PNNL-24340

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Building the Analysis in Motion Infrastructure

June 2015

K Kleese van Dam
RR LaMothe
A Vishnu
WP Smith
M Thomas
P Sharma
DV Zarzhitsky
E Stephan
TD Elsethagen

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <http://www.ntis.gov/about/form.aspx>
Online ordering: http://www.ntis.gov

This document was printed on recycled paper.
(8/2010)

PNNL-24340

Building the Analysis in Motion Infrastructure

K Kleese van Dam
RR LaMothe
A Vishnu
WP Smith
M Thomas
P Sharma
DV Zarzhitsky
E Stephan
TD Elsethagen

June 2015

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

Abstract

Science and national security missions are driven by the need to assimilate and interpret ever-increasing volumes of data to accelerate scientific discovery and make critical decisions, so the speed of analysis is as important as the choice of data to be collected. The Analysis in Motion Initiative (AIM) proposes to develop a new analysis paradigm—persistent/ dynamic knowledge synthesis—that will provide continuous, automated synthesis of new knowledge and dynamic control of measurement systems contemporaneously with observed phenomena. Working on streaming data, this new capability will automate the current time-intensive manual analysis and interpretation steps and collaborate with scientists and analysts to optimize insight creation, decision making, analysis, and data capture adaptation to meet the needs of their discovery process in a timely manner. This technical report outlines the creation of the underpinning software infrastructure that enabled the streaming and adaptive analysis approach of AIM. We report the results of our requirement capture, technology selection, initial infrastructure design, and changes introduced based on year-one operational experiences.

Summary

Today the ability to make sense of data is foundational to all discoveries, innovations, and decision making; the outcome of our work often critically depends on the speed and adaptability of the analysis and interpretation processes we employ. While much progress has been made in automating the analysis of standard events, little is available to support complex or rare event analysis situations that require human knowledge and ingenuity in addition to high-speed, high-volume analytical and interpretive processes. Consider examples such as emergency response, scientific discovery, national security, or critical business decisions where humans play a key role in analyzing facts as the situation evolves. The Analysis in Motion (AIM) Initiative is developing streaming analysis applications and an infrastructure to support this new analysis paradigm for high-volume, high-velocity data situations. This report describes the first 12 months of AIM infrastructure development. The work discussed can be broken into three principal stages: selection of key enabling technologies and their initial integration, extension of the original design to support multiple users, and transition to the OpenStack-based Pacific Northwest National Laboratory (PNNL) Institutional Research Cloud to scale past the limitations of the initial single-server hosting model.

Current commercial stream data processing frameworks (e.g., Amazon's Kinesis or International Business Machine's Infosphere Streams) as well as state-of-the-art scientific systems, like the Large Hadron Collider's triggers, use highly parallel scale-out techniques and have been shown to reliably process millions of data elements per second; however, they are strongly oriented towards implementing fixed, fully-automatic workflows (e.g., real-time retail microtargeting) and leave the user out of the loop. As part of the AIM Initiative we decided to investigate if we could build an adaptive, user in the loop, high data velocity infrastructure from these existing commodity software components and evaluate which would be most suitable for our challenge.

We started the work by defining key infrastructure requirements in terms of maximum throughput, ability to adapt the mixture of analysis models at runtime, and easy integration of different programming languages and models. This list of requirements was then translated into evaluation criteria that were used to review a wide range of existing solutions. Interestingly, while basic requirements such as throughout rates could be met by many, the native and high-performance support of multiple programming models proved to be the deciding factor. Based on our evaluation we chose the Apache Kafka framework as our key infrastructure component for our initial implementation, coupled with Apache Avro containers and the PNNL-developed Laboratory Integration Framework and Toolset (or LIFT) integration infrastructure.

The initial implementation was hosted on a single high-performance computing node and provided a web interface controlled test and execution environment where individual users could control analysis models, data streams and message queues. In initial user tests we discovered difficulties in providing stream state synchronization between low-level Kafka primitives and higher-order representational state transfer interfaces in the AIM server software. In addition, the single-node PNNL Institutional Computing hosting environment with shared, network-based file repository proved to be a suboptimal configuration for running the ZooKeeper services that support Kafka messaging. Network issues, increased latency, and lapses in reliability of InfiniBand connections all contributed to periodic failures of the AIM software, requiring complete system reinitialization. Although tolerable for short-term development activities and useful for evaluating various aspects of AIM design, reliability and scalability concerns required a rethinking of the future direction for the software development effort. However, the initial infrastructure

could sustain message rates of up to 600K/sec when running, it provided the flexibility to integrate different programming languages and models and analysis model configurations could be changed at runtime. It was therefore deemed sensible to continue with the initial core components, but improve them as we moved into phase two of the infrastructure development.

Our development work coincided with the introduction of PNNL Research Cloud infrastructure, which enabled us to take advantage of a number of foundational technologies to improve the AIM codebase in a principled and significant way. The cloud ecosystem brought elastic capabilities to AIM, while OpenStack Infrastructure-as-a-Service (IaaS) features provided a path forward to ensure high availability and horizontal scale out of the AIM services. The popularity of OpenStack and active community support allowed us to use existing solutions for template-based specification of AIM services, leveraging OpenStack HEAT orchestration, meaning that AIM clients can expect a consistent level of system performance regardless of the data volume being processed by the system. (This does assume a certain reasonable limitation on the network capacity and ability to combine virtual machines into a reliable cluster with a distributed-memory data repository capability for massive parallel processing tasks.) AIM clients will also benefit from active, real-time monitoring of the system, with automated response to outages and service interruptions. Overall, the cloud migration of AIM enables both a significantly greater performance scale and an operationally improved reliability of AIM service at only a modest increase in hardware resource usage—a cost that is optimally shared across all cloud tenants. Based on these assessments, the team migrated the existing AIM infrastructure into a cloud-based environment, making the necessary changes to ensure greater stability of the environment and enabling multiuser access to its capabilities. The new system is currently under extensive user testing and holding up well.

Based on experiences with the current infrastructure implementation, the team has identified a range of additional research and development areas, core among them are the runtime adaptation of analysis model coupling and scalability work to reach the necessary data throughput on high data volumes.

The AIM software infrastructure team plans to release any modifications or contributions to the open-source software used in the AIM implementation back to the community. There is an important, virtuous cycle of innovation, contribution, and idea sharing that factors into the value of the AIM Initiative's contributions. We hope that by actively integrating the most innovative concepts and fielding novel technologies in challenging problem domains we advance the state of the art in streaming data analytics. We expect that feedback as to which of these technologies worked well and which failed to realize their potential for streaming data analysis will help the community better direct their engineering efforts and continue to improve the toolbox with which we address important open problems and challenges.

Acknowledgments

The work described in this report is part of the Analysis in Motion Initiative at Pacific Northwest National Laboratory. It was conducted under the Laboratory Directed Research and Development Program at Pacific Northwest National Laboratory, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy.

Acronyms and Abbreviations

AD	Active Directory
AIM	Analysis in Motion
API	application program interface
CEP	complex event processing
CPU	central processing unit
DOE	Department of Energy
GB	gigabyte
GMT	Global Memory and Threading
GUI	graphical user interface
HDFS	Hadoop Distributed File System
HPC	high-performance computing
HTTP	hypertext transfer protocol
IBM	International Business Machine
JavaEE	Java Enterprise Edition
JMS	Java Message Service
JMX	Java Management Extensions
JNI/JNA	Java Native Access/Java Native Interface
JSON	JavaScript Object Notation
JVM	Java virtual machine
LDAP	Lightweight Directory Access Protocol
LIFT	Laboratory Integration Framework and Toolset
MPI	Message-Passing Interface
MQ	message queue
NMR	nuclear magnetic resonance
NPM	Native Programming Model
OPA	Online Predictive Analysis
PIC	PNNL Institutional Computing
PNNL	Pacific Northwest National Laboratory
RAM	random access memory
REST	Representational State Transfer
SHyRe	Streaming Hypothesis Reasoning
SPL	Streams Processing Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security

Contents

Figures

Tables

1.0 Introduction

Science and national security missions are driven by the need to assimilate and interpret ever-increasing volumes of data to accelerate scientific discovery and make critical decisions, so the speed of analysis is as important as the choice of data to be collected. Ideally we would like to identify and interpret phenomena and events of interest as they are emerging and adapt our analysis and data collection as they evolve to create an optimized set of information in support of our discoveries and decisions. However, today's predominant analysis paradigm remains the *post-hoc* evaluation of results, often relying heavily on manual labor, in particular in the value-adding areas of results interpretation in the domain context and hypothesis evaluation. This strongly human-centered approach does not offer the scalability required for timely decision making in big data environments, delaying or preventing necessary decisions and actions by days, months, or years.

The Analysis in Motion (AIM) Initiative proposes to develop a new analysis paradigm—persistent/ dynamic knowledge synthesis—that will provide continuous, automated synthesis of new knowledge and dynamic control of measurement systems contemporaneously with observed phenomena. Working on streaming data, this new capability will automate currently time-intensive manual analysis and interpretation steps and allow scientists and analysts to optimize data taking to meet the needs of their discovery or decision-making process in a timely manner. To achieve this goal, AIM will focus its research activities on four areas that combine to accelerate the complete analysis cycle:

1. Streaming characterization methods that can identify and tag features of importance in high-rate, large-volume data streams
2. Continuously evolving models that can interpret identified features as early indicators for phenomena of interest and relate these features and phenomena into explanatory hypothesis that can assist humans and machines in interpretation of results and optimization of future data taking
3. Capturing human background knowledge through a new interaction paradigm that will not only support evaluation of candidate hypothesis, but introduce new knowledge into the analysis process
4. A streaming analysis integration and execution environment that will ensure the timely analysis, interpretation, and steering required.

This technical report will outline the results of our requirement capture, technology selection, initial infrastructure design, and changes introduced based on year-one operational experiences.

2.0 Infrastructure Requirements

AIM is focused on use cases where:

- Data arrive at such high velocity and volume that storage for later analysis might not be possible
- Critical decisions have to be made while data are still arriving
- Events of interest are so rare that it is not possible to train algorithms sufficiently for reliable detection
- Tacit knowledge is required for accurate interpretation of the data.

To address these use cases effectively, AIM proposes to provide a streaming model (see Figure 1) where:

- A wide range of models work collaboratively or in parallel on analyzing and interpreting the data streams together with the user
- Data are forgotten; each model's cache is small relative to the data volume
- Single pass with no access to the data stream beyond the sample

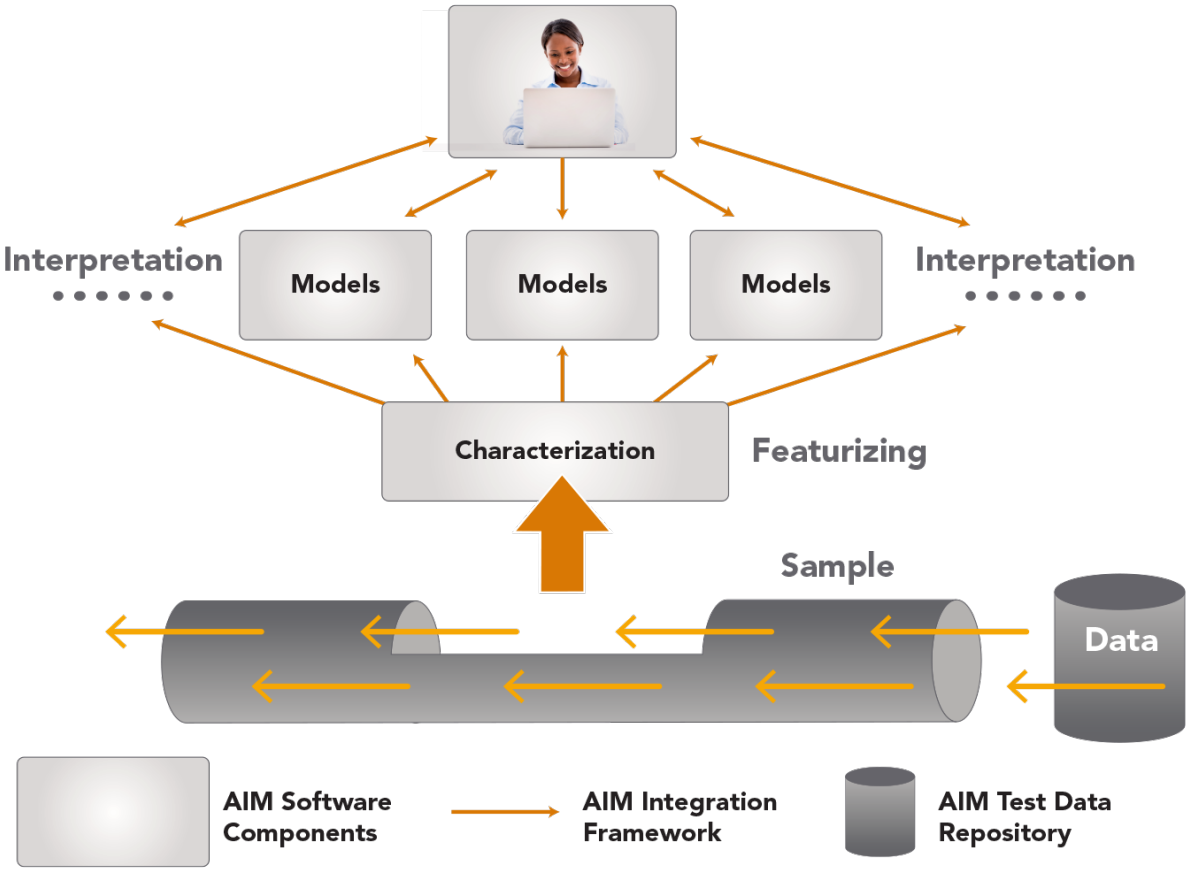


Figure 1. AIM Conceptual Analysis Framework

The AIM infrastructure will need to provide:

