SAND2014-20565R

US NDC Modernization

SAND200X-XXXX
Unlimited Release
December 2014

US NDC Modernization: Service Oriented Architecture Study Status

Version 1.1

Benjamin R. Hamlet, Andre V. Encarnacao, James M. Harris, and Christopher J. Young

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

SAND200X-XXXX
Unlimited Release
December 2014

US NDC Modernization: Service Oriented Architecture Study Status

Benjamin R. Hamlet, Andre V. Encarnacao, James M. Harris, and Christopher J. Young
Next Generation Monitoring Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0401

Abstract

This report is a progress update on the USNDC Modernization Service Oriented Architecture (SOA) study describing results from Inception Iteration 1, which occurred between October 2012 and March 2013. The goals during this phase are 1) discovering components of the system that have potential service implementations, 2) identifying applicable SOA patterns for data access, service interfaces, and service orchestration/choreography, and 3) understanding performance tradeoffs for various SOA patterns

REVISIONS

Version	Date	Author/Team	Revision Description	Authorized by
1.0	3/31/2013	US NDC Modernization Team	Initial Release	M. Harris
1.1	12/19/2014	IDC Reengineering Team	IDC Release	M. Harris

CONTENTS

FIGURES

TABLES

NOMENCLATURE

DOE	Department of Energy
ESB	Enterprise Service Bus
IDC	International Data Center
SNL	Sandia National Laboratories
SOA	Service Oriented Architecture
US NDC	United States National Data Center
XML	Extensible Markup Language

1. INTRODUCTION

This report describes progress for the Service Oriented Architecture (SOA) study completed during US NDC Modernization Inception Iteration 1 (October 2012 – March 2013). Goals during this phase are:

1. Discover components of the System that have potential service implementations
2. Identify applicable SOA patterns for data access, service interfaces, and service orchestration/choreography
3. Understand performance tradeoffs for various SOA patterns

The first item is covered by a service identification exercise based on ranking system components using qualities typically found in services. The second item is designed to provide comparison points between the current system architecture and potential future architectures. The third item provides metrics for these comparisons and is based on understanding performance implications of accessing services using eight different potential architectures.

2. PRELIMINARY SERVICE IDENTIFICATION

The first step of the SOA study involves service selection. The goal of this exercise is to identify service selection criteria, understand service selection techniques, and discover which aspects of the system have potential to be implemented as services. This is only a preliminary service identification used to help understand system scope. Final service selection will occur at a later date if SOA is used in the modernized system architecture.

All candidate services must have at least the following qualities:

- Reusability: a service must be useful in more than one context or to more than one user.
- Composability: a service must be a useful component of a larger business need rather than serving an isolated purpose

Any component that is not reusable or which serves an isolated purpose on its own should either be a standalone application, a subcomponent of a service, or accessed through a library. Candidate services are rated according to the following four qualities:

1. *Granularity* – measures the ratio of how much computation is performed in a single call to a service to its invocation overhead.

   Assignments: coarse, medium, fine

   Discussion: A fine-grained procedure has a low ratio, does not perform much computation in a single call, and is potentially limited in performance by high communication overhead. A coarse-grained procedure has a high ratio, performs relatively large amounts of computation in a single call, and has performance less coupled to communication overhead.

2. *Autonomy* – specifies the likeliness of a potential service's results being used or meeting a defined need on their own versus use as an intermediate step in a larger process.

   Assignments: low, medium, high

   Discussion: A basic signal processing function has low autonomy if it is always used as an intermediate step to help solve higher order problems in areas such as detection, location, or association. Detection, location, and association algorithms themselves likely have high autonomy.

3. *Modularity* – identifies ability to describe a component with a well-defined interface, allowing for the consistent use of multiple different implementations.

   Assignments: low, medium, high

   Discussion: Standardized interfaces are necessary components of re-implementable services.

4. *Volume* – indicates how often a component is used during system operations.

   Assignments: low, medium, high

   Discussion: Volume count assignments do not automatically preclude a component's use as a service. Since high volume can result in high aggregate communication costs, it can be a driving factor in implementing a component as a library rather than as a service. In general, volume assignments are based on relative invocation counts compared to other components in the system. A medium volume assignment is given to an operation that is called routinely, perhaps for every piece of data processed by the system. Low and high volume operations are called significantly less or more often.

Selecting which components are ultimately implemented as services is a tradeoff of the granularity, autonomy, modularity, and invocation volume of the component. A component performing a specific, fine-grained task might be a good service candidate if it has high independence, whereas a coarse-grained task that has low independence or modularity might be better implemented as a subcomponent.

Components isolated from automatic pipeline or interactive processing operations tend to be identified as poor candidates for services as they are unlikely to meet either the reusability or composability criteria.

Several system components have been identified as not meeting the service selection criteria and have been eliminated from further consideration as services:

1. Analyst tools are not services but graphical applications that access services.
2. Unclassified to classified data transfer requires a highly specialized, secure implementation.

3. System recovery is a rarely initiated process independent of standard mission processing and operations. It may access services.

Please see Appendix A: Service Selection for system component ranking and identification as services, libraries, or applications.

3. SOA PIPELINE PATTERNS

Several key architectural decision points for using a SOA on the modernized system concern information flow through the processing pipeline structure, database access, and typical service interfaces. Two alternatives in each of these areas are discussed in this section, and performance for each of the alternatives is discussed in the following section.

Automatic pipeline processing uses separate processing components which each take a particular set of inputs, perform processing on those inputs to create outputs, and then pass those outputs to the next processing component for further refinement. Two possibilities for passing data between processing components in a SOA are direct communication between services, where one service communicates results to the next service, and central controller based communication where a central controller acts as a hub handling all messaging. In the latter option, the central controller receives outputs from services and then passes them to the next service in the pipeline. This option supports a flexible pipeline by allowing the central controller to make decisions about which services to call, but this flexibility comes at the cost of increased messaging loads.

The current generation of software tends to have each system component directly access data from the backing database. This can be efficient, but involves several important tradeoffs: system components are aware of the physical data model, preventing components and the data model from varying independently of one another, and data access logic is distributed throughout the codebase, leading to potential code duplication and requiring individual developers to be familiar with making database queries. Migrating software to use a Common Object Interface (COI) could mitigate some of these problems by providing a central access point for database interactions. Two possible patterns are to allow services direct access to a COI or to use a data access service.

