer required to resend entire parameter datasets upon invocation of different service operations or even different invocations of the same operation.

The final function of WSDL-D is the ability to pass unaltered files as dataset parameters or results. WSDL-D web service systems have the ability to fetch or receive datasets via a variety of protocols (i.e. FTP, HTTP). Rather than having to encode file contents to be compatible with the message formats utilized by present web service systems, WSDL-D permits transferring the dataset directly to the target service.

3.2. Parameters

The metadata for describing the datasets required by an invocation specifies how those required datasets should be acquired and handled by the WSDL-D service provider, as well as how the resultant dataset should be supplied to the service consumer. The object encapsulating this extra metadata will henceforth be referred to as a WSDL-D parameter.

The first bit of metadata provided by a WSDL-D parameter specifies how the dataset parameters are to be acquired. In order to make this specification, one must provide an actor-action pair for each parameter. In WSDL-D the valid actors are *Client* (service requester) and *Server* (service provider). The valid WSDL-D parameter actions are *Push*, *Pull*, and *Use*. The *Push* action indicates that the specified actor will directly provide the corresponding dataset parameter. If the *Pull* action is specified, the actor will fetch the dataset, possibly from a third party. The *Use* action is used to indicate that the actor should utilize a stored dataset.

As an example, a web service consumer passes a WSDL-D parameter *P* with transfer method "Client-Push." This indicates that the client will provide the service with the dataset directly. Alternatively, if the consumer passed *P* with transfer method "Server-Pull," the server would fetch the dataset, from the client's site or a third party host. Fig. 2 demonstrates valid actor-action pairs for parameter passing and result return. Entries marked as "—" are invalid combinations.

There are five other pieces of required metadata for a WSDL-D parameter. The first is transfer protocol (FTP, HTTP, web service, etc.) that should be used by the specified actor when sending or receiving the given dataset. WSDL-D is not restricted to the three protocols listed; however, these protocols are common and provide the necessary mechanisms for the Push and Pull actions required by WSDL-D. The second metadata is a URL data path specifying where the

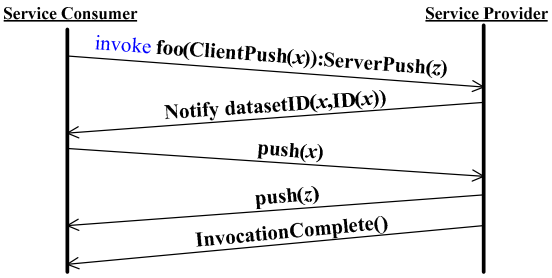

Figure 3: Invocation using ClientPush

specified actor should push or pull the specified dataset. The next two metadata are login and password credentials for accessing the specified dataset. These credentials will be used for dataset hosts that require authentication in order to access server resources. The final metadata is a time-to-live value that specifies if and for how long the given dataset should be kept at the server for invocations utilizing dataset reuse.

3.3. States and Messages

The separation of parameter and resultant datasets from invocation or result messages requires both service provider and service consumer to maintain their states in order to track the asynchronous invocation and dataset specification. The state maintained by the service provider tracks which invocations require which datasets, which datasets are currently required by an invocation or have been flagged for client reuse, which invocations are ready to execute, as well as which datasets have expired. On the client's end, the state reflects pending web service invocations, the datasets the server currently has available for reuse, as well as which datasets the server is expecting to be provided by the client.

Instead of the typical single request, single response messages in a WSDL service, a WSDL-D service invocation requires multiple messages to be sent from the web service provider to the web service consumer. In addition to the invocation and response messages, WSDL-D also requires messages for dataset ID coordination, dataset readiness notification, and service completion notification.

WSDL-D service invocations are similar to that of WSDL web services. The service consumer sends the service provider a message with an operation to invoke, the required WSDL-D parameters, as well as an additional WSDL-D parameter used to specify how the resultant dataset is to be returned to the service consumer. In response to this initial invocation message, the WSDL-D service provider returns its ID for the invocation. This ID can later be utilized by the

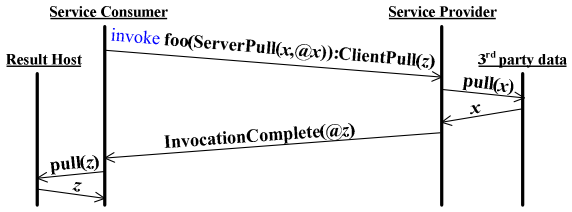

Figure 4: Invocation Using ServerPull

service consumer to poll the service provider for the corresponding invocation's state of progress.