- **An integration framework** that supports communication between the different models as required. It is anticipated that different use cases and problem settings within them will require different combinations of models. We also foresee that, driven by the observed phenomena or user decisions, the combination and interaction of models might change at runtime. Therefore, the infrastructure needs to provide the ability to support creation of flexible model couplings, potentially on the fly.
- **Real-life data streams** (e.g., extreme scale computers, experimental instruments, or sensors) made available to the models involved in the analysis.
- **Test data repository and testing range** that provide test data of known quality and can create data streams of varying rates, volumes, and window sizes. Provide code instrumentation, capture application performance and results, and evaluate test results against expected results, metrics, and previous performance.
- **A development environment** in which models can test their functionality and algorithms performance, and tradeoffs can be characterized.

2.1 Quantitative Infrastructure Requirements

Key factors in the infrastructure design are the data rates and volumes to be supported. These rates are determined by a number of initial use cases that were selected by the AIM Initiative, foremost those in the chemical imaging domain, which displays the highest data rates and volumes. The most challenging of these is the analysis and interpretation of images produced by transmission electron microscopes, which are currently experiencing a fast-paced technology change that is driving its data rates into unprecedented rages. In evaluating the use case, we see that the infrastructure needs to support the following initial maximum data rates and volumes:

- Maximum – 2Kx2K image, 1,000,000/second, 2 terabytes/second, 10-second burst
- Moderate – 2Kx2K image, 1000/second, 2 gigabytes (GB)/sec, 1-hour burst

The maximum number of data sources is not yet determined, but likely will not exceed five to ten different sources during the runtime of the initiative. At present we expect the data rate per source to be lower if there are more data sources; for example, if we have a maximum data rate of 1,000,000 2Kx2K images per second it will come only from one source. Very high data rates will only be sustained for a short period, whereas lower data rates might be sustained indefinitely.

The combination of potentially high data volumes and rates, short time to decision, and a complex multistep analysis process will require the resulting system to be closely coupled and highly optimized in its orchestration and execution from an algorithmic, hardware, and networking point of view.

The time from initial data taking to initial hypothesis generation will vary between the different science use cases, but are expected to range from a few seconds to several minutes, thus providing a stringent time window on the execution of the complete analysis pipeline.

The combination of potentially high data volumes and rates, short time to decision, and a complex multistep analysis process will require the resulting system to be closely coupled and highly optimized in its orchestration and execution from an algorithmic, hardware, and networking point of view. This excludes solutions that require intermediate file input/output (e.g., Hadoop) or commercial cloud access (e.g., Amazon Kinesis) due to latency.

2.2 Levels of Parallelism and Model Coupling

In examining the use cases further we can detect different levels of parallelism and coordination that are required. Single analysis applications might range from modest to highly parallel. These applications might be run in parallel with others of the same type, requiring modest coordination at the start (who works on what) and the end (combination, correlation, or bakeoff of results). Furthermore, these clusters of applications of the same type might be run in parallel with other clusters of other applications, with a potential need for coordination at the end to assess, rank, and filter results. The more parallel the single analysis algorithm is, the less of these algorithms will need to be run in parallel at any given time. It is expected that instances of all analysis steps are running at any given time; however, they might not all be working on the same data window at the same time, but on prior or subsequent windows. The user will be able to give feedback that will flow into the system in the opposite direction of the data stream, influencing future analysis steps.

Projects need to run at moderate to high parallelism, requiring orchestration and coordination of multiple tasks running in parallel.

2.3 Programming Language and Model Expertise

AIM researchers need to use familiar programming languages and models to be most effective in their model design and development, given the complex nature of their research and relatively short individual project life cycles. A brief survey revealed that different project groups have varied experiences in programming languages, ranging from C, C++, Fortran, R, and MATLAB to Python and Java. Most researchers are familiar with one or two languages but feel the learning curve to adapt a third would be too high an overhead for their project. When it comes to parallel programming models, Message-Passing Interface (MPI) programming is the predominant expertise, with only a few researchers familiar with functional programming models such as MapReduce.

2.4 New Development versus Utilization of Existing Technologies

Given the requirements of the AIM Initiative, a fast complex event processing (CEP) system was determined to be most suitable. As this is not a green field and many solutions already exist, we will use existing tools and capabilities as far as possible and focus any development needs on areas than cannot be met by existing tools.

2.5 Business Requirements

Laboratory directed research and development programs (LDRD) such as AIM are the principle mechanism to develop novel ideas, extend the laboratory's capabilities, and attract new business.

Therefore, when designing the computational fabric for an initiative, we need to also consider which capabilities would allow us to build future business with key clients. For AIM we have identified two key client areas: open science and national security.

Open science requirements include:

- Basic science research, which is currently funded by the Department of Energy (DOE) Office for Advanced Scientific Computing Research, the National Institute of Health, and the National Science Foundation, all require the developed and utilized tools to be open source in their entirety.
- Applied research, tools, centers, and services are funded by DOE Office for Biological and Environmental Research, DOE Basic Energy Research, DOE High Energy Physics, the National Institute of Health, and the National Science Foundation, all require the utilized tools to be open source in their entirety.
- Internal usage that the Environmental Molecular Sciences Laboratory at Pacific Northwest National Laboratory (PNNL) would be prepared to accept with commercial licensing and costs at a moderate rate, if benefits can be demonstrated.
- Open science prefers complete solutions, analysis models and infrastructure, and do rarely adopt tools that require significant additional development or integration effort.
- National security requirements include:
- The analytics business model is a deploy-in-client-platform approach for most national security clients.
- Clients use a plethora of platforms (streaming or not) so the technical approach should be platform independent to the extent possible, but it would be beneficial if the newly developed capability could be demonstrated in an environment that is similar to the client's in its main characteristics.
- Where clients adopt platforms they tend to be commercial or commercially supported open source, if possible.
- Cloud computing solutions, including cloud + high-performance computing (HPC), are beginning to dominate.

Clearly the business requirements for the two client spaces are quite different; however, there is a middle ground that would serve both. Commercially supported open-source software is a model that works for both client spaces; solutions that use standard programming languages and models as well as being modular are preferred by both communities to enable easier, low-risk transition.

3.0 Evaluation of Existing Infrastructure Frameworks

As a result of our requirements gathering process, CEP solutions were evaluated to see if they met the initiative's needs and which would be most suitable.

3.1 Resulting Evaluation Criteria

Our initial evaluation criteria included the following metrics:

- Technical
  - Performance capabilities
  - Framework programming language
  - Supported programming languages and model
  - Third-party integration capabilities
  - Instrumentation (monitoring, debugging)
- Available ecosystem
  - Available analysis libraries
  - Visualization tools
  - Training
  - Support
- Costs
  - License
  - Training
  - Support
  - Learning curve
- Risk Opportunities
  - Risk
  - Experience with solution at PNNL

3.2 Evaluated Solutions

As part of our review we investigated the following frameworks:

- Amazon Kinesis
- Apache Spark
- Apache Storm
- Cloud::Streams

- GridGain In-Memory Streaming
- International Business Machines (IBM) InfoSphere Streams
- PNNL Laboratory Integration Framework and Toolset (LIFT)
- Red Hat JBoss Data Grid + Infinispan
- SAS® Event Stream Processing Engine
- Yahoo Big Data
- Yahoo SAMOA
- Native Programming Models (NPMs)

Evaluation of the above frameworks provided the following high-level results for the front runner technologies (see Table 1, a fuller list of evaluation criteria and results can be found in Appendix A).

Table 1. Results of Initial Evaluation of Possible Messaging Framework Solutions

Framework	Technical	Ecosystem	Cost	Risk
Apache Storm	1. Java 2. C, C++	Java/Scala ecosystem	Open Source Supported	Medium/Low PNNL known
GridGain	1. Java, Scala	Java/Scala ecosystem	Open Source/ Commercial	Medium/Low PNNL known
InfoSphere	1. SPADE/SPL 2. C, C++, Java	Application Library	Commercial	High Learning curve
LIFT	All languages All models	Java/Scala ecosystem	Open source	Medium/Low PNNL known
Yahoo Big Data	1. Java, Scala 2. C, C++,R	Vast No. of open libraries	Open Source Supported	Medium
NPM MPI	All languages	Vast No. of open libraries	Open source	Low PNNL known

In the end no clear front runner could be identified in this field, given the selected criteria. However, our investigation found that most of the frameworks were limited in the range of programming models they could support, while still providing high performance. We decided to investigate in more detail which programming models needed to be supported by the AIM infrastructure framework to help narrow our choices.

3.2.1 Embarrassingly Parallel Computations

Each computation is entirely independent of the other and there is no requirement for synchronization. The teams have several algorithms that can be classified in this category. For these types of applications, a MapReduce-based programming model with in-memory computation would be sufficient, although MPI-based solutions would provide similar or better performance.

3.2.2 Embarrassingly Parallel + Reduction/Aggregation

These algorithms divide data among parallel entities, perform computation locally (such as kernel calculation K-means or self-self-distance calculation in canopy clustering) and synchronize at the end of each step for reduction/aggregation. Several AIM algorithms fall under this category. These algorithms fit well with both MapReduce and MPI, with the choice of programming solution dependent on the algorithm and data requirements.

3.2.3 Bulk Synchronous Communication + Reduction/Aggregation

Many clustering algorithms require asynchronous data movement for distance calculation (e.g., hierarchical agglomerative clustering, canopy clustering, and support vector machines). MapReduce will not work; MPI-based solutions will require careful writing to overlap communication with computation, but present a viable option.

3.2.4 Irregular Communication with Varying Computation

These algorithms provide a very high degree of irregularity in communication and little to no computation. The semantic graph algorithms fall under this category. MapReduce or MPI are not suitable for this type of problem. Giraph and Pregel suffer from communication only at the synchronization points. An alternative is Graphlab, which performs aggregation to reduce the overhead of communication. Within PNNL, the Global Memory and Threading (GMT) run time developed under the Center for Adaptive Supercomputing Software can be considered as an alternative.

We could identify three programming models that AIM is likely to pursue; MapReduce (in-memory), OpenMP/MPI, and Graphlab/GMT. Next we evaluated which of these programming models would be supported by the top framework contenders. For our assessment we used a number of categories:

- Native – framework naturally supports this programming model without any changes or adaptations necessary.
- Potential – framework could support this programming model; however, changes would be needed that would affect the performance to a significant extent.
- Unknown – no information could be found if this programming model can or has been successfully supported by this framework.
- Yes – programming model can be supported but requires changes to the framework; however, no significant performance implications are expected from these changes.

The results of our investigation are presented in Table 2.

Table 2. Programming Model Support Offered by Selected Messaging Frameworks

Frameworks	MapReduce	OpenMP/MPI	Graphlab/GMT
Apache Storm	Native	Potential	Potential
GridGain	Native	Potential	Potential
InfoSphere Streams	Unknown	Native	Unknown
Yahoo Big Data	Native	Potential	Potential
LIFT/NPM (MPI)	Yes	Yes	Yes
LIFT/Apache Kafka	Yes	Yes	Yes

The most complete, flexible, and performant solutions are enabled by low-level messaging models such as MPI or Apache Kafka. A combination of Apache Kafka and LIFT offers the necessary orchestration, configuration, and instrumentation support.