Database centric communication is typical of many components in the current system. If one component needs to send data to another component, the first component writes that data to the database, alerts the second component that data is available, and the second component reads that information from the database. Using this type of communication in a SOA results in light service interfaces as most of the information used by a service is passed through the database and not through the service interface. An alternative approach is to implement services using rich interfaces that pass data directly through interface parameters rather than the database.

3.1 Pattern Summary

The SOA patterns considered in this study are summarized below, organized by pattern decision points. Each decision point offers two alternatives, yielding a total of eight combinations. Performance for each of these combinations is analyzed in the Performance Testing section.

1. Understand relationship between data flow and control flow
    a. Data marshaling and unmarshaling requirements for direct service-to-service or client-service communication
    b. Data marshaling and unmarshaling requirements using a centralized controller to broker communication (service-controller-service or client-controller-service communication)
2. Understand relative costs associated with potential data access architectures
    a. Data accessed directly from COI
    b. Data accessed using COI service
3. Understand relative costs associated with potential SOA interface designs
    a. Parameters passed directly to service
    b. Parameters passed through data store; interface parameters describe database queries

Each of these decision points is discussed in more detail below.

3.2 Information Flow

Two general types of information flows exist in the system: Control flow and Data flow. Control flow refers to how service invocation messages are passed and data flow refers to how service inputs and outputs are passed. Combining services to solve problems involves passing control and data between services. Two primary methods of passing information are:

1. Centralized control logic: a central controller component brokers communication between services. All services receive input from the controller and pass results back to the controller. Services are completely decoupled from one another.

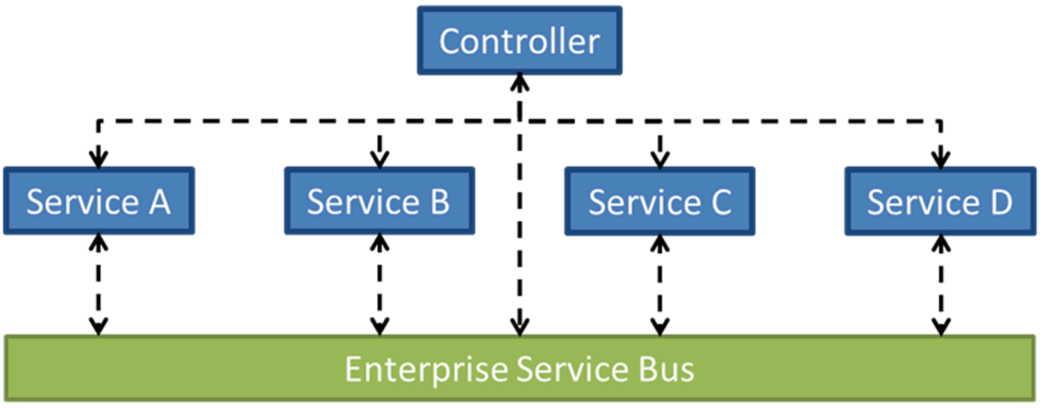

Figure 1. Centralized controller component

2. Distributed control logic: service pass messages directly to other services without using a central controller component.

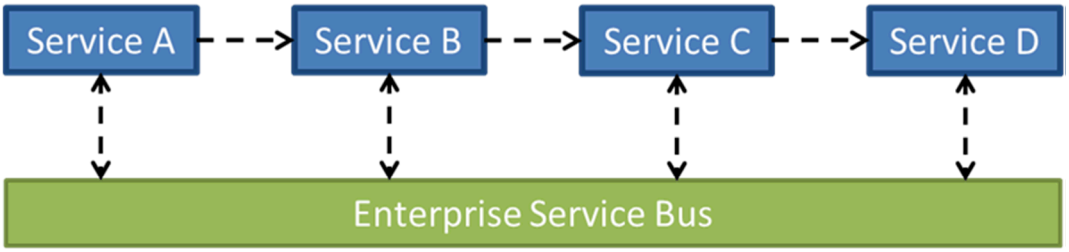

Figure 2. Distributed control logic (no centralized controller component)

Of concern to SOA performance is the amount of marshaling and unmarshaling required to pass data between services. Consider control flowing from Service A to Service B. There are several options for how Service A's outputs are mapped to Service B's inputs:

1. Forward: Service A's outputs are passed directly to Service B
2. Subset: Some of Service A's outputs are passed to Service B
3. Append: All or some of Service A's outputs plus additional data are passed to Service B
4. No dependence: Service A does not produce any inputs required for Service B

A trade study of ESB and messaging implementations is required to understand how data marshaling occurs in these cases both when services are invoked on the ESB as higher-order services and when they are invoked in client code as applications. Only the first option is studied in the performance tests.

3.3 Data Access Methods

Two design possibilities combining a SOA with a COI were investigated in this performance study:

1. Data access via a COI, i.e. abstracted from data storage -- Services perform data storage and retrieval using a COI. The COI handles all interactions with the backing data store. Abstracted data access allows the data store to vary independently of data access.

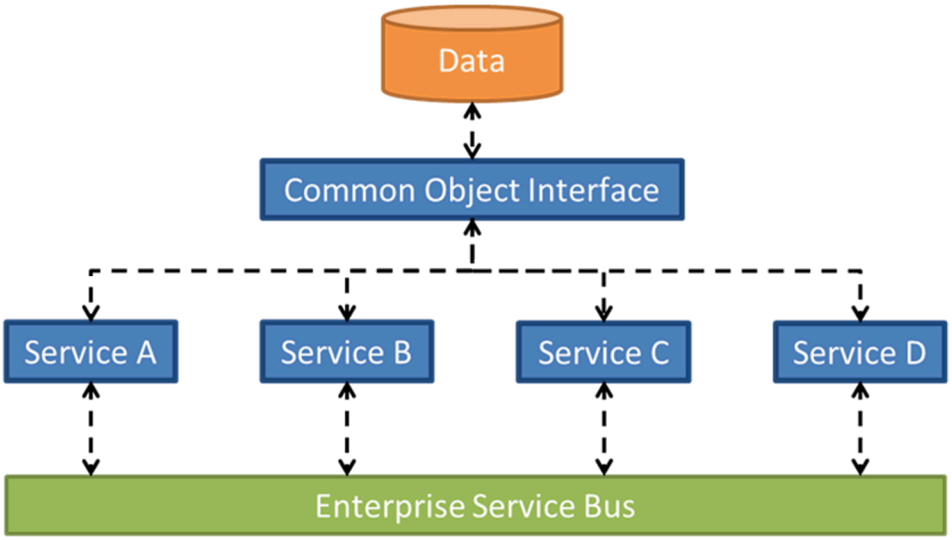

Figure 3. Data is accessed directly through a COI

2. Data access via a service – This design extends abstracted data access by requiring services to store and retrieve data through a data access service. This decouples services from data access, allowing the data access service to vary independently of the processing services.