After an invocation has been received by the WSDL-D service provider, the service provider may need to send additional information back to the service consumer. A WSDL-D service invocation with a dataset provided by Client-Push (Fig. 3), for example, creates a table entry for the dataset and returns to the consumer an ID to be used to identify that dataset when uploading (or pushing) it to the service. The same goes for persistent datasets that will be maintained on the service provider host. When a persistent dataset is specified, a dataset ID message is sent back to the invoking client so that the client may reuse the dataset on future invocations.

When a service invocation completes, the service provider notifies the invoking client of completion via an *InvocationComplete* message. Depending on how the service consumer requested the resultant dataset to be returned, the service provider may return the resultant dataset directly to the consumer or provide the necessary address and access information for the resultant dataset in the *InvocationComplete* message.

Fig. 4 illustrates the messages needed for a WSDL-D invocation utilizing the ServerPull data acquisition mechanism using a message sequence diagram. The message sequence begins with an invocation call from the service consumer. The service consumer calls the method *foo*, passing a WSDL-D parameter as input, as well as an additional WSDL-D parameter that describes how the resultant dataset should be handled. The WSDL-D input parameter "ServerPull(x,@x)" of *foo* specifies the data acquisition method ServerPull and provides the necessary data path and credential information (denoted as "@x" in the figure) needed to pull the dataset. When the service receives the invocation message, it uses the provided data path and credentials to fetch the dataset from the specified location. The service then executes and sends an *InvocationComplete* message back to the service consumer. Since the service consumer requested that the resultant dataset be returned via ClientPull (as indicated by "ClientPull(z)" in Fig. 4), the service provider provides the path and credential information ("@z") to the service consumer that may pull the resultant dataset z to complete the invocation.

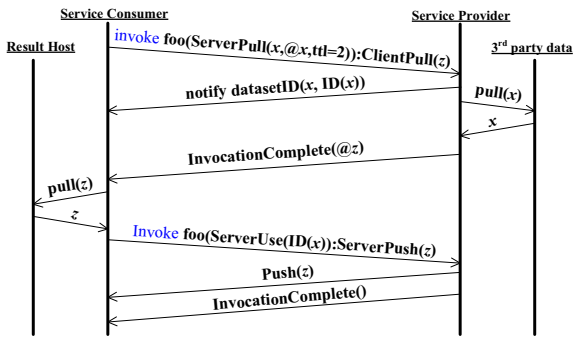

Figure 5: Invocation Using ServerUse

Fig. 3 and 5 demonstrate the message sequences for the ClientPush and ServerUse dataset acquisition techniques in WSDL-D, respectively. The messages utilized are similar to those in Fig. 4. Note that the first invocation message in Fig. 5 specifies its parameter as "ServerPull(*x*,@*x*,ttl=2)". The specification identifies the ServerPull method and indicates that the dataset will be used for two invocations ("ttl=2"). In the subsequent call, "ServerUse(ID(*x*))" specifies the reuse of the dataset *x*, where ID(*x*) is the unique identifier of the dataset *x* at the service provider and was sent back to the service customer in the "notify" message occurred earlier.

3.4. Syntax Augmentation to WSDL

We now discuss extension of WSDL to WSDL-D. In the extension, we focus on (1) compatibility with WSDL (a WSDL invocation should also be legal and recognizable to WSDL-D services), and (2) that changes to WSDL should be kept at a minimal.

In a WSDL-D web service invocation, the content or its passing method of a parameter must be specified. These are specified through attributes. Specifically, attributes include:

- *transferMethodIn*, whose possible values are: `clientPush` (dataset will be pushed to the Server), `serverPull` (dataset will be pulled by the Server), and `serverUse` (an existing data will be reused).
- *transferMethodOut*, whose possible values are: `clientPull` (dataset will be pulled by the Client), and `serverPush` (dataset will be pushed to the Client by the Server).
- *address,* address of the dataset.
- *dataID*, ID of exiting data. When *transferMethodIn* has the value `serverUse`, this ID can tell the Server which existing data should be used.
- *persistent,* indicates whether the Server should keep the dataset for future reuse. If this is set, dataset ID will be returned.

In addition to the above new attributes, there is a change to WSDL XML Schema type. Attribute *nillable* of input/output message type must be set to true. Input or output message can be parameter of XML data type (with value) or parameters with one or more of the above attributes (without value).