3.3 Architecture Decisions

We decided to recommend the adoption of the Apache Kafka and LIFT combination as the basic event processing framework. Kafka is a message bus to which all components will have access, giving them the ability to sample data or communicate with other components, while remaining independent in their execution (MPI would have required a tighter, direct coupling of the components).

Applications will interact directly with the message bus. If using a programming model with specific higher-level framework for execution such as Apache Storm, Storm will take its data stream directly from Kafka.

LIFT will in particular orchestrate the configuration, start up, and monitoring of the different components involved in the complete AIM analysis process.

4.0 AIM Infrastructure Implementation Phase 1

Based on our core architecture decision we developed and deployed Phase 1 (from May to September 2014) of the AIM infrastructure. On top of Kafka and LIFT, we decided to develop a test data provider that could stream data at user-selected rates and an interface that would allow users to test their models in the infrastructure against the available test data. We also settled on the use of Apache Avro, which is flexible, standard "big data" open-source messaging container (see Figure 2).

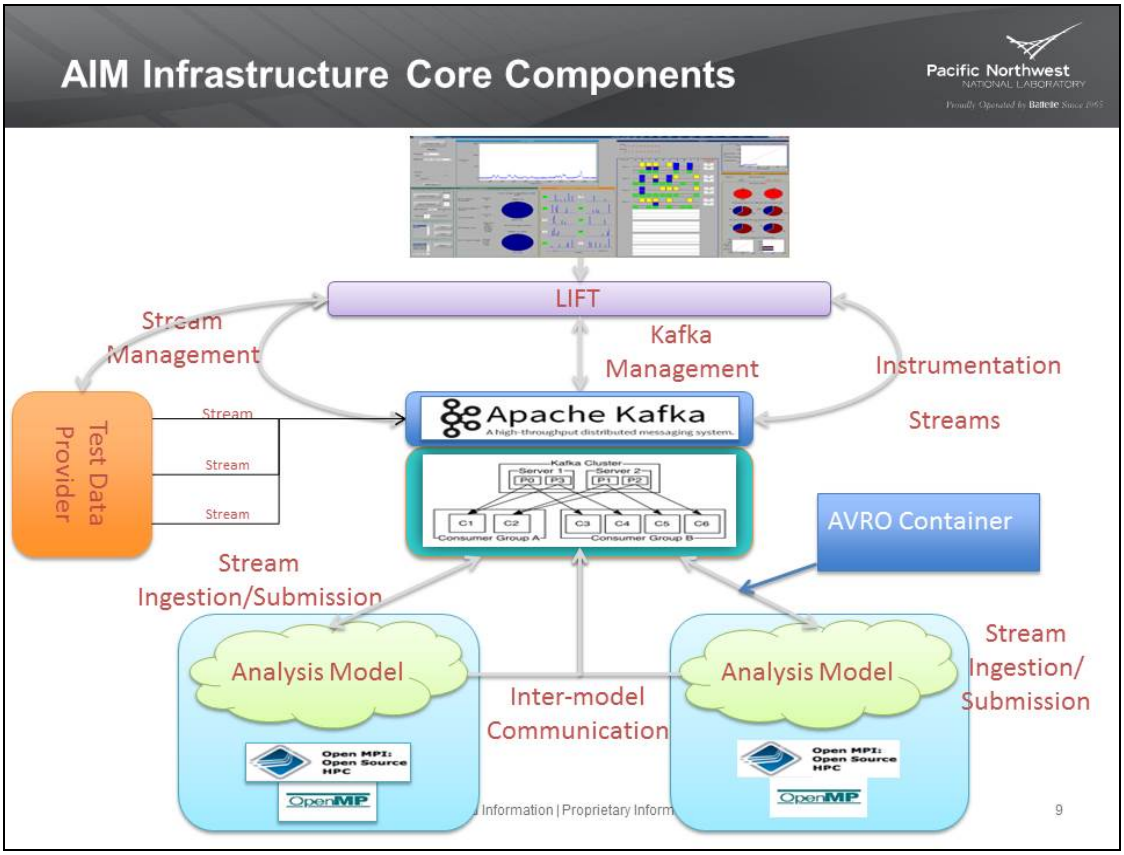


Figure 2. AIM Phase 1 Core Infrastructure Components

In the following sections we describe our initial development and deployment work.

4.1 Kafka Setup

Apache Kafka is a distributed message-processing system based on a disk-first approach, where incoming messages are committed to a disk-backed queue as soon as they are received. [1] The messages remain in the queue until a user-specified age threshold is reached, at which point the queue is truncated and the messages are discarded. Kafka's designers were interested in tracking LinkedIn user interactions with the site in detail and needed a way to collect high-volume streams of data from multiple sources. Kafka architecture incorporates functional elements of both queue-based and subscribe-and-publish methodologies, allowing a high degree of customization while retaining an ordered, deterministic interface for message retrieval. Kafka takes advantage of the high-performance state management

provided by Apache ZooKeeper to support a large-scale, high-availability configuration on low-end commodity hardware.

In Kafka terms, a message queue (MQ) is called a *topic* and it represents the most general message-grouping attribute available. To support distributed message processing, Kafka introduces the notion of a *partition*, which is a sequentially ordered subset of messages in a given topic, with the subset partition key defined by message senders (called *producers* in Kafka documentation). To support high-availability configurations, each topic partition is *replicated* (copied and kept synchronized) across multiple network hosts. Kafka *brokers* are built-in software services that coordinate client access to the partitions. On the client side, concurrent consumers of Kafka messages are organized into consumer groups, with each individual consumer in the group dynamically assigned to a partition within the topic. This design ensures that messages from each partition are consumed in their original order, without placing performance limits on the number of messages that can be sent and received for a given topic.

As messages are read, Kafka brokers maintain a set of partition offsets for different consumer groups. This means that each consumer process can effectively rewind and fast-forward the shared MQ to facilitate its own processing requirements without affecting concurrent consumers in other groups. Kafka brokers also handle failover scenarios, so that topics with a replication factor of *N* can continue to serve messages (perhaps with increased latency) even if *N-1* hosts become unavailable. Since only one client in the consumer group can read from a given topic partition, the number of topic partitions determines the maximum size of a consumer group for that topic. There is no limit on the number of concurrent consumer groups that can participate in the message exchange. Similarly, until they expire due to age limits, all messages can be retrieved an unlimited number of times by the consumers.

AIM infrastructure automated deployment support for Kafka-based messaging services uses a set of Puppet-based configuration scripts. Puppet is a script-based application management system with both commercial and open-source licenses. [3] Hierarchical template files are used to specify active components and their startup options. Puppet also supports active monitoring of service availability, a functionality we plan to exploit in the future. Puppet's ability to query its operating system environment simplifies deployments to multiple targets such as development and production, and supports cross-platform configurations between Linux, Windows, and MacOS hosts. Use of Puppet to manage AIM operational configuration also reduces the need to manually customize source code distribution, thus significantly reducing the amount of time to stand up a fully operational analytic environment "out of the box" on a new system.

4.2 Kafka/LIFT Integration

Keeping with the design ideas of cross-platform support and template-based automation, the AIM infrastructure team based the core software on LIFT. This software package is an internal PNNL product used by several teams for research and large-scale production purposes. [2] At its core, LIFT is a Java-based platform focused on enterprise integration patterns through the strategic use of open source Spring and Spring Integration packages. AIM software built with LIFT support is designed to run within a Java Enterprise compliant container environment. Apache Tomcat application server is used to host AIM at PNNL, but the server software should be compatible with products from other vendors such as Red Hat JBoss Application Servers or any Java Enterprise Edition (JavaEE) compliant servlet engine.

LIFT provides developers with a complete solution for rapidly bootstrapping their code and its dependencies into functioning web applications with industry standard build tools, such as Maven and Gradle. Support provided by the Spring modules helps add advanced enterprise features such as authentication against Lightweight Directory Access Protocol (LDAP) and Active Directory (AD), Transport Layer Security (TLS) encryption, dynamically generated web services with support for both Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) based designs. [8][7] Web service components support both annotation and XML-based configurations to produce documented resource descriptions and contracts that clients can access in a programmatic way. Build utilities used by LIFT include convenient dependency injection capabilities to set configuration values in code at run time, so that the same software can be deployed on multiple different machines without additional code changes. To help maintain software engineering quality, LIFT integrates popular Java quality assurance tools, such as *findbugs* and *cobertura*, which help analyze and report on potential software defects. [4][5] Programmer-directed validation using the *JUnit* testing framework is also supported. [6] All of these development tools and aids are fully integrated with the Eclipse development environment, supporting developer productivity while encouraging good design and implementation practices.

In support of the AIM Initiative, we extended the built-in LIFT connectivity options to provide a web enabled interface for managing and accessing Kafka streams. This interface follows the REST design principles and exposes each Kafka topic as a web service resource. Clients connect using a hypertext transfer protocol (HTTP) family of transfer protocols to a specific endpoint that represents a data resource on the system. Some of these endpoints represent interfaces to predefined data streams and others act as frontends for dynamically created resources. LIFT provides the templates needed to embed several data producing sources to support AIM client data requirements. Each data producer is a Spring *bean* (in JavaEE terms). This architecture relies on LIFT to manage all interactions between the client and the data backend, and effectively abstracts all implementation details about Kafka into a consistent, platform agnostic web interface. True to the original design goals for LIFT, the AIM web services components were constructed with minimal custom code and enforcement of industry standard interfaces. The AIM-driven improvements to LIFT were subsequently backported to the core LIFT distribution for reuse by others.

4.3 Data Producer

The data producer is responsible for reading data from a location, creating messages from the data, and streaming them into a specific Kafka topic. A separate data producer is defined for each type of data. The AIM infrastructure management web service is responsible for calling the data producer when a user requests to start a data stream. In response the matching data producer creates a single stream. In a multiuser test environment, each data stream will further be associated with a specific user (e.g., NMR_Raw_Data_Kerstin). If a user requests to start the same stream twice then an appropriate error message is sent back to the user via the management web service.

The location of the data and the Kafka topic to which it needs to be streamed are specified as a key-value pair in gradle.properties in the AIM infrastructure setup parameters. In multiuser test environments, the data producer appends this topic name with username to create a unique topic for the requesting user, it then iterates over the data location to parse the data and divide them into messages. Each data producer works with a specific Avro schema. It creates the message as a Java object associated with the Avro

schema and serializes it into byte buffer. This byte buffer is then sent to the topic created for the user. The data producer continues to parse the data, create Avro messages, and send them to the topic until the user requests to stop the stream.

For the current use cases, data are stored in csv file format on the system where the AIM infrastructure and models are executed. A single data location contains multiple csv files, each containing data for a large number of messages (e.g., all messages for one complete experiment). The data producer iterates over these files and reads a single row from each file to create an Avro message. For example, it reads the first row of the first file and creates the Avro message, then sends the message over the created topic. When the last row of the last file is read and sent over the topic, it restarts from the first row of the first file. This loop continues until the user requests to stop the stream.

4.4 User Interface for Testing Range

To use the infrastructure to test algorithms we developed a testing range graphical user interface (GUI). In phase one of the infrastructure setup we had two research projects that we supported in their model development—Online Predictive Analysis (OPA) and Streaming Hypothesis Reasoning (SHyRe). The test case scenario we chose allowed them to test their models separately or together. The initial data stream provided consisted of nuclear magnetic resonance (NMR) spectra records.

The user interface was implemented using a GUI toolkit implemented in MATLAB (MathWorks 2014a), shown in Figure 3.