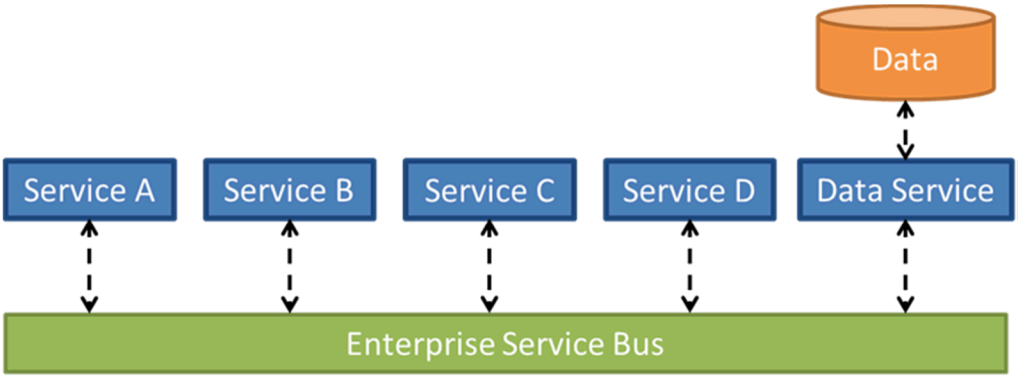

Figure 4. Data is accessed through a Data Access Service

3.4 SOA Interface Standards

We investigated two primary options for service interface design in the system:

1. *Light interfaces* follow the current system design, where information is passed primarily through the database. In this option, simple messages are sent between services declaring where in the database the invoked service may find its input parameters. Service interfaces are decoupled from one another by passing parameters through the backing data store, but services rely on a contract external to the interfaces defining what parameters are expected in the data store.

2. *Rich interfaces* provide more service separation from the surrounding environment by defining all input parameters as part of the interface, freeing services from having to negotiate a common parameter area in the data store. Services can optionally access the data store for additional configuration parameters that are either invariant from one invocation to the next or are implementation specific parameters hidden from clients.

4. PERFORMANCE TESTING

Messaging performance testing provides data points for use in making decisions regarding service selection and SOA architectural patterns. In particular, messaging costs associated with services are required to understand service granularity and invocation volume, as discussed in the *Preliminary Service Identification* section, and are a comparison point for tradeoffs involved in using the various architectural patterns discussed in the *SOA Pipeline Patterns* section.

Canonical SOA performance testing tasks have been selected to establish the performance costs associated with using a Service Oriented Architecture as the basis of an explosion monitoring system similar to those operated by the USNDC and IDC. Mock services implementing a variety of interfaces with similar types, amounts, and sizes of parameters to potential services are used. The mock services do not perform actual computations, allowing performance tests to study the overhead associated with various architectures.

Messaging costs for data flow through a sample processing pipeline are also computed, allowing for higher order performance comparisons.

SOA architectures often employ an Enterprise Service Bus (ESB) as a central component involved in message queuing, routing, translation, service registration, and service orchestration. As a central hub brokering client-service and service-service communication, ESB performance is a factor in overall system performance. ESB performance is not considered in this report. Existing ESB performance test results are available [1] and are a potential topic for future study.

4.1 Message Formats

The message content and formatting we used follow openly available schema and encoding mechanisms. The selected schemas are currently supported by the seismological community and message encoding products are supported by common enterprise computing products.

Two message formats were used:

1. Human readable messages with data in CSS 3.0 XML format
2. Binary versions of CSS 3.0 XML format messages using the Fast Infoset [2] format

Both message formats carry the same information and all messages conform to the same schemas. The same tests were run using each format to study relative performance. Fast Infoset was used as the binary XML format because it is a stable product of the Glassfish community. It operates with JAXB, allowing significant reuse of parsing and writing code with that used for the human readable messages.

4.2 Service Interfaces: Message Content

Mock services use three classes of information:

1. Alphanumerics
   - Event data: long, double, and string values as stored in the *origin, assoc,* and *arrival* tables.
     READ: Event data is generated from the database by querying a specified time period
     WRITE: Event data is written to the database
   - Arrival data: long, double, and string values as stored in the *arrival* table.
     READ: Arrival data is generated from the database by querying a specified time period
     WRITE: Arrival data is written to the database
   - Origin data: long, double, and string values as stored in the *origin* table.
     READ: Origin data is generated from the database by querying a specified time period
     WRITE: Origin data is written to the database
   - Site data: long, double, and string values as stored in the *site* table
     READ: Site data is generated from the database for a specific geographic area
     WRITE: Site data is written to the database
   - Bulletin data: long, double, and string values as stored in the *origin* and *origerr* tables
     READ: Bulletin data is generated from the database by querying a specified time period
     WRITE: Bulletin data is written to the database
   - String data: Java string object
     READ: String data is randomly generated from uppercase characters in the English alphabet
     WRITE: String data is not written

2. Waveforms
   - Arrays of 8-byte doubles with long, double, and string metadata from the *wfdisc* table
     READ: Waveform data is generated by querying *wfdisc* rows from the database for a specified station, channel, and time period and then using the *wfdisc* to retrieve the required *.w waveform files from disk.
     WRITE: Waveform data is written as a *.w waveform to disk

3. Imagery
   - java.awt.Image objects. Images are marshaled and un-marshaled in a base64 encoding.
     READ: Image data is read from GIF and PNG images stored on disk
     WRITE: Image data is not written

These basic data types are building blocks for mock services designed to input and output various combinations and quantities of data representing operations performed in automatic and

interactive monitoring software. Rather than measuring performance of all anticipated system messages, the study covers messaging performance for canonical families of messages. Performance for specific operations can then be estimated using the cost of a similarly sized message using a similar balance of data types. The tables below show some possible mappings of message types to operations performed on the system. These tables are not exhaustive, but were used to verify performance metrics were gathered for a range of messages spanning system usage.