Along with the syntax changes, there are several constraints that define the behavior. For example, when any sub-element of input or output message is not empty, all extended attributes are ignored. When any sub-element of input or output message is empty, its *transferMethodin* or *transferMethodOut* cannot be ignored. We omit the details here but provide an example in the Appendix.

It is important when addressing the shortcomings of current WSDL-based web service systems to maintain compatibility with those systems. The WSDL-D service extension is intended to be implemented in a manner that allows the typical, request-reply oriented service invocation to be handled as it would be if the WSDL-D extension were not installed. Applications referencing a WSDL-D web service should have the option of invoking an operation via WSDL web service mechanisms, passing service parameters within the invocation message. Alternatively, applications are also able to invoke the same operation using WSDL-D parameter passing mechanisms, using separate messages to initiate service invocation and parameter dataset provision. Similarly, the web service results can be sent via standard web service messages or by WSDL-D techniques.

4. Prototype Implementation

A prototype of WSDL-D was developed on top of the .NET framework. Instead of attempting to expand the web service enabling code base of .NET or Java (as is the ultimate intent), the prototype implementation was developed as a proxy system, sitting between the service consumer and service provider. A .NET web service was deployed on both client and server hosts to act as a proxy between a normal WSDL-based service provider and service consumer. The two proxies fulfill the sate maintenance needs of the WSDL-D design as well as all dataset creation, dataset transfer, asynchronous messaging, invocation and dataset management, as well as resultant dataset creation and transfer. The following diagram illustrates how the prototype was deployed.

4.1. The Client Interface

The intermediate web service residing on the client host (the Client Interface) acts as a proxy between the

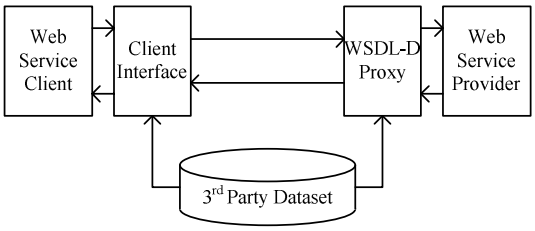

Figure 6: Architecture of the WSDL-D prototype

service-consuming client application and the WSDL-D proxy service. The responsibilities of the Client Interface include maintenance and tracking of all invocation calls and parameter datasets sent to the WSDL-D service, the handling of all asynchronous messages received from the WSDL-D service, as well as the receipt and handling of result datasets returned by the service. These responsibilities are addressed by the use of a directory table for both invocations and datasets as well as a web service operation that allows the WSDL-D service to update the state of an invocation or dataset.

Through the Client Interface, web service consumers can invoke WSDL-D services as they would any typical WSDL web service. Upon invocation, the Client Interface creates the necessary invocation and dataset directory table entries and forwards the service call to the WSDL-D proxy. The Client Interface thread for that particular invocation then enters a state of waiting. Subsequent messages from the WSDL-D proxy update the state of the invocation and dataset directory tables as the service processes. Once the service has completed, an *InvocationComplete* message is then sent back to the Client Interface, awakening the waiting thread. From the service consumer's perspective, service calls through the Client Interface appear to function as normal, blocking until the service has completed processing and produced a return result.

4.2. The WSDL-D Proxy

The second intermediate web service is a wrapper for the desired WSDL-based web service. Similar to the Client Interface, this service (the WSDL-D Proxy) maintains directory tables for tracking all incoming service invocations and parameter or resultant datasets. Each invocation is mapped to the parameter datasets it requires in order to process as well as the resultant dataset created upon service completion.

When the WSDL-D Proxy receives a service invocation, it creates a new entry in the invocation directory table as well as new dataset records for each new WSDL-D parameter in the directory table. For those parameters that are to be pushed by the client or

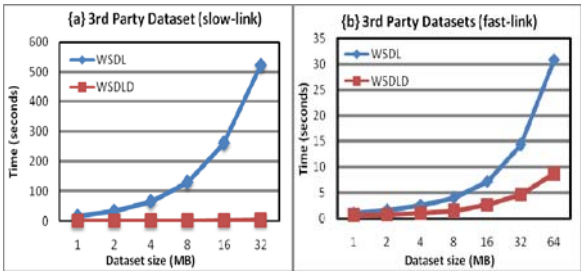

Figure 7: Service execution time (the dataset at a 3rd party host)

have been flagged for reuse, the WSDL-D proxy sends dataset identification messages back to the Client Interface proxy web service. Once the directory tables have been updated and the new or persistent dataset IDs have been sent to the Client Interface, a new thread is started to handle the WSDL-D invocation and the ID of the new invocation is returned to the Client Interface. The invocation-handling thread continues to run, fetching all necessary datasets, calling the wrapped WSDL-based target service, building a resultant dataset, and finally sending that resultant dataset back to the Client Interface in the manner specified upon invocation.