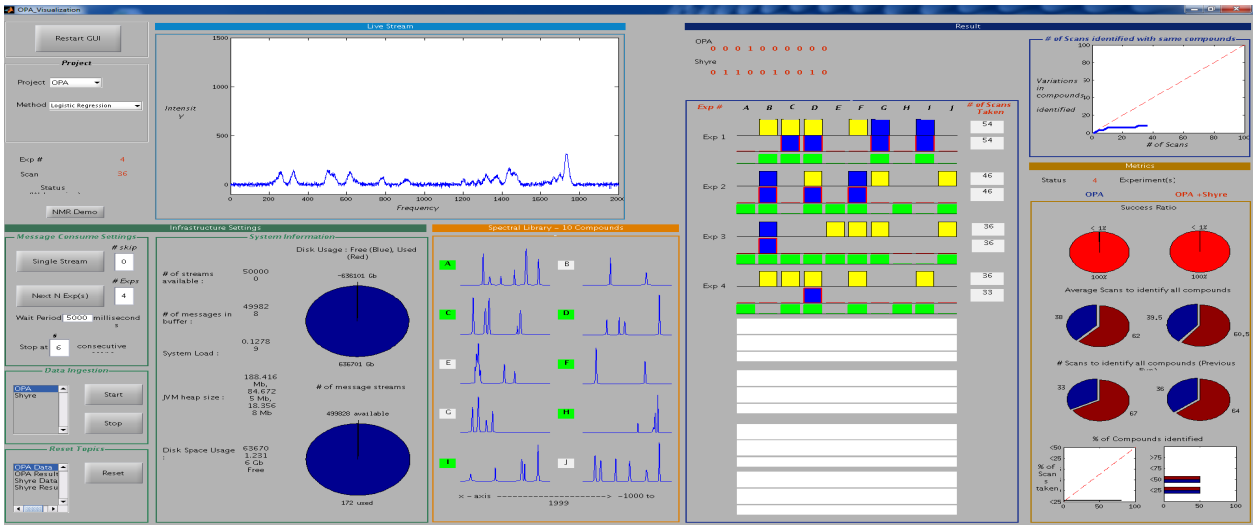
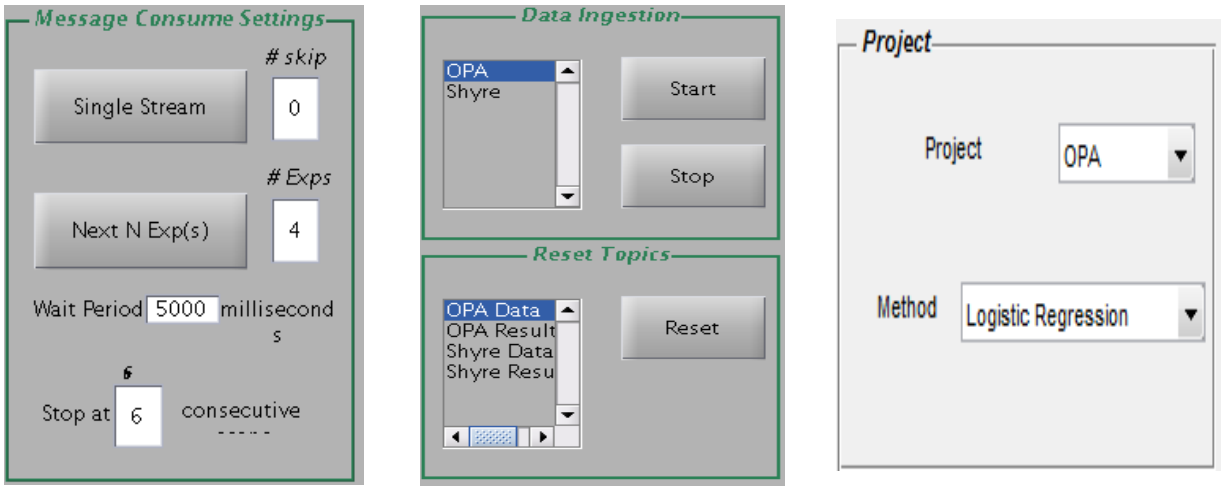


Figure 3. GUI for Testing Range

The tool has four major components:

1. Infrastructure Connect: Enables the user to select data stream characteristics, administer the data streams, and select the models to be tested (Figure 4).



(a) Message Consumption Settings

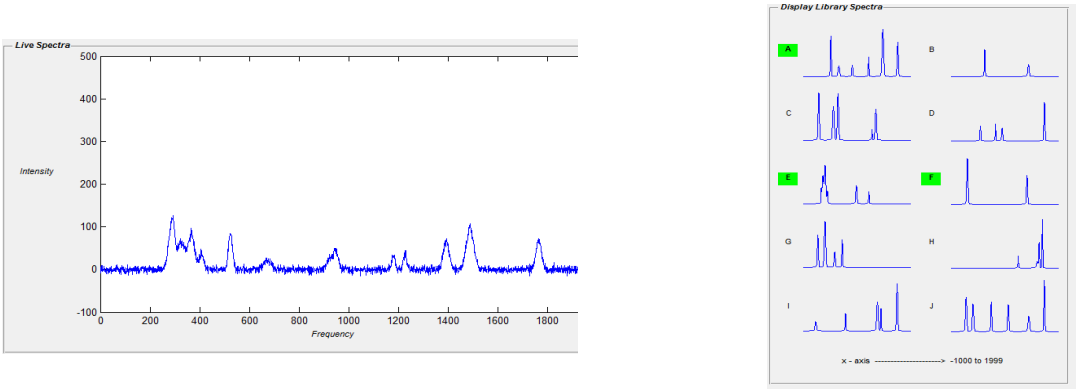
(b) Data Ingestion or Resetting of Topics

(c) Algorithms And Models

Figure 4. Infrastructure Connect

As shown in Figure 4, (a) allows the user to select the number of data streams, the number of overall messages to be streamed (by selecting the number of experiments to be streamed), speed of the data stream (by selecting the waiting time between messages), and the success condition for each experiment; (b) allows users to start selected data streams as well as to reset the message topics (i.e., deleting all messages from previous tests that might still be in the MQs); and (c) enables the user to select the models to be tested.

2. Data Streaming: The tool displays content for each stream of incoming messages (NMR spectra) in the live streaming window and ground truth for the current experimental sample being streamed to the models as shown in Figure 5 (a). For the latter we display the list of all possible compounds that could be in our test examples as shown in Figure 5 (b). This particular example set had a maximum of ten compounds (A-J) per experiment. The compounds present in the current stream are highlighted in green.



(a) Live Incoming Spectra

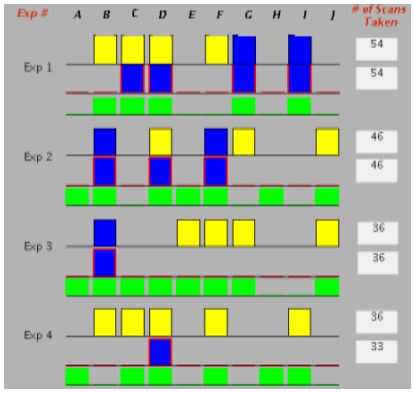
(b) Library Compounds Used to Generate NMR Data

Figure 5. Data Streaming

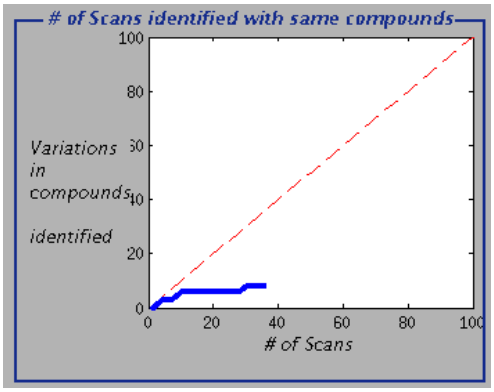
3. Results: The results are presented in two ways.

In Figure 6 (a), each experiment has three rows and ten columns, one each for each possible compound. The bottom row for each experiment shows the compounds actually present, the middle row shows compounds only predicted by model 1, and the top row shows compounds that have been identified by model 2. The graphics uses three colored bars. The green bars represent the compounds that are actually present in each incoming stream, the yellow bars represents compounds identified by one algorithm, and the blue bars represent compounds identified by both algorithms. Two blue bars denote that the compound is most likely present and a blue/yellow combination suggests that more validation is required to confirm the presence of that compound. Usually each experiment will consist of 100 scans; the aim is for the algorithms to minimize the number of messages they are required to process to identify all present compounds correctly. The box to the right of the middle and top row per experiment provides the number of scans that they required to complete the analysis.

The *'Variations in cpds identified' vs. '# of scans'* plot shown in Figure 6 (b) represents the number of consecutive messages with identical results by a horizontal line.



(a) Results for Multiple Experiments

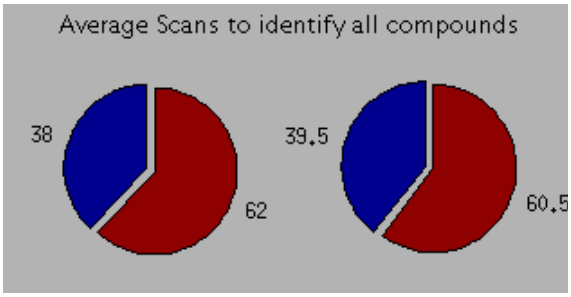


(b) Plot for Variations in Compounds Identified vs Number of Scans

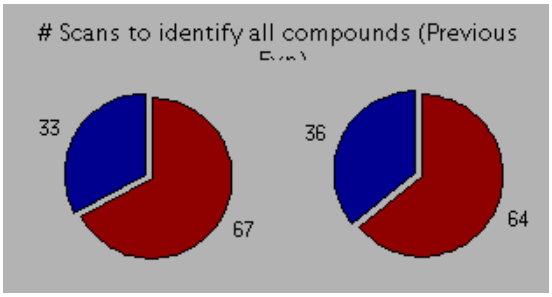
Figure 6. Graphical Results

4. Metrics: Some basic metrics are generated and stored for each full test run. The average percentage of successful identification of all compounds present from all experiments in the current test run is represented in Figure 7 (a). The average number of scans taken to identify all compounds is displayed in the pie charts in Figure 7 (b) for the current experiment and (c) for all experiments. The horizontal bars in Figure 7 (d) represent the average amount of compounds identified at < 25%, <50%, <75%, and >75% where the red bar is for the current experiment and blue bar is for the average of previous experiments.

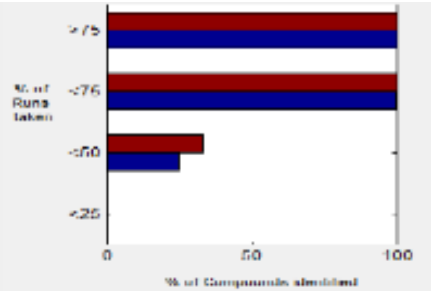
(a) Average Success Percentage for 'n' Experiments



(b) Average Scans to Identify All Compounds Current Experiment (right)



(c) Average Scans to Identify All Compounds for All Experiments



(d) Percentage of Compounds Identified

Figure 7. Metrics

4.5 Avro Container

The Apache Avro data serialization system was selected as the messaging container for use by the AIM infrastructure. A key feature of Avro is that each message includes the message schema as well as the message itself. Avro's message data model is based on JavaScript Object Notation (JSON) and can be represented as either JSON or in a compact binary form. Avro provides a sophisticated schema language, also in JSON, used to describe message data structure. Avro has many benefits including direct mapping to and from JSON, compact binary representation making it efficient for high-volume usage, multiple language bindings for client development, a robust schema language (also in JSON), and support for schema evolution.

Schemas are a critical feature within Avro, conceptually similar to a relational database table schema, affording producers and consumers the capability to exchange message data knowing they are correctly formatted. Avro schemas can also be self-describing; they can provide semantics of data fields listed in the schema through use of their "doc" field. Also, unlike other popular messaging systems (e.g., Thrift, Protocol Buffers), Avro provides dynamic typing. That is, Avro does not require data access codes to be generated at build time based on the schema definition. This enables development of a generic data processing framework using Avro's API to reference the schema at runtime.

For AIM, Avro was the ideal message format choice as it provides the means for flexible model coupling developments due to its language independence and the possibility for the consumer to interpret

and use the message as it arrives. Using Avro in this way proved a means for application developers to prototype messages with JSON and quickly generate, receive, and analyze messages from other applications through the interface. However, this approach does represent a cultural shift for most developers who are used to working on their own or are familiar with messaging methods such as MPI, where message format and interpretation have to be discussed and agreed between developers rather than be available in the message.