1. Perform query

Request	Response	Examples
Small	Small	Lookup station information, lookup phase identification, lookup event location
Small	Medium-large	Get map of specified region, get specified channel waveform, get event bulletin

2. Perform monitoring operation

Request	Response	Examples
Medium-large	Small	Locate event (various number of associated arrivals), identify event
Medium-large	Medium-large	Associate signal detections, identify phases, compare bulletins, compute batch of earth model prediction, locate group of events

3. Perform operation

Request	Response	Examples
Small	None/ Confirmation	Log message, pipeline control/invoke services
Small	Small	Time/date functions, geometric calculations, geographic calculations, magnitude and other derived alphanumeric calculations

4. Perform waveform operations

Request	Response	Examples
Large	Small	Waveform quality, signal detection, feature extraction
Large	Medium	Waveform based event processing (outputs a bulletin)
Large	Large	Filter waveform, beam forming, convolution, deconvolution, correlation, compute PSD, compute spectrogram, compute fk

4.3 Sample Pipeline

Assembling services into a sample monitoring pipeline consolidates individual messaging costs and allows higher order comparisons of various message formats, data access methods, pipeline control, and types of service interfaces. Our sample pipeline consists of four stages:

1. Data acquisition: individually receive 1 minute intervals of waveform data from a network of 50 one channel stations
2. Signal Processing: immediately signal process waveforms to produce arrivals

3. Event Processing: collects 45 minutes of arrivals then run network processing to form events
4. Write Bulletin: immediately store events and associated detections to a database

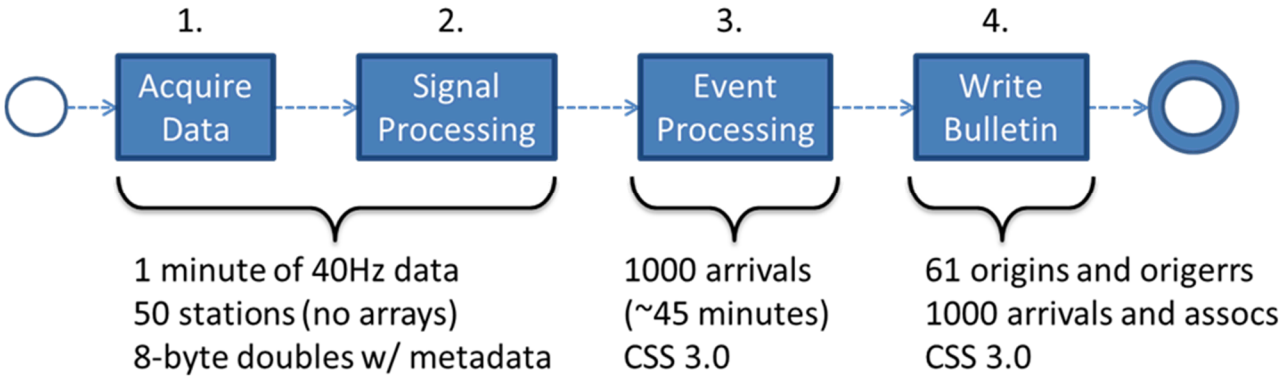

Figure 5. Sample pipeline

Arrival and origin counts are based on queries of the IDCX database, which records all of the automatic signal detections made by the system before analyst review. Our analysis of the IDCX database indicates an average of about 1,000 automatic arrivals and 61 automatically built events in 45 minutes.

Performance for eight versions of this sample pipeline are considered in order to account for all combinations of information flow (central controller or no central controller), data access (direct data access or data access through a data service), and service interface standards (light or rich interfaces) discussed above.

Since pipeline performance is based on individual messaging performance, extending results for different pipeline structures or types of messages is straightforward. For instance, an obvious advantage of using light interfaces is that intermediate processing results are immediately stored to the database.
This can also be achieved using rich interfaces by writing processing results to the database independent of normal pipeline control and dataflow, introducing additional system load. Quantifying this load only requires adding the database writing costs into the output cost of each processing stage.

4.4 Results and Discussion

Message overhead results for individual messages are tabulated in *Appendix B: Messaging Performance Results* and performance results for the sample pipeline are in *Appendix C: Sample Pipeline Performance Results*. This is a rich set of information that we hope will serve as a useful reference for design of various types of service-based systems.

We identified the following as the most important observations:

- Database centric waveform operations dominate overall costs.
- As the light interface standard passes waveforms through the database, the light interface based pipelines are an order of magnitude slower than the rich interface based pipelines.
- Data persistence is a side effect of using database centric communication. Persisting intermediate results is a requirement of the next generation system but is not considered by the rich interface pipeline. Whether or not the system separates persistence from control flow, persistence costs for the rich interface based pipeline are easily obtained by adding the costs of PUT operations from corresponding pipeline stages using light interfaces.
- Using a central controller component with rich interfaces results in roughly twice the messaging overhead compared to direct service-to-service communication since information must be packaged at one service, sent to the controller, unpackaged, analyzed, then repacked and sent to the receiver. In direct service-to-service communication the redundant marshaling and unmarshaling at the central controller is eliminated. It is possible that messages could be constructed such that the central controller does not need to unmarshal and remarshal the entire message, so costs reported here represent the worst case.
- Using a central controller component with light interfaces does not have any impact for this pipeline as the cost associated with a string of 10,000 characters or less is negligible.
- Using a data access service with rich interfaces results in < 1% performance penalty for this pipeline since data flows directly into services.
- Using a data access service with light interfaces results in 14.9% performance penalty for this pipeline since service communication occurs through the database.

While it is premature to make architectural decisions based on this study, the performance results indicate service oriented architectures and data access services are feasible on this system. There are alternatives to consider for each type of architecture, such as handling persistence in rich interfaces prior to writing final results or handling waveforms separate from alphanumeric data when using light interfaces to avoid expensive file system operations. Additionally, a careful study of implications to meeting system requirements is needed before deciding on a final architecture.