4.3. Other Prototype Considerations

While developing the prototype implementation, a few decisions regarding how various components would be implemented had to be made. For one, we had to decide where the additional functionality should be added. As mentioned in previous sections, WSDL-D is meant to extend current web service systems (i.e. Java's GlassFish, or Microsoft's .NET web services). The prototype was developed as wrapper, or proxy, web services both to be compatible with any SOAP-based web service system and to allow us to focus on creating the new functionality instead of trying to figure out how the new features would fit into a specific web service framework.

In order to achieve asynchronous messaging between the Client Interface and WSDL-D Proxy, we considered a few approaches. Socket connections were at one point considered a possible solution. The WSDL-D Proxy could make a connection to the Client Interface machine and update some sort of shared resource. This was quickly dismissed; however, as it broke from the loosely-coupled, web-like paradigm we wanted to adhere to. The second messaging technique considered was the use of WS-Eventing [2]. However, while WS-Eventing provides a protocol and messaging schema for event-driven, asynchronous communi-cations between two web services, as of this writing,

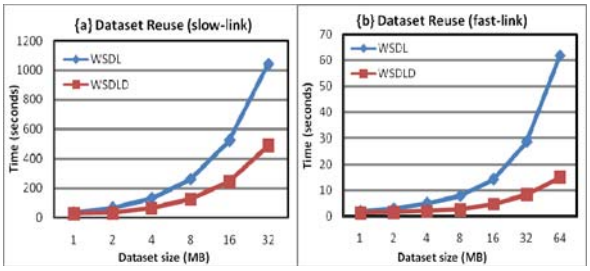

Figure 8: Service execution time
(the dataset stored at the service provider)

no standardized implementations of WS-Eventing are publicly available. As a result, the WSDL-D prototype utilizes additional service operations to enable communications between the WSDL-D Proxy and Client Interface web services.

It should be noted that while this prototype was built using .NET web services, future prototypes need not do so. The WSDL-D system is not meant to be an exclusive extension to .NET web services.

5. Experimental Evaluation

In order to gauge the performance enhancements offered by WSDL-D, the implementation described in Section 4 was deployed and tested. Overall, WSDL-D performed as expected, reducing total service runtime when compared to an analogous WSDL services in our preliminary experimental evaluation. WSDL-D invocations utilizing the ClientPush acquisition method (the most similar technique to WSDL-based services) performed on par with the conventional WSDL-based technique. Slight gains in performance were achieved by removing the encoding requirement for binary datasets, reaching roughly twenty percent gains.

5.1. Setup

Three different scenarios were tested to compare the performance of WSDL services and WSDL-D services.

The first environment was composed of a WSDL-D Proxy and an end point web service hosted by a lab machine on the `cs.ucsb.edu` domain (this host will henceforth be referred to as the WSDL-D host). The Client Interface and service-consuming client GUI were hosted by a PC connected to the Internet via cable modem with bandwidths of 5Mbs downstream and 512kbs upstream (this host will henceforth be referred to as the client host). We refer to this environment as slow-link since the service-consuming client is behind a relatively slow connection.

The second environment (fast-link) differed from the first in that the service-consuming client was deployed to another lab machine in the `cs.ucsb.edu` domain. The connection between the WSDL-D service provider and consumer in this environment was 100Mbs (about 35Mbs actual) Local Area Network.

Each experiment was run with 1, 2, 4, 8, 16, and 32 megabyte (MB) sized text datasets. A dataset of 64 MB was additionally tested in the fast-link environment for the third party and dataset reuse scenarios.

5.2. Third Party Datasets

For evaluating ServerPull from a 3rd party data host, an additional host in the `cs.ucsb.edu` domain was deployed to host datasets (the dataset host). A 100Mbs Local Area Network connection served as the link between the WSDL-D host and the dataset host.

In this scenario, the client host consumed the end point service from the WSDL-D host. For typical WSDL service invocations, the client downloads the dataset from the dataset host first, and then invokes the service with the parameter data in the invocation message. The WSDL-D invocation was made specifying ServerPull as the dataset acquisition method, i.e., the WSDL-D host pulls the dataset from the dataset host directly.