4.6 Initial Deployment Environment

The first version of the AIM server software accessible to internal PNNL clients was hosted on a single Red Hat Enterprise Linux 5 node provided by PNNL Institutional Computing (PIC). [8] This host has two 16-core Advanced Micro Devices (AMD) Opteron processors (32 cores total) operating at 2.1 GHz, 64 GB of random access memory (RAM), and a one-terabyte local disk allocation with sustained read speeds of 125 MB/sec. In addition to local storage, the system also has access to several common file shares accessed via a parallel Lustre file system over a 40GbE high-performance QDR InfiniBand interface. [10]

AIM server software deployments for this host configuration consist of two phases: the initial configuration and launch of the Kafka messaging infrastructure via Puppet scripts, followed by update of the application web archive (WAR) file within the Tomcat server with Java code hosting web services and data producer routines. Kafka bootstrapping is executed manually via shell scripts and, under normal operating conditions, no additional work is required. The Puppet scripts handle ZooKeeper configuration, Kafka broker start up, as well as the initial creation of test and validation topics to assist with debugging and confirmation of successful installation of the messaging system. The Tomcat server is configured to monitor its web applications folder continuously and to reinitialize applications once their source WAR file is updated. The WAR file for AIM is deployed via the one-button deploy Jenkins build system using the Subversion repository as its source. Jenkins build jobs encapsulate system access credentials, provide environment configuration, and perform scripted tasks that automate package updates.

For the initial deployment of the AIM server software, we implemented four preprogrammed data producers that AIM clients could activate to populate Kafka MQs with known test values. Data sets containing validation data were generated and stored as a collection of comma-separated-value text files on a shared network drive. The web services management frontend can intercept client requests for starting, stopping, and resetting the data stream, and issue the corresponding commands to Kafka on the client's behalf. Additionally, this first version of the AIM stream interface enabled clients to access messages both in a sequential manner (retrieve next message or skip *N* messages and retrieve the following message, etc.) as well as the ability to *window* the stream (select a subset of the streamed data and process the information using a batch-oriented approach). Figure 8 illustrates the key functional modules for the first version of AIM server software as deployed on the PIC head node.

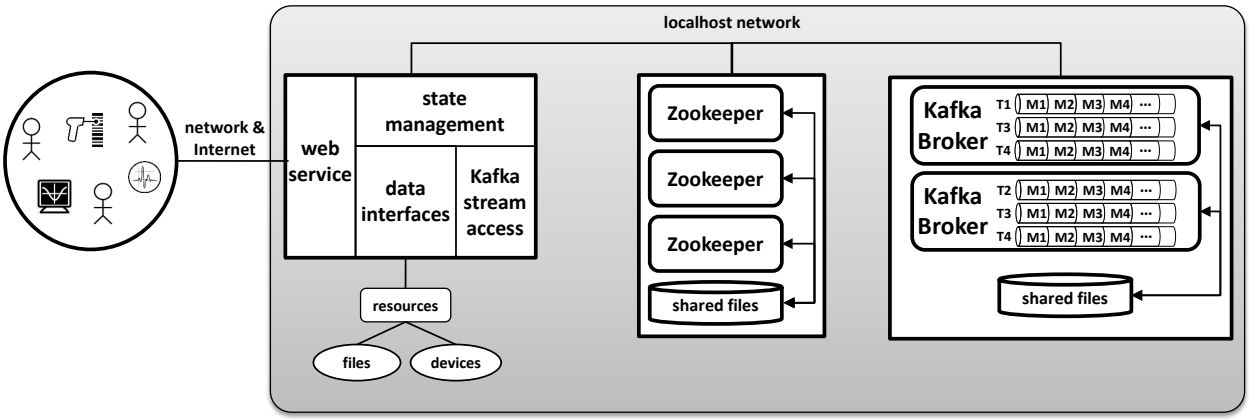


Figure 8. Functional Block Diagram of AIM Server Software on the PIC Head Node

4.7 Initial Infrastructure Performance Tests

After the basic Kafka installation we conducted a number of scalability tests to assess if we could achieve a suitable level of performance or where further optimization might need to be applied.

4.7.1 Third-Party Test Results

To get a measure of the performance we should be able to achieve we identified a set of publicly available test results:

C++ (librdkafka) Results

https://github.com/edenhill/librdkafka/blob/master/INTRODUCTION.md#performance-numbers

Performance numbers

The following performance numbers stem from tests using the following setup:

- Intel Quad Core i7 at 3.4 gigahertz, 8 GB of memory
- Disk performance has been shortcut by setting the brokers' flush configuration properties as so:
  - log.flush.interval.messages=10000000
  - log.flush.interval.ms=100000
- Two brokers running on the same machine as librdkafka
- One topic with two partitions
- Each broker is leader for one partition each
- Using rdkafka_performance program available in the examples subdir.

Test results

- Test1: two brokers, two partitions, required.acks=2, 100 byte messages: 850000 messages/second, 85 megabytes/second
- Test2: one broker, one partition, required.acks=0, 100 byte messages: 710000 messages/second, 71 megabytes /second
- Test3: two broker2, two partitions, required.acks=2, 100 byte messages, snappy compression: 300000 messages/second, 30 megabytes /second
- Test4: two broker2, two partitions, required.acks=2, 100 byte messages, gzip compression: 230000 messages/second, 23 megabytes /second.

4.7.2 PNNL Tests on OS X Laptop

Initially we set up and tested our Kafka configuration on a standard laptop (see Figure 9). Our goal was to see how long it would take to send 1,000,000 messages through the system; we used a smaller message size than would be produced by one of our experiments and limited messages to 35 bytes.

Performance numbers

- Processor: 2.3 gigahertz Intel Core i7
- Memory: 16 GB 1600 megahertz DDR3
- Software: OS X 10.9.2 (13C64)

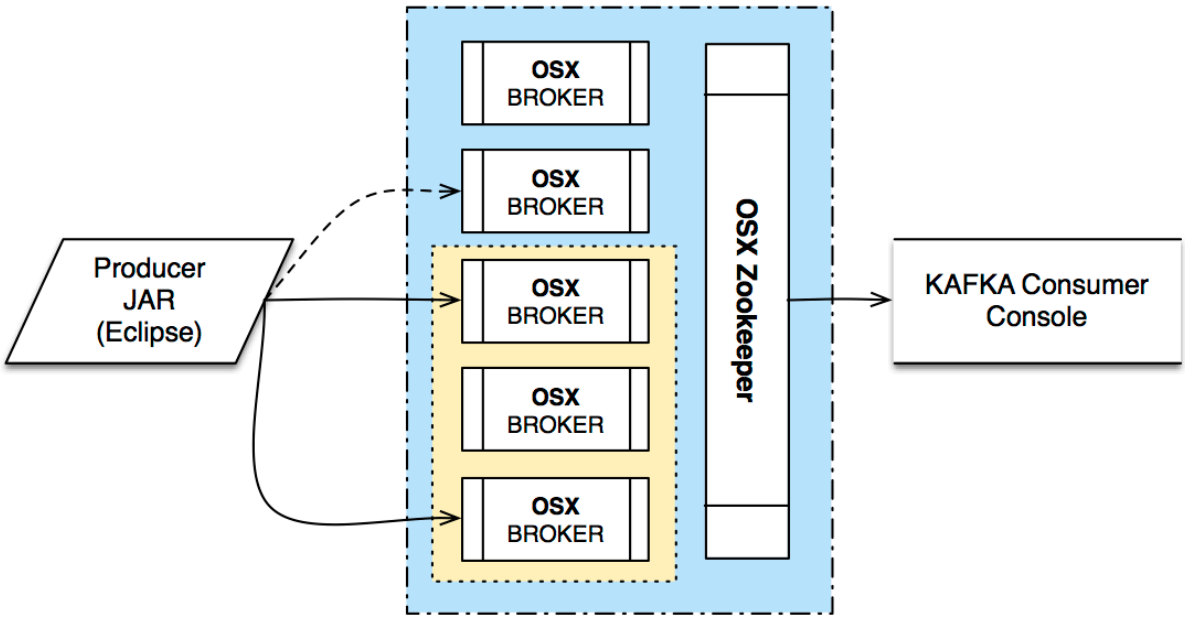


Figure 9. AIM Initial Kafka Test Configuration on Standard Laptop

Test results

- Test 1: three brokers, 15 partitions, three replications: 1 million messages: 272 seconds

- Test 2: three brokers, four partitions, three replications: 1 million messages: 247 seconds
- Test 3: three broker, four partitions, five replications: 1 million messages: 199 seconds
- Test 4: 5 broker, 15 partitions, five replications: 1 million messages: 447 seconds
- Test 5: five broker, four partitions, five replications: 1 million messages: 410 seconds
- Test 3: five broker, four partitions, two replications: 1 million messages: 212 seconds.

4.7.3 PNNL Test on High-Performance Computing Cluster Head Node

Our target deployment environment was our local high-performance computing (HPC) cluster, a 20,000 core cluster composed of regular, fat, and HPC nodes. We used one node of the system including 32 central processing unit (CPU) cores, 64 GB RAM, Lustre file system support, and InfiniBand networking. The Kafka configuration is presented in Figure 10, the related performance results are shown in Table 3 (five broker, synchronous, acknowledgement from broker required, 7-day queue storage, sending 1 million messages), and the optimized results are provided in Table 4 (five brokers, synchronous, no acknowledgement, 60-second queue storage, 1 million messages).

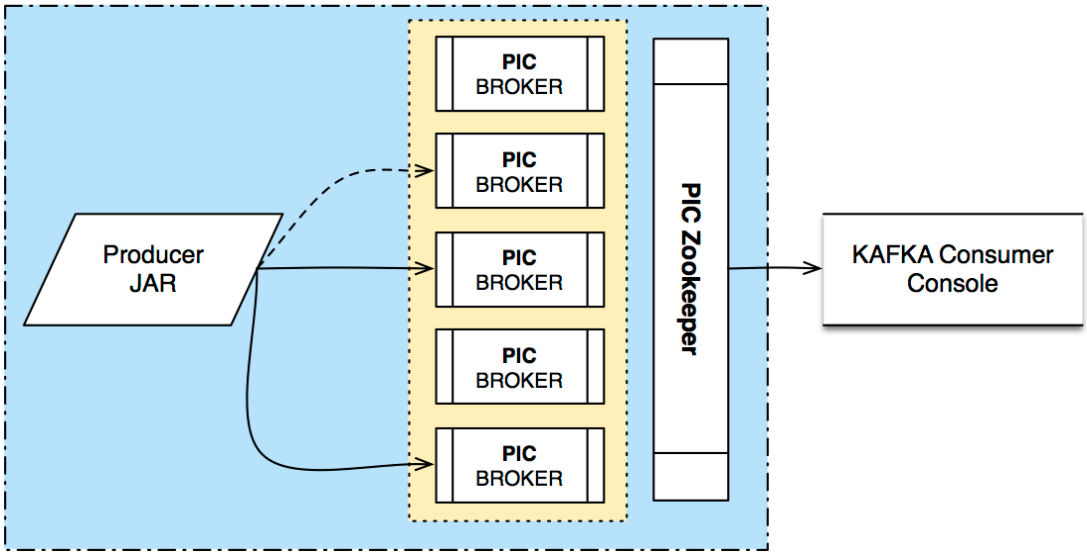


Figure 10. AIM Initial Kafka Test Configuration for Target HPC Environment

Table 3. Performance Test Results

Replication	Partitions	Run 1 (s)	Run 2	Run 3
5	15	1144	1075	1110
4	5	844	945	733
4	1	814	838	768
3	15	643	645	676
3	4	644	663	706
3	2	695	656	599
2	4	629	606	639
2	1	658	633	551
1	1	434	491	503

Table 4. Optimized Test Results

Replication	Partitions	Run 1 (s)	Run 2	Run 3	Throughput Avg. Kb/s
5	15	288	228	265	44.81434059
4	5	209	221	245	51.85185185
4	1	205	213	217	55.11811024
3	15	184	183	215	60.13745704
3	4	208	183	182	61.08202443
3	2	178	181	182	64.69500924
2	4	172	179	181	65.78947368
2	1	177	174	174	66.66666667
1	1	168	164	162	70.85020243

4.7.4 Tests after Further Optimization

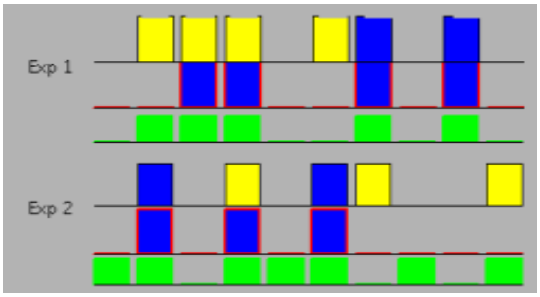
Based on the initial configuration test matrix, we selected the settings that resulted in the highest throughput (i.e., no data replication with a single broker) and repeated the message-publishing test over a 2-hour period. The sustained measured rate with the 35-byte test payload on a single node was 600K messages per second. The observed high-performance of the direct interface to the Kafka messaging subsystem in this configuration represents a useful baseline to help drive additional AIM design work. The next section discusses some of the challenges and issues we identified during integration of Kafka with the AIM web services.