5. FUTURE WORK

The first phase of the US NDC Modernization SOA Study focused on 1) using a service selection exercise to understand if the system has components that can be beneficially implemented as services 2) identifying key SOA architectural tradeoffs, 3) gathering performance metrics for a variety of messages typical of system operations, and 4) formulating a simple processing pipeline implemented using the different architectural patterns previously identified and combining individual messaging costs to study performance tradeoffs of the patterns. Further work on the SOA Study will focus on implementing a proof of concept SOA system performing seismic signal and event processing. This will allow us to compare projected performance for various system designs from this study with actual results.

6. REFERENCES

1. ESB Performance. AdroitLogic Private Ltd., 2012 (http://esbperformance.org).

2. Fast Infoset. Java.net, 2012 (http://fi.java.net/).

APPENDIX A: SERVICE SELECTION RESULTS

Table 1. Service Selection

C:Coarse F:Fine H:High L:Low M:Medium -:any valid option
Service column: A:Application L:Library S:Service. The primary option appears first when multiple options are suggested.

Potential Service	Granularity	Autonomy	Modularity	Volume	Service	Notes and Questions
Data Acquisition						
• USAEDS waveforms	C C	H H	H H	M M	A A	
• Standard external network waveforms (IDC and other)	C	H	M	L	A	
• Non-standard external network waveforms	M F	H H	M M	L L	S A	
• Bulletins	-	H	L	L	A	
• Event Messages						
• Non-waveform misc. external resources						
Data access						- Modularity rating assumes a Common Object Interface and is medium due to the difficulty in defining a fully abstracted interface. - Primary concern is data access speed.
• Waveforms	-	L	M	M	S/L	
• Alphanumeric	-	L	M	H	S/L	
System logging	F	H	H	H	L/S	- Handles receiving and storing log messages. - Log type is general and includes at least software processes, hardware, data acquisition, security, and pipeline health.
Analyst work assignment creation	F	H	H	L	A	Communication overhead is immaterial for infrequently run processes.
Analyst work assignment distribution	F	H	H	L	A	Communication overhead is immaterial for infrequently run processes.
Monitoring Network SOH	M	H	H	L	S	Includes acquisition SOH
Hardware SOH	M	H	H	L	S	Disk, memory, processor, network loads and outages.
Process monitoring	M	H	H	L	S	

C:Coarse F:Fine H:High L:Low M:Medium -:any valid option Service column: A:Application L:Library S:Service. The primary option appears first when multiple options are suggested.						
Pipeline controller	C	H	H	M	S	Includes automatically launching system components and services to run the automatic pipeline.
Event location	C	H	H	H	S	Covers all technologies used for location.
Signal association	C	H	H	M	S	
Event QC	M	M	H	M	S	Might be called by an association algorithm to measure quality of proposed events.
Event identification • Individual discriminants • Full event identification	 F C	 H H	 H H	 M M	 S/L S	
Report generation • Internal reports • External reports	 M M	 H H	 L M		 N N	These are not services because they are not *Composable.*
Publishing • Reports • Bulletins	 M M	 H H	 H H	 L L	 S S	
Bulletin comparison	C	H	H	L	S	
Performance monitoring • Network capability • Station capability • Event comparisons (detecting stations, picks, residuals, location, etc.) • Station ambient noise • Capability estimation	 M-C M-C M-C C C	 H H H H H	 H H H H H	 L L L L L	 S S S S S	- Monitors observed system performance - Covers comparisons between current and historical capability, analyst and automatic performance, and analyst to analyst performance. - Capability estimation covers simulated/predicted system performance and refers to NetSim/NetCAP/NetMOD - Simulated/predicted results can be compared to historical results
GIS: produce maps	M	H	H	L	S	Does not include the analyst map
Data forwarding	-	H	H	M	S	- Covers forwarding between the OPS, ALT, SUS, and Training subsystems as well as to external sources like the IDC or national labs. - Transfer confirmation is included in the service.

C:Coarse F:Fine H:High L:Low M:Medium -:any valid option Service column: A:Application L:Library S:Service. The primary option appears first when multiple options are suggested.						
Data backup	C	H	H	M	A	- Could portions be implemented as a type of data forwarding?
Geometric operations • Point-polygon intersection • Ellipse-polygon intersection • Ellipse-ellipse intersection	F	L	H	L	L/S	
Geographic operations • Distance • Azimuth • Azimuthal gap • Great circles	F	L	H	M	L/S	
Date/time functions	F	L	H	M	L/S	
Waveform operations • Individual waveform quality metrics • Arrival time • Amplitude • Period • Filtering • Beam forming • Rotation • Polarization features • Convolution and deconvolution • SNR • PSD • Spectrograms • Background noise statistics • fk • Slowness • Back-azimuth • FFT • Waveform correlation • etc.	F	L	H	M	L/S	Compute one operation per invocation.

C:Coarse F:Fine H:High L:Low M:Medium -:any valid option Service column: A:Application L:Library S:Service. The primary option appears first when multiple options are suggested.						
Waveform quality • Gaps • Repeated amplitudes • Amplitude Spikes • …	F	L	H	M	S	All metrics at once
Alphanumeric operations • Network magnitude • Station magnitude • Yield	F	L	H	M	L/S	Compute one operation per invocation.
Phase identification (also used by assoc.)	F	M	H	M	S	
Earth models : • General EM, one op. per call • General EM, many ops. per call • 1D • 2D • 2.5D • 3D	 M F F F M M	 M M M M M M	 H H H H H H	 H H H H H H	 S/L S L/S L/S S S	Operations include: - Correction surfaces - Travel time - Azimuth - Slowness - Attenuation - Blockage - Uncertainties
Classic signal detection • Arrival time • Phase identification • SNR • Amplitude • Period • Magnitude • Yield	M	M	H	M	S	All calculations occur during a single invocation.
Waveform correlation based signal analysis	C	H	H	M	S	Assume a full system acting as a detector, associator, locator, and identifier.
Analyst collaboration tools • Analyst to analyst messaging • Message broadcasting • Data object brokerage (e.g., trading arrivals)	F-M	H	M	L	S	