For the WSDL-based service, the time to download the dataset from the third party host was added to the time required to invoke the service to get total invocation time. This time was compared to the total time it took to run the service using WSDL-D's ServerPull mechanism. Since the WSDL-D ServerPull mechanism eliminates the extraneous transfer of the dataset from the client host to the WSDL-D host, the performance difference was quite significant.

Fig. 7(a) shows the improvement by WSDL-D when the requester has a slow connection to the service provider. For 32MB datasets, WSDL-D improved performance by nearly 30 times. Fig. 7(b) illustrates that even in environments where network connection speeds are fast and symmetric, WSDL-D techniques still provide significant reductions in the time needed for the service to acquire the necessary datasets.

5.3 Dataset Reuse

In this scenario, the client host consumed the end point service from the WSDL-D host. This time, the client made two calls to the service using the same parameter dataset. For typical WSDL service consumption, the client had to transmit the entire dataset upon each invocation of the service.

Two service invocations were also used when testing the WSDL-D approach; however, on the first call to the service the client specified ClientPush as the

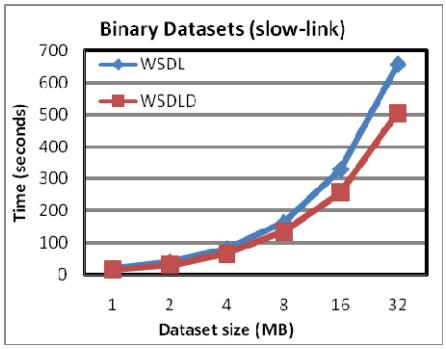

Figure 9: Invocations with binary datasets

acquisition method and signaled that the dataset should be maintained for future use. On the second WSDL-D invocation, the client specified ServerUse as the data acquisition technique and provided the ID of the already-acquired dataset to be used by the service.

For the WSDL service, the time required to invoke the service once was doubled and counted as the total time needed to make two service calls using the same dataset. For the WSDL-D experiments, the total time was calculated by adding the amount of time taken by each of the WSDL-D invocations (one using ClientPush and the other using ServerUse).

Fig. 8(a) shows that using the dataset reuse feature cuts over-all service runtime nearly in half when making two invocations. Fig. 8(b) shows that even in environments where hosts share fast connections and state maintenance becomes relatively more costly, WSDL-D is still more efficient.

5.4 Binary Datasets

In the final scenario, binary data was required as a parameter for the service invocation. For both WSDL-D and WSDL service invocations, this binary data originated on the client host. In order for the WSDL-based service invocation to succeed, the binary parameter data had to first be encoded into XML-compatible base64 binary. WSDL-D services do not require this step because the binary data is transferred as a file from the client to the server.

For the WSDL-based service, the time required for a single invocation was counted as the total time needed to make a WSDL-based service call with binary dataset inputs. For the WSDL-D experiments, the client used the ServerPull mechanism so that the server would fetch the raw binary file from the client host.

Due to the additional overhead associated with encoding and decoding binary data for transport via SOAP messages in addition to the extra space needed for base64 binary encodings, WSDL-D invocations outperform WSDL invocations by roughly 20%. These gains in performance are illustrated by Fig. 9.

6. Conclusions

WSDL provides a simple interface for web services. However, it lacks support for handling large datasets. This paper presents an extension of WSDL to allow decoupling the invocation request and invocation parameters (datasets). The extension allows obtaining datasets from a third party, reusing a prior dataset, etc. Preliminary experiments show that the performance gain out-weighs the overhead for large datasets. It is noted that the WSDL-D extension is compatible with WSDL-S [1]. It is also interesting to further evaluate WSDL-D in practical applications.

7. References

[1] R. Akkiraju, J. Farrell, et al. *Web Service Semantics—WSDL-S*, W3C Member Submission, November 2005 (http://www.w3.org/Submission/WSDL-S/)

[2] D. Box, L.F. Cabrerea, et al. *Web Services Eventing* (*WS-Eventing*), W3C, March 15, 2006 (http://www.w3.org/Submission/WS-Eventing/)

[3] Earth System Grid, http://www.earthsystemgrid.org/

[4] Federation of Earth Science Information Partners, http://www.esipfed.org

[5] B. Ludäscher and C.A. Goble. *Guest editors' introduction to the special section on scientific workflows*. *SIGMOD Record*, 34(3):3-4, 2005

[6] C. Reed. *Integrating Geospatial Standards and Standards Strategies into Business Processes*, 2004 (http://www.opengeospatial.org/pressroom/papers)