4.8 Initial Model Integration and Use Case Tests

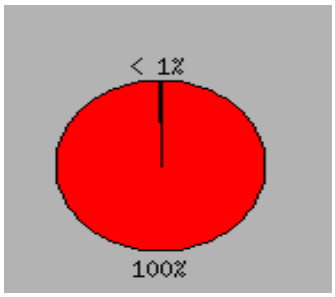
At the end of the initial AIM infrastructure deployment phase, we were able to successfully integrate two models into the infrastructure. Once the integration and model testing were completed we carried out an end-of-deployment performance test. We selected our NMR use case as test scenario; the limiting factor in these experiments was the number of scans or repetitions of each experiment. Each scan is a representative summation of all the previous scans and hence produces a cleaner spectrum with improved signal-to-noise ratio. The challenge for the algorithms was to identify the compounds present in each experiment within a fixed set of scans. We did two rounds of 100 scans and 65 scans. If all compounds for an experiment were identified within the set number of scans, the GUI would automatically skip to the next experiment and repeat the process. Each incoming stream was simultaneously passed through a single algorithm (OPA) and a combination of two algorithms (OPA and SHyRe) to identify the benefits of using a single algorithm vs. multiple algorithms in tandem.

Out of the 150 experiments, we determined the algorithms, both individually and in combination, were able to identify all compounds present with 100% accuracy in an average of 75 scans. The benefits of using a combination of algorithms were evident when the number of maximum scans available for the algorithms was limited to 60. For almost all experiments, a single algorithm alone was unable to identify all the compounds present.

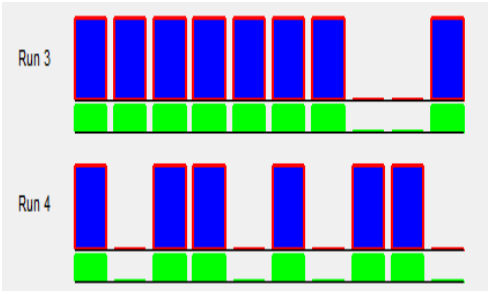
As shown in Figure 11 (a) and (c), executing the algorithms in tandem resulted in some compounds being identified by both algorithms (blue bars) and some identified only by one (yellow bars) as possibly present/absent, while green represents actual results. The success ratio shown in Figure 11 (b) and (d) are biased in that the result is counted as successful only if all compounds are correctly identified.



(a) Limited to a Maximum of 60 Scans



(b) Success Ratios for Case a



(c) Limited to a Maximum of 100 Scans

(d) Success Ratios for Case c

Figure 11. Results of Two Experiments

The infrastructure performed well during the tests, delivering the messages reliably and responding quickly to the interface commands. We found that message throughput rates were limited by the algorithms and not by the infrastructure. The OPA model required less than 0.8 second per message, so could process between 75 and 100 messages per second. SHyRe was limited in their speed by the Pellet reasoner they were using, and required 10-15 seconds per message, thus had to employ a much more stringent sampling approach to keep up with the message stream.

4.9 Conclusion and Final Setup

Performance results and message throughput measurements obtained with the direct interface to the Kafka messaging system represent an optimal goal state for Kafka's integration with the rest of the AIM software. Once integrated with the web service components of AIM server code, AIM clients no longer have direct access to Kafka topic streams. Instead, data upload and download are facilitated via HTTP REST endpoints using AIM-managed resource locations and application-specific options. The tradeoff of this approach is a simpler, more robust interface for the client code, which helps focus the AIM interface code development on solving the science problem without exposing technical aspects of the underlying infrastructure. The cost of this simplification is a loss of efficiency; client requests are issued to the AIM server software running as a web services application inside a Tomcat server. The AIM server software maintains state information about client requests and provides an interface to the Kafka brokers for data access. The overhead of HTTP-based communication, the extra layer of indirection, and the network backed data store for shared Kafka log files are some of the main reasons why the web services implementation is less efficient than the performance evaluation configuration described earlier.

In addition, we have identified ZooKeeper's sensitivity to network latency as a contributing factor to system instabilities and software faults that have caused periodic instability in the AIM-to-Kafka interface in production. Monitoring system logs on the PIC head node server revealed periodic network timeouts on the InfiniBand link that supports the file shares. The operating system has sufficient error handling to prevent catastrophic data loss, but the latency on input/output access times would sometimes jump by several orders of magnitude. Normal users of the system would see this manifested as exceedingly long response times for file system commands like 'ls'. For Kafka and ZooKeeper, it would appear that the system services have become unresponsive and the connection would be terminated, resulting in the perceived downtime of the system. Furthermore, the first version of the AIM server software lacks the ability to actively poll Kafka configuration to detect such "under the hood" failures; thus from the standpoint of the client, the original request is still active, resulting in an inconsistent state between all of the various system components. Recovery in this case consists of explicitly shutting down the client connection and reinitializing the Kafka and AIM interface parameters. However, since this is a manual process involving intervention by a system developer, the overall service recovery time varies depending on the time of initial error detection and developer availability.

Another issue that had an adverse impact on the client experience with the AIM-to-Kafka implementation was the complexity of the low-level Kafka API that permitted advanced Kafka functionality, such as rewinding and windowing of stream data (as opposed to the simpler "move forward-only" access). For example, our testing identified a problem with the Kafka interface, trigged by a client reading past the end of the stream causing a serious error. When this condition occurred, the internal Kafka stream access data structure was marked invalid, yet the failure remained invisible to the higher-level AIM web services code. As a result, subsequent requests for messages using the same stream access structure would fail silently, giving the appearance that no more data were available in the stream. Similar to the ZooKeeper issue, recovery involved a manual reset of the entire system, affecting all of the AIM topics, not just the one with the error. Without a clearly defined and requested requirement for random stream access, it was our recommendation to limit future revisions of the Kafka stream reader interface to the simpler approach of forward-only stream access.

5.0 AIM Infrastructure Deployment Phase 2

After the deployment and evaluation of phase one, the overall AIM Initiative increased its research activities in phase 2 (October 2014 – April 2015) to nine model development efforts, with a view to supporting additional summer student projects as well. To support this increase, the infrastructure needed to evolve further, to provide:

- Multiuser support
- Load balancing
- Redundancy
- Unrestricted topic partitioning
- Provenance and metrics capture

In the following sections, we will describe our development efforts for phase two.

5.1 Multiuser Support

As described earlier, we identified several stability issues caused by ZooKeeper availability as well as advanced Kafka stream manipulation that was difficult to track in the higher-level AIM interface code. At the same time, the project demonstrated a growing requirement to support concurrent users, allowing them simultaneous access to shared data streams and in need of a coordination/cooperation mechanism to enable shared data flows between dynamic, interacting components. Taking this as an opportunity to apply lessons learned from the first iteration, the infrastructure team refactored the AIM server software implementation to improve robustness and deliver a more consistent and stable environment to the system users.

The first improvement was the integration of the AIM system with the PNNL LDAP and AD services. This allowed users to authenticate with the AIM infrastructure using their regular network credentials. Internally, the AIM software appended network user names to client requests for data when creating Kafka topics, ensuring that each client had a dedicated MQ and could request and process data without interfering with other users in the system. The fact that user network credentials were being used in a web service environment meant that the AIM system needed to protect the data as they traveled through the network. Therefore, we modified the server configuration to force TLS encryption on all incoming connections. The AIM REST endpoints were upgraded to use HTTPS and the Tomcat server was configured to use the PNNL Secured Sockets Layer (SSL) certificate. Integration with AD and HTTPS, data encryption were implemented within the LIFT base software framework and these innovations were contributed back to the LIFT codebase to make the same functionality available to other PNNL projects using LIFT for their web services infrastructure.

The timing of these changes was also favorable to upgrade the Java runtime environment from version 1.7 to 1.8 (or *Java 8* as it is sometimes referred to by Oracle). This new version of the Java development platform offered a redesigned concurrency system that AIM used to simplify management of multiuser request queues. In particular, the AIM infrastructure used a job stealing executor to encapsulate AIM user requests for Kafka data. This allowed the AIM infrastructure to manage

concurrency in a structured, asynchronous manner with a significant improvement in error detection and handling capabilities. The new interface served as a natural approach to encapsulating Kafka read primitives so that failures in the underlying stream would result in an exception report in the main AIM software, providing a powerful means for reporting problems back to the user. However, to realize these advantages the design required greater tolerance for latency (as the requests were queued up and scheduled by execution internal to the Java virtual machine [JVM]). Furthermore, this approach still relied on a single AIM application to keep track of user requests, thus limiting both scale and robustness of the overall system. Scalability and stability are the two characteristics that the team has identified as key target areas for follow-on improvement.

5.2 Scalability and Stability Measures

Drawing on inspiration from the current HPC trends in both research and industry, we have identified migration to a cloud-based architecture as the most effective and practical path forward for future AIM development. The cloud-based approach, which in this context can be thought of as a collection of self-contained services with the ability to scale both vertically (individual node capabilities) and horizontally (number of nodes) to handle increasing volumes of data, addresses the key problems and challenges we identified in the earlier revisions of the AIM infrastructure design. For example, issues with network resource latency are addressed in the cloud via distributed file repositories that provide built-in replication and redundancy so that intermittent network problems with a single file server node can be overcome by accessing a copy of the data from a replicated backup. For these cases, the distributed data store typically employs a *sharding* (i.e., partitioning) technique that ensures high-speed parallel access for multiple clients. In the cloud, network resources in general are accessed via a pool of load-balancing gateways that actively monitor the network resources and route client requests via the optimal path. Failures of individual nodes do not result in complete failures of the system because their failure is mitigated by peer nodes that provide temporary relief coverage while the affected resource is repaired and restored.

Because each node must provide for a graceful failover, it is important they carry only the minimal state information in their local cache. That way, when a failure occurs, one of the peer nodes can pick up the same task in a manner that is transparent to the client. Out of the box, ZooKeeper installation provides support for such a high-availability scenario (see Figure 12 below). ZooKeeper uses a *quorum*; a coordinated group with a dynamically elected lead node and a set of followers that replicate the leader's data. [12] Information can be read from any ZooKeeper server in the quorum, leading to direct performance gains with distributed applications. New information is written to the lead node only and then replicated to the follower nodes via several tunable mechanisms. While the second revision of AIM server software still keeps all of its state management data in local memory, the refactoring of the software to support multiple users helped define the boundaries of this shared state information. Encapsulation of the AIM global state in the modern Java concurrency interface should aid in converting this solution into a truly distributed, stateless, cloud-ready application.

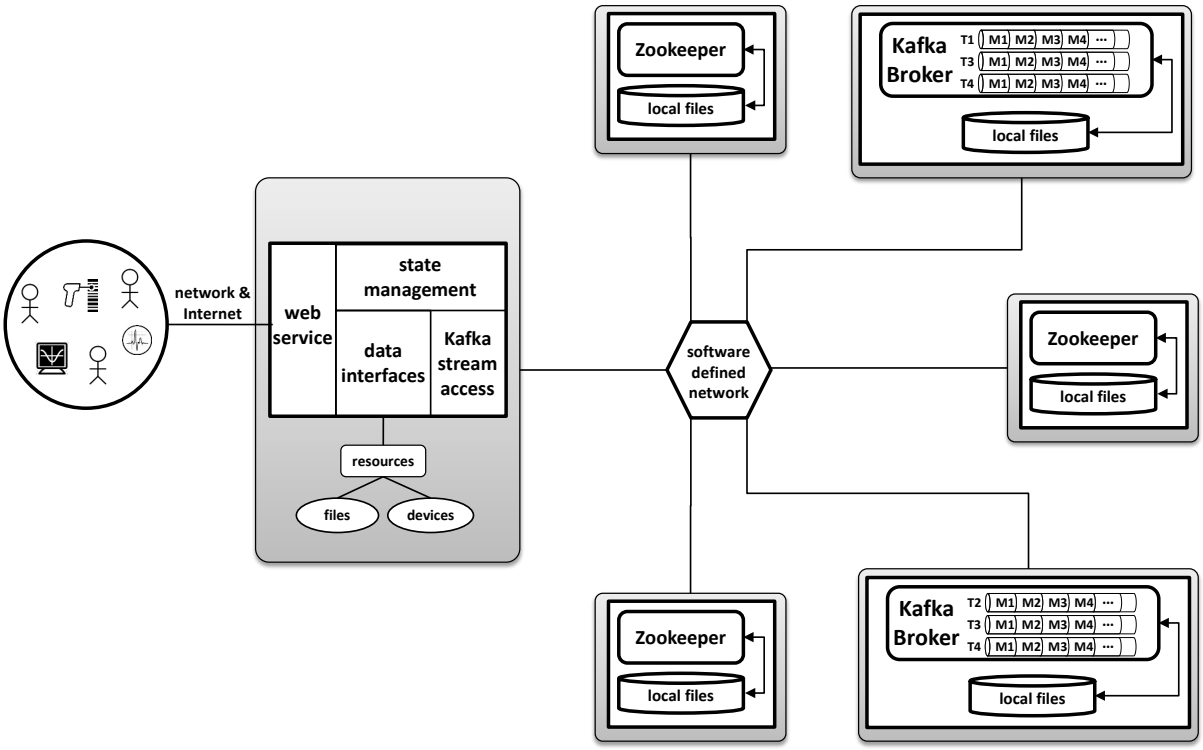


Figure 12. Second Revision of AIM Server Software with High-Availability ZooKeeper Configuration