APPENDIX B: MESSAGING PERFORMANCE RESULTS

Test Bed
Performance tests were repeated on three machines:
Workstation –Windows 7 workstation with Sandy bridge Xeon E5-1620 processor
Linux server – RHEL 6 server with Nehalem Xeon xx5570 processors
Solaris server– Solaris SunOS 5.10 server with SPARC processor (contains database)
Waveform NAS: NetApp FAS3240, 256GB cache, 1Gb network connection (connected to separate switch than the processing machines)

Test Configuration
Each test is run using a pattern designed to isolate tests from one another:

```
testData = gatherTestData()
for i : 1 to numIterations
     if(i == startTimingIteration)
          Start timer
     runTest(testData)
stop timer
record results
clean up and garbage collect
```

The test data array is initialized prior to running any tests. Several iterations are run before the timer starts to allow JVM runtime compilers to optimize code used by the test without affecting the timing measurements.
Average processing time per message is computed from total processing time spent on each test.

Results
The table below details messaging costs for a selection of datatypes and sizes of potential system messages. The tests were run on the three machines (Windows 7 workstation, a Linux server, and a Solaris server previously described.

Database account information:
The following data from IDC database accounts was used for messaging performance tests.
- idcleb: reading *origin*, *origerr*, *assoc*, and *arrival* tables
- idcstatic: reading *site* table
- separate user account: writing *origin*, *origerr , assoc*, *arrival*, and *site* tables
- idcidcx.wfdisc_snl: reading the *wfdisc* table and obtaining waveform references
- Starting time reference for SQL queries = 1136073600 epoch seconds (Sun, 01 Jan 2006 00:00:00 GMT)

Legend:
Read: read data. Includes database and disk access as appropriate
Write: write data. Include writing to database and disk as appropriate
M(XML): Marshal human readable XML message
M(bin): Marshal binary XML message
U(XML): Unmarshal human readable XML message
U(binary): Unmarshal binary XML message

XML Size: size of human readable XML message
W: test time for the Windows workstation
L: test time for the Linux server
S: test time for the Solaris server

Table 2. Messaging performance

Test #	Data type	Details	Amount of data		XML Size (byte)	W (ms)	L (ms)	S (ms)
1	Events	select * from origin o, arrival ar, assoc a where ar.arid = a.arid and a.orid = o.orid and o.time > 1138225233 and o.time < 1138225234	1 origin with 10 arrivals/assocs	Read	N/A	96	49	206
				Write	N/A	250	228	215
				M(XML)	9660	14	17	29
				U(XML)	9660	19	22	35
				M(bin)	3962	9	19	20
				U(bin)	3962	11	15	18
2	Events	select * from origin o, arrival ar, assoc a where ar.arid = a.arid and a.orid = o.orid and o.time > 1137593003 and o.time < 1137593004	1 origin with 100 arrivals/ assocs	Read	N/A	61	56	171
				Write	N/A	527	500	409
				M(XML)	92661	18	25	49
				U(XML)	92661	27	39	72
				M(bin)	34185	16	24	51
				U(bin)	34185	13	18	37
3	Events	select * from origin o, arrival ar, assoc a where ar.arid = a.arid and a.orid = o.orid and o.time > 1136073600 and o.time < 1136152800	61 origins with 1,017 arrivals/ assocs	Read	N/A	216	217	323
				Write	N/A	3116	2726	2746
				M(XML)	961093	150	210	447
				U(XML)	961093	200	284	538
				M(bin)	352167	171	222	517
				U(bin)	352167	131	149	374
4	Events	select * from origin o, arrival ar, assoc a where ar.arid = a.arid and a.orid = o.orid and o.time > 1136073600 and o.time < 1136749500	558 origins with 10,018 arrivals/ assocs	Read	N/A	2026	2056	2407
				Write	N/A	24921	23917	25962
				M(XML)	9437446	1519	2020	4541
				U(XML)	9437446	1957	2552	5047
				M(bin)	3420950	1927	2382	5885
				U(bin)	3420950	1283	1443	3604
5	Origins	select * from origin where time > 1136073600 and	10 origins	Read	N/A	28	19	52
				Write	N/A	76	77	76
				M(XML)	4748	2	2	3

Test #	Data type	Details	Amount of data		XML Size (byte)	W (ms)	L (ms)	S (ms)
		time < 1136088000		U(XML)	4748	3	5	7
				M(bin)	1952	1	1	4
				U(bin)	1952	1	1	3
6	Origins	select * from origin where time > 1136073600 and time < 1136203200	100 origins	Read	N/A	31	26	59
				Write	N/A	216	182	185
				M(XML)	46713	8	10	24
				U(XML)	46713	11	14	29
				M(bin)	17489	9	11	28
				U(bin)	17489	7	8	19
7	Origins	select * from origin where time > 1136073600 and time < 1137443400	1,000 origins	Read	N/A	136	76	117
				Write	N/A	1470	1420	1460
				M(XML)	466466	75	98	237
				U(XML)	466466	103	130	268
				M(bin)	173676	85	111	275
				U(bin)	173676	64	74	189
8	Origins	select * from origin where time > 1136073600 and time < 1146067200	10,000 origins	Read	N/A	1793	1507	1732
				Write	N/A	13086	12783	13721
				M(XML)	4660676	771	1021	2461
				U(XML)	4660676	1011	1312	2707
				M(bin)	1727404	938	1205	3022
				U(bin)	1727404	649	756	1918
9	Bulletin	select * from origin o, origerr oe where o.orid = oe.orid and o.time > 1136073600 and o.time < 1136088000	10 origins/ origerrs	Read	N/A	33	35	89
				Write	N/A	169	147	151
				M(XML)	8865	3	3	5
				U(XML)	8865	3	4	7
				M(bin)	4078	2	3	7
				U(bin)	4078	2	2	5
10	Bulletin	select * from origin o, origerr oe where o.orid = oe.orid and o.time > 1136073600 and o.time < 1136203200	100 origins/ origerrs	Read	N/A	270	46	101
				Write	N/A	451	431	346
				M(XML)	88522	17	22	51
				U(XML)	88522	22	28	59
				M(bin)	40276	20	23	57
				U(bin)	40276	13	15	37
11	Bulletin	select * from origin o, origerr oe where o.orid = oe.orid and o.time > 1136073600 and o.time < 1137443400	1,000 origins/ origerrs	Read	N/A	1450	1426	1474
				Write	N/A	2899	2748	2756
				M(XML)	883845	159	204	476
				U(XML)	883845	187	242	504
				M(bin)	399282	182	233	561
				U(bin)	399282	125	146	370
12	Bulletin	select * from origin o, origerr oe where o.orid = oe.orid and o.time > 1136073600 and	10,000 origins/ origerrs	Read	N/A	3643	3442	3825
				Write	N/A	25179	24620	30216
				M(XML)	8807786	1639	2129	4968
				U(XML)	8807786	1887	2441	5027

Test #	Data type	Details	Amount of data		XML Size (byte)	W (ms)	L (ms)	S (ms)
		o.time < 1146067200		M(bin)	3933040	2312	2739	7264
				U(bin)	3933040	1258	1478	3709
13	Sites	select * from idcstatic.site where lat > 0 and lat < 10 and lon > 0 and lon < 140	12 sites	Read	N/A	21	16	31
				Write	N/A	82	73	70
				M(XML)	3104	0	1	3
				U(XML)	3104	1	1	3
				M(bin)	1088	1	1	3
				U(bin)	1088	0	1	2
14	Sites	select * from idcstatic.site where lat > 30 and lat < 60 and lon > 30 and lon < 60	108 sites	Read	N/A	31	21	35
				Write	N/A	152	147	144
				M(XML)	28767	4	6	13
				U(XML)	28767	7	9	19
				M(bin)	10816	5	5	14
				U(bin)	10816	3	4	10
15	Sites	select * from idcstatic.site where lat > -45 and lat < 90 and lon > 0 and lon < 180	1,003 sites	Read	N/A	64	57	63
				Write	N/A	1008	972	910
				M(XML)	264617	35	47	107
				U(XML)	264617	48	63	133
				M(bin)	90552	41	51	115
				U(bin)	90552	30	35	87
16	Sites	select * from idcstatic.site	1,700 sites	Read	N/A	175	72	87
				Write	N/A	1669	1553	1526
				M(XML)	449327	59	80	178
				U(XML)	449327	82	106	220
				M(bin)	153903	68	87	202
				U(bin)	153903	52	59	147
17	Waveform	From idcidcx.wfdisc_snl table -- station "KK31", channel "be", start time = 1136766609, end time = 1136766668.975	2,400 samples (1 minute @ 40Hz)	Read	N/A	1284	289	571
				Write	N/A	4	186	188
				M(XML)	60767	11	14	30
				U(XML)	60767	18	21	48
				M(bin)	22874	12	14	31
				U(bin)	22874	13	15	39
18	Waveform	From idcidcx.wfdisc_snl table -- station "KK31", channel "be", start time = 1136766609, end time = 1136768408.975	72,000 samples (30 minutes @ 40Hz)	Read	N/A	1304	288	574
				Write	N/A	19	317	380
				M(XML)	1828011	328	401	885
				U(XML)	1828011	528	610	1436
				M(bin)	325219	340	407	878
				U(bin)	325219	380	430	1115
19	Waveform	From idcidcx.wfdisc_snl table -- station	144,000 samples (60 minutes @	Read	N/A	1202	290	583
				Write	N/A	34	395	302

Test #	Data type	Details	Amount of data		XML Size (byte)	W (ms)	L (ms)	S (ms)
		"KK31", channel "be", start time = 1136766609, end time = 1136770208.975	40Hz)	M(XML)	3658203	669	804	1785
				U(XML)	3658203	1061	1212	2850
				M(bin)	628580	683	817	1734
				U(bin)	628580	759	850	2239
20	Waveform	From idcidcx.wfdisc_snl table -- station "KK31", channel "be", start time = 1136766609, end time = 1136773808.975	288,000 samples (120 minutes @ 40Hz)	Read	N/A	1814	451	893
				Write	N/A	66	539	577
				M(XML)	7314793	1328	1640	3558
				U(XML)	7314793	2125	2424	5679
				M(bin)	1235341	1366	1652	3476
				U(bin)	1235341	1525	1726	4468
21	String	N/A	Length of 100	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	193	0	0	0
				U(XML)	193	0	1	1
				M(bin)	127	0	0	0
				U(bin)	127	0	0	0
22	String	N/A	Length of 1,000	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	1093	0	0	0
				U(XML)	1093	0	0	1
				M(bin)	1030	0	0	1
				U(bin)	1030	0	0	0
23	String	N/A	Length of 10,000	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	10093	0	0	1
				U(XML)	10093	0	1	1
				M(bin)	10030	0	0	1
				U(bin)	10030	0	0	1
24	String	N/A	Length of 100,000	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	100093	3	5	16
				U(XML)	100093	2	3	12
				M(bin)	100030	2	3	10
				U(bin)	100030	3	4	11
25	Image	Satellite image from Google maps	400x600 GIF	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	253713	206	230	1171
				U(XML)	253713	80	74	979
				M(bin)	253650	206	262	1189
				U(bin)	253650	76	80	1029
26	Image	Satellite image from Google maps	400x600 PNG	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	1003049	679	807	2611
				U(XML)	1003049	250	301	1533