5.3 Provenance and Metrics Capture

AIM has a range of key targets to achieve with its full system (infrastructure and models). To monitor how individual projects and the initiative as a whole are progressing towards achieving these goals, it will be necessary to define, capture, and evaluate a number of crucial success metrics, some of which will be captured by the infrastructure. The key hypotheses we want to test within AIM are the following:

- Insight generation
  - Goal: AIM will allow humans to use streams to develop correct interpretations of the world, with reproducibility across different users.
  - Insight is a tradeoff between utility, throughput, and accuracy.
- Throughput
  - Hypothesis: AIM will ingest streams at a rate sufficient for the problem domain.
  - Hypothesis: AIM will yield judgments at a speed sufficient for the problem domain.
  - Metrics captured by infrastructure: message throughput rate per model, basic and complete infrastructures, model combination solution, time to solution required.
- Accuracy of algorithms
  - Hypothesis: AIM systems will converge to correct interpretations under two gold standards (compared to the known state of the world as reflected in the data and compared to reference static analytic algorithms running over the total data).

  - Hypothesis: F1 (precision/recall) measures will be greater than with algorithms alone or humans alone.
  - Metrics captured by infrastructure: model-specific results, model combination results, model results with human input, ground truth, results from static algorithms on the same test case, calculated deviation score.
- Utility of AIM's output
  - Hypothesis: AIM will provide stream interpretations that usefully support insight in its users.
  - Utility of human input in AIM (vs. purely algorithmic streaming classifiers).
  - Hypothesis: Users will be able to usefully guide streaming classifiers.
  - Hypothesis: Correct human interpretations will occur earlier in the stream with AIM.
  - Metrics captured by infrastructure: human steering input into the system; number, content and timing, correlate with accuracy results, capture human insight; conclusion, timing, accuracy against ground truth.

Next to capturing the pure metrics, we want to enable infrastructure and model developers to identify root causes for changes (positive and negative) in their performance to aid their further progress. In consequence, we have decided to develop an extended provenance system that captures what has been run, when, and where, and combines this information with the metrics listed above. As performance could be influenced not only be models and the AIM infrastructure, but also the utilized execution environment and system architecture, we decided to develop a comprehensive workflow performance provenance model to capture all aspects of performance and the inherent interdependencies. Furthermore, we are currently developing a high-performance, compact, provenance capture system that is directly integrated into the AIM infrastructure. The resulting provenance is stored in a scalable archive that will enable users to easily evaluate and explore their performance details. Our provenance work will be described in separate publications in more detail.

6.0 Future Work

6.1 Cloud/HPC Plans

The next task on AIM infrastructure team's roadmap is conversion and deployment of the current single "head node" implementation of the server software into a full-fledged cloud architecture, with exciting and useful features such as dynamic load balancing (also known as *elastic scaling*) and high availability (e.g., the system will have instant failover, with no impact on the client). Cloud architectures are at the forefront of current computational science innovation and the rapid growth in this area comes from the union of mutually reinforcing schools of thought. Hardware virtualization, or the ability to "emulate" self-contained computing units within other operating environments, forms the foundation of any cloud-based effort. Software-defined networking is the second foundational technology that works alongside virtualization to enable complete ecosystems of networked virtual hosts.

By relinquishing direct control of the hardware, a cloud-managed application may take advantage of powerful, high-level interfaces and services provided by the virtualized environment. The AIM infrastructure team worked closely with the new PIC Research Cloud personnel to ensure close alignment of future collaboration efforts to ensure maximum leverage of existing investments. AIM is currently in the process of migrating all of its software for both infrastructure and individual science project teams to the PNNL OpenStack-based cloud platform, as diagrammed in Figure 13.

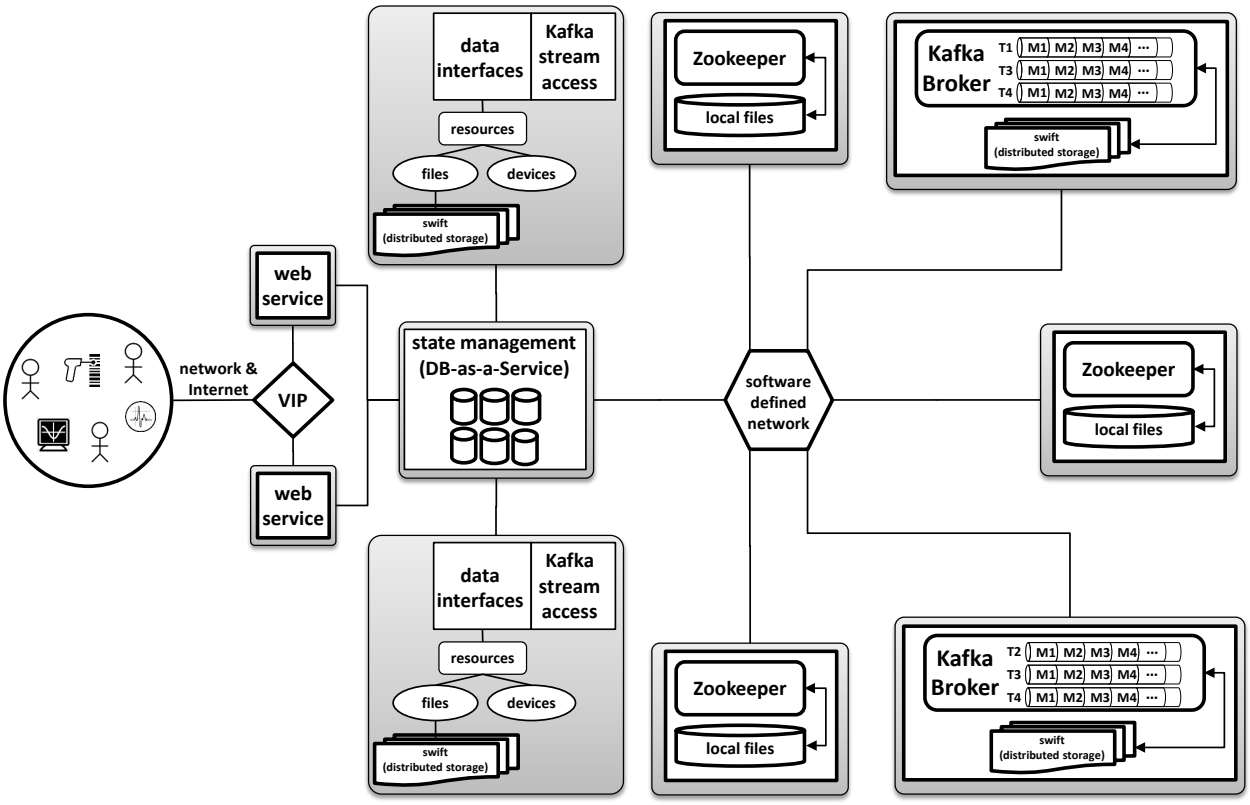


Figure 13. Functional View of a Cloud-Ready AIM Infrastructure Software

OpenStack is a union of both open source and commercially supported software with the broad goal of building a production scale cloud environment compatible (and competitive) with the industry leading Amazon web services products. OpenStack is a thriving community effort supported by a global group of volunteers alongside established enterprises, such as IBM, Intel, Hewlett Packard, RedHat, and AT&T. There are official releases of the core technology maintained by the project, along with many supporting utilities hosted on GitHub and similar community development platforms. At PNNL, the cloud installation already provides the principal functionality: software-defined networking with advanced load balancing and virtual internet protocol management, multiple Linux operating system virtual machine images, a distributed data file system called Swift, database-as-a-service layer called Trove (supporting both relational and non-relational data models), and a customizable solution for distributed data processing service called Sahara.

Abstraction of hardware and the network into fully virtualized entities yields the important operational advantage of being able to accurately "snapshot" the state of the system, pause and resume processing, modify the physical location of virtualized resources, and collect in-depth metrics on system performance to help drive improvements and continuous redesign. Real-time metrics are an important tool in ensuring availability of the system by helping to detect and mitigate anomalous system performance. As soon as abnormal operating parameters are detected, another instance of the affected resource can be created and deployed into service, while the malfunctioning component is quarantined or recycled. In OpenStack, this functionality is implemented via a component called Ceilometer. The tool makes use of a special configuration construct called a *Heat orchestration template* that provides a blueprint for a cloud-enabled virtual machine or service. OpenStack management software uses these template files to dynamically create and allocate new resources in response to increased data loads. The ability of a properly written OpenStack software system to scale its processing capability on demand is an important motivating factor for steering future AIM design and development toward the cloud architecture.

Because the existing AIM infrastructure is designed as a web services platform, extending its current implementation to fit a cloud environment is a straightforward task. As explained earlier, decentralization of internal state-keeping is the first improvement that is needed to enable the transition to the cloud. In the OpenStack implementation, this will consist of using a distributed database to keep track of client requests, along with a distributed data store for shared files and other resources. The second step is ensuring that all AIM services are horizontally scalable; the software design must allow for elastic addition and removal of compute nodes without imposing artificial restrictions on the number of services capable of processing data.

6.2 Scalability Research

A key goal for AIM is the analysis of high-velocity, high-volume data streams, such as those expected from experiments like the Dynamic Electron Transmission Microscopes, which are set to create 1M images/sec, which would equal 1-2 terabytes per second depending on image size. To achieve throughput on that scale we need to investigate two key questions:

- How to scale the basic AIM infrastructure to support data rates of that magnitude? and
- How best to scale applications in the infrastructure to enable these to work on such data streams?

Over the coming months we will carry out a wide range of scalability tests to identify possible bottlenecks, research viable optimization strategies and investigate the potential tradeoffs between an easy

to use infrastructure implementations and high performance. One key challenge identified early on is the requirement of our analysis models to receive messages in the same order they were sent, which is not natively guaranteed by Kafka in the high-performance, multi-partition configuration. We will investigate how we can provide this functionality, without significant performance impact. Furthermore we will evaluate the impact that larger numbers of users could have on the infrastructure performance. Finally we will investigate optimization strategies for different classes of model algorithms that will allow us to reach the required throughput performance.

6.3 Adaptive Workflow

The AIM analysis paradigm expects to create unique analysis model combinations for each class of problems addressed, furthermore it is anticipated that the system will need to instantiate and retire analysis models at runtime depending on user feedback and the events observed in the streaming data. To facilitate such additivity in workflow creation and change we need a light weight, easy to use, flexible and scalable workflow description and execution environment. At present we are not aware of a system that would satisfy all of these requirements. It is our intention of the coming years to investigate existing solutions and either build on them or develop our own high-throughput, highly flexible workflow implementation for AIM.

6.4 Community Support

Any discussion of future work on the AIM effort must emphasize the important role that the open source community and various projects play in influencing the design and evolution of AIM. On one hand, many technical components of the AIM infrastructure software trace their origin to an open source project, such as Java Spring and Spring Integration for simplifying web service implementation, Kafka as the key message-processing framework, or the OpenStack project that offers an integrated cloud-management environment. Complementing the resources of the open source software community is the active research and development carried out by the HPC practitioners, who offer advanced algorithms and mathematical models needed to use the increasingly large amounts of data. By exploiting the best ideas and effectively applying lessons learned by both communities, the AIM Initiative plans to innovate and contribute to the state of the art in these fields.