Test #	Data type	Details	Amount of data		XML Size (byte)	W (ms)	L (ms)	S (ms)
				M(bin)	1002986	705	889	2468
				U(bin)	1002986	252	277	1517
27	Image	Satellite image from Google maps	1200x1800 GIF	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	2051161	1908	2279	4956
				U(XML)	2051161	522	544	2009
				M(bin)	2051098	1974	2387	5214
				U(bin)	2051098	531	613	2032
28	Image	Satellite image from Google maps	1200x1800 PNG	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	8726565	5954	7154	15457
				U(XML)	8726565	2188	2464	6724
				M(bin)	8726502	6066	7386	16397
				U(bin)	8726502	2103	2403	5953
29	Image	Non-color map image from Pyxis	400x600 GIF	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	169589	220	251	737
				U(XML)	169589	56	52	427
				M(bin)	169526	231	300	743
				U(bin)	169526	56	56	412
30	Image	Non-color map image from Pyxis	400x600 PNG	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	89885	538	661	1437
				U(XML)	89885	63	71	303
				M(bin)	89822	541	660	1499
				U(bin)	89822	63	71	320
31	Image	Non-color map image from Pyxis	1200x1800 GIF	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	1374653	1849	2185	4647
				U(XML)	1374653	375	406	1306
				M(bin)	1374590	1885	2274	4654
				U(bin)	1374590	379	417	1290
32	Image	Non-color map image from Pyxis	1200x1800 PNG	Read	N/A	N/A	N/A	N/A
				Write	N/A	N/A	N/A	N/A
				M(XML)	485493	3429	4566	9499
				U(XML)	485493	414	560	1320
				M(bin)	485430	3434	4632	9506
				U(bin)	485430	416	638	1324
33	Arrival	select * from arrival a where a.time > 1136073600 and a.time < 1136074950	10 arrivals	Read	N/A	31	28	113
				Write	N/A	102	87	87
				M(XML)	5197	13	15	26
				U(XML)	5197	19	21	35
				M(bin)	1925	23	12	18
				U(bin)	1925	14	9	14
34	Arrival	select * from arrival a where a.time > 1136073600 and a.time < 1136087000	100 arrivals	Read	N/A	33	30	102
				Write	N/A	224	191	196
				M(XML)	52072	11	13	41
				U(XML)	52072	19	25	46
				M(bin)	19205	11	12	28

Test #	Data type	Details	Amount of data		XML Size (byte)	W (ms)	L (ms)	S (ms)
				U(bin)	19205	8	11	21
35	Arrival	select * from arrival a where a.time > 1136073600 and a.time < 1136153300	1,000 arrivals	Read	N/A	91	84	139
				Write	N/A	1563	1460	1505
				M(XML)	520206	83	104	240
				U(XML)	520206	107	151	276
				M(bin)	187584	94	117	280
				U(bin)	187584	66	93	193
36	Arrival	select * from arrival a where a.time > 1136073600 and a.time < 1136747562	10,000 arrivals	Read	N/A	613	619	872
				Write	N/A	13658	13406	14274
				M(XML)	5190930	873	1121	2600
				U(XML)	5190930	1059	1455	2755
				M(bin)	1851943	1038	1346	3045
				U(bin)	1851943	674	940	1926

APPENDIX C: SAMPLE PIPELINE PERFORMANCE RESULTS

Pipeline performance metrics computed using the Linux server and human readable XML messages. Control message marshaling and unmarshaling costs are negligible compared to costs of data messages and are ignored (costs in the messaging performance table are at most 1ms for messages up to 10,000 characters).

Legend:
PUT: Store to database, including disk access for writing waveforms.
GET: Read from database, including disk access for reading waveforms.
M: Marshal message
U: Unmarshal message
arrivalWF: arrival data for 1 minute of waveform data across the network
arrivalAll: arrival data for 45 minutes of waveform data across the network
WF: 1 minute of waveform data at a single station
Event: network event data for 45 minutes

Costs for single operations (numbers outside parenthesis) and total cost during the 45 minute pipeline processing time period (numbers inside parenthesis) are reported in the cost column.

Table 3. Performance results

	Acquire data	Cost	Signal processing	Cost	Event processing	Cost	Write bulletin	Cost	Subtotals (ms)	Totals (s)
SERVICE-TO-SERVICE COMMUNICATION										
Light (direct data access) input		0	GET(WF)	289 (650250)	GET(arrivalAll)	84	N/A: side effect of previous step	0	650334	1075.475
output	PUT(WF)	186 (418500)	PUT(arrivalWF)	87 (3915)	PUT(event)	2726		0	425141	
Light(data service) input		0	GET(WF) + M(WF) + U(WF)	324 (729000)	GET(arrivalAll) + M(arrivalAll) + U(arrivalAll)	339	N/A: side effect of previous step	0	729339	1235.344
output	M(WF) + U(WF) + PUT(WF)	221 (497250)	M(arrivalWF) + U(arrivalWF) + PUT(arrivalWF)	123 (5535)	M(event) + U(event) + PUT(event)	3220		0	506005	
Rich (direct data access) input		0	U(WF)	21 (47250)	U(arrivalAll)	151	U(event)	284	47685	82.796
output	M(WF)	14 (31500)	M(arrivalWF)	15 (675)	M(event)	210	PUT(event)	2726	35111	

	Acquire data	Cost	Signal processing	Cost	Event processing	Cost	Write bulletin	Cost	Subtotals (ms)	Totals (s)
Rich (data service) input		0	U(WF)	21 (47250)	U(arrivalAll)	151	U(event)	284	47685	
output	M(WF)	14 (31500)	M(arrivalWF)	15 (675)	M(event)	210	M(event) + U(event) + PUT(event)	3220	35605	83.29
CENTRL CONTROLLER COMPONENT										
Light (direct data access) input	same as above	0	same as above	289 (650250)	same as above	84	same as above	0	650334	
output	same as above	186 (418500)	same as above	87 (3915)	same as above	2726	same as above	0	425141	1075.475
Light(data service) input	same as above	0	same as above	324 (729000)	same as above	339	same as above	0	729339	
output	same as above	221 (497250)	same as above	123 (5535)	same as above	3220	same as above	0	506005	1235.344
Rich (direct data access) controller input			U(WF) + M(WF)	35 (78750)	U(arrivalAll) + M(arrivalAll)	255	U(event) + M(event)	494	79499	
		0	U(WF)	21 (47250)	U(arrivalAll)	151	U(event)	284	47685	162.295
output	M(WF)	14 (31500)	M(arrivalWF)	15 (675)	M(event)	210	PUT(event)	2726	35111	
Rich (data service) controller input			U(WF) + M(WF)	35 (78750)	U(arrivalAll) + M(arrivalAll)	255	U(event) + M(event)	494	79499	
		0	U(WF)	21 (47250)	U(arrivalAll)	151	U(event)	284	47685	162.789
output	M(WF)	14 (31500)	M(arrivalWF)	15 (675)	M(event)	210	M(event) + U(event) + PUT(event)	3220	35605	


Sandia
National
Laboratories


U.S. DEPARTMENT OF
ENERGY