To illustrate the benefits of prudent concept reuse from both of these sources, it is helpful to consider the Dynamic Transmission Electron Microscope use case and its associated data volume. To produce such large amounts of data in a short period of time, a massively parallel hardware infrastructure must be present on the instrument side. Given the constraints of electronic system integration, plus the need for real-time response, the high-rate instrument is likely to use specialized hardware specially tuned for this one, specific application. The instrument firmware will not use “heavyweight” technologies such as web services, user authentication, or an embedded, real-time, streaming analytics engine. Because providing these advanced features is AIM software responsibility, both systems must define a common interface to facilitate data hand off.

The distributed (or sharded) data store provided by a cloud computing infrastructure represents an enabling technology for supporting such data sharing. Because the data store is fully distributed, multiple external processes can write data to the cloud storage simultaneously without adversely affecting each other (assuming the network infrastructure can sustain the required data rate). OpenStack provides the

Swift file storage system that meets the design objectives for this data transfer. Once the source data is available in Swift, an Apache Spark (or Apache Spark Streaming) high-performance analytics system can be tasked with running processing, detection, and analysis algorithms in a horizontally scalable configuration (i.e., the more nodes, the better the performance, assuming the network can sustain the data transfer rates).

Apache Spark also offers an in-memory processing option, which helps eliminate performance penalties for the network operations. Commercial vendors, such as GridGain, offer high-performance extensions to the in-memory cluster data repositories that AIM can use to accelerate processing by prestaging the required data in each virtualized node's RAM. [11] Because the various players and stakeholders in this area are strongly motivated to support each other's' software, the overall community benefit is maximized. As a result, we are able to select the optimal mix of software technologies and products for maximum AIM performance. At the same time, because these different technologies still share a similar data interface, costs of maintaining the interoperability are minimal, at least in the short to medium timeframe that we expect these technologies to exist in their present form. In the long term, international standards bodies will formally govern evolution of the cloud architectures and data exchange mechanisms in a manner that is similar to today's web technologies.

7.0 References

[1] Apache Software Foundation. “Kafka 0.8.2 Documentation.” Apache Kafka - a high-throughput distributed messaging system, accessed May 4, 2015. URL: http://kafka.apache.org/documentation.html

[2] Vilwock, W and T Stavenger. “LIFT Overview.” Laboratory Integration Framework and Toolset. PNNL Confluence Wiki, Feb 20, 2015. URL: https://confluence.pnnl.gov/confluence/display/LIFT/LIFT+Overview

[3] Puppet Labs. “Puppet Labs Documentation.” Accessed May 4, 2015. URL: https://docs.puppetlabs.com/

[4] Hovemeyer, DH and WW Pugh. “FindBugs ™ Manual.” March 6, 2015. URL: http://findbugs.sourceforge.net/manual/index.html

[5] Christou, S. “Cobertura.” Jan 11, 2015. URL: http://cobertura.github.io/cobertura/

[6] Birkner, S et al. “JUnit Getting Started.” March 30, 2015. URL: https://github.com/junit-team/junit/wiki/Getting-started

[7] Mahmoud, QH. “Service-Oriented Architecture and Web Services: The Road to Enterprise Application Integration.” April 2005. URL: http://www.oracle.com/technetwork/articles/javase/soa-142870.html

[8] Fielding, RT. “Architectural Styles and the Design of Network-based Software Architectures.” Ph.D. Dissertation, Chapter 5: Representational State Transfer. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[9] Plata, C. “PNNL: Institutional Computing.” Accessed May 4, 2015. URL: http://pic.pnnl.gov/projects/proposals.stm

[10] Seagate Technology LLC. “About the Lustre File System.” April 16, 2015. URL: http://lustre.org/about/

[11] Ivanov, N. “In-Memory Database vs. In-Memory Data Grid: Revisited.” June 9, 2014. URL: http://www.gridgain.com/in-memory-database-vs-in-memory-data-grid-revisited/

[12] Apache Software Foundation. “ZooKeeper: A Distributed Coordination Service for Distributed Applications.” Accessed May 4, 2015. URL: https://zookeeper.apache.org/doc/trunk/zookeeperOver.html

Appendix A

Streaming Framework Features

Table A1. Licensing, Support, and Available Ecosystems

Technology	Licensing	Support	Available Ecosystem
International Business Machine (IBM) InfoSphere Streams	Commercial See Pricing List prices are 1. IBM InfoSphere Streams Developer Edition Authorized Single User License + Software Subscription And Support for 12 months (D0H5BLL) **$4000** 2. IBM InfoSphere Streams for Non-Production Environment Resource Value Unit License + Software Subscription and Support for 12 Months (D0V9ELL) **$20,000** IBM InfoSphere Streams Resource Value Unit License + Software Subscription and Support for 12 Months (D0V9GLL) **$41,000**	Commercial Included in licensing costs Training available See SPL Training. Different courses are available covering several aspects of the programming language and model; prices range from $600 - $2000 per course per person.	Comes with a number of built-in capabilities that are provided by IBM, including data mining, text analytics, predictive analytics, geospatial analytics, OpenCV, statistics, mathematical modeling, and acoustics. Where are graph analytics and processing? **Unknown** Technically, with Streams' ability to execute operations via third-party C, C++, or Java applications, Streams should be able to leverage all existing software in these respective ecosystems. Streams has an Eclipse-based Integrated Development Environment to support both text and graphical editing of Streams Processing Language (SPL) and SPL mixed-mode applications (SPL + Perl). See Mining Toolkit See Deployments and Use Cases
Apache Storm	Open source	Open-source community, hosted by Apache Incubator, with active mailing lists. • user@storm.incubator.apache.org • dev@storm.incubator.apache.org • Freenode: #storm-user Some third-party companies involved in analytics offer training with a proprietary platform: http://hortonworks.com/hadoop/storm/	Apache Storm does not come with built-in analytics. Storm is a lower level message-passing/event-propagation "push" framework that is intended to provide real-time analytic capabilities by executing external third-party code on messages.

Technology	Licensing	Support	Available Ecosystem
Apache Kafka	Open source	Open-source community, hosted by Apache Incubator.	Apache Kafka does not come with built-in analytics. Kafka is a lower level message-passing/event-propagation "push" or "pull" framework, designed as a distributed commit log that is intended to provide real-time analytic capabilities by supporting publish/subscribe semantics from external third-party code.
GridGain In-Memory Streaming	Commercially supported open source As of February 2014, GridGain now comes in two flavors: Enterprise and Open-Source. Enterprise Version • **$5000** for 4 CPU cores or 16 GB RAM, whichever is greater • Annual subscription-based licensing model • Perpetual license also available at ~2.5 times the cost of annual license • Discounts on volume/multi-year contracts • Grace period for functionality after license expires Open-Source Version • Open-source version is new as of the end of February • See http://gridgain.com and http://www.gridgain.org respectively	Open-Source version Support forum: http://stackoverflow.com/questions/tagged/gridgain • Commercial enterprise version http://www.gridgain.com/ • Training: http://www.gridgain.com/purchase/training/ • Quote: http://www.gridgain.com/purchase/get-a-quote/	GridGain Systems, makers of widely adopted open-source software used to build smarter and faster data processing systems within finance, retail, healthcare, telecommunications, government, and other markets, has an ecosystem to natively support customers and services written for Amazon's EC2, Rackspace's OpenStack, and Microsoft's Azure cloud hosting platform. GridGain Systems' Java and Scala based open-source middleware platform allows companies to perform real-time processing and analytics on live big data.
LIFT	Open source	PNNL Internal and open-source community.	LIFT is based on standard enterprise integration technologies and design patterns. The primary purpose is to provide integration with and execution orchestration of any technology written in any language, as long as it is reachable via code or network. Since LIFT uses Java technologies, the ecosystem

Technology	Licensing	Support	Available Ecosystem
			of available software and libraries is extensive and well-supported including instrumentation. LIFT does not come with a suite of analytics, only capabilities to integrate analytics as needed.
Yahoo Big Data Uses: Apache Storm Apache Spark Apache Hadoop Optional: Apache Mesos LIFT	Commercially supported open source Apache Spark is included as a core component of Cloudera Hadoop 5.0 Enterprise Licensing information is available here. Premium support comes with 24/7 availability with 15-minute guaranteed response time for critical issues. Costs unknown. **NEED TO CONTACT VENDOR**	Commercial support available, otherwise open-source self-support Training available Consulting available See Cloudera Training	The Apache Storm, Apache Hadoop, and Apache Spark ecosystem basically covers the entire spectrum of today's state-of-the-art big-data analytics. Storm and Spark are the premier open-source solutions for streaming analytics. They approach the problem in different ways and therefore apply to different types of streaming analytic problems. The one major feature of this technology stack is that every component can run on top of Hadoop/Hadoop Distributed File System (HDFS) and specifically use YARN and ZooKeeper (or Mesos). This means that all of AIM's data and infrastructure can exist in a single, distributed, multi-tiered ecosystem of tightly integrated yet loosely coupled technologies.
NPMs: MPI, OpenMP, and others	Open source Used in a wide variety of supercomputers up to 128K node count.	Excellent support available from vendors essentially included as a component of package management on most deployments.	MPI and NPMs are bare bones, but provide all functionality by use of models such as Hadoop MapReduce, spark, and others. The ecosystem comes with everything needed for scheduling large jobs. For example, all systems are equipped with SLURM (Simple Linux Utility for Resource Management), which is an excellent job manager. The Lustre file system will provide fastest access to data with caching mechanisms for temporal and spatial reuse.

Table A2. Programming Models, Programming Language, and Third-Party Integration Capability

Technology	Programming Models	Programming Language	Third-Party Integration Capability
IBM InfoSphere Streams	Custom SPADE Programming Model Main components of SPADE applications are tuples, data streams, operators, processing elements, and jobs Applications written in SPL are designed around workflows, called topologies, which operate very similar to how topologies operate in Apache Storm (see Storm topologies and tuples). Events are ingested at a “source,” processed through a workflow, and emitted to a “sink” or destination. Stream events can be propagated in a single stream or sent across streams depending on the topology. SPL is supported with many built-in operators in the standard toolkit and special toolkits such as data-mining, geospatial, and database toolkits. The standard toolkit operators are grouped in categories as shown below: • Adapter operators • Relational operators • Utility operators • XML operators • Compat operators See SPADE Programming Model	Primary: IBM SPL Secondary: C, C++, Java The InfoSphere framework needs to be programmed in SPL. SPL is a special streaming domain specific language that supports executing non-SPL languages via "operations" such as JavaOp. InfoSphere Streams requires the IBM Java SE Version 6 SDK.	It is currently how easily existing third-party toolkits can be plugged into Streams. There are some examples of integrating OpenCV, so it clearly must be possible. See Description and Background Example from user exchange: You can link in C++ code in dynamic or shared libraries as well as code that is contained solely within .h files. If you can link it into a C++ program, you can call it from SPL. Java code can also be handled using a Java operator. Virtually any device, sensor, or application system can be defined using SPL, but there are also predefined source and output adapters that can further simplify application development. As examples, IBM delivers the following adapters: • Transmission Control Protocol/Internet Protocol (TCP/IP), User Datagram Protocol/Internet Protocol, and files • IBM WebSphere Front Office, which delivers stock feeds from major exchanges worldwide • IBM solidDB® includes an in-memory persistent database using the Solid Accelerator application program interface (API) • Relational databases that are supported using industry standard Open Database Connectivity (or ODBC) Some of the source and sink adapters

Technology	Programming Models	Programming Language	Third-Party Integration Capability
			included within the Streams product allow the developer to define custom adapters, including custom analytics or operators written in C++ or Java. Existing analytics can also be called from Streams applications.
Apache Storm	Storm uses a one-way parallel execution model where execution is determined by topologies that process streams of tuples (data). Each topology is a graph consisting of spouts (that produce tuples) and bolts (that transform tuples). This supports both "pipeline" analytics as well as "map/reduce" style analytics. Input can be of any type where a spout exists (or can be written) and output can be exported anywhere accessible to Storm.	Primary: Java Secondary: C/C++, Python	Any code that is available via Java Native Access/Java Native Interface (JNI/JNA), stdin/stdout, or web services can be integrated into these systems.
Apache Kafka	Kafka is a distributed commit log architecture supporting a robust message-passing and propagation framework that underlies a number of large-scale real-time analytics frameworks such as LinkedIn's Apache Samza. The programming model is based on a publish/subscribe queue/topic design, where individual clients subscribe to queues or topics and are notified when messages they are interested in are propagating through the system. Kafka also supports additional advanced functionality such as guaranteed message ordering, message replay and more. Output from Kafka is generally consumed by other frameworks such as Apache Storm or Apache Hadoop.	Primary: Java Secondary: C/C++, Python	The list of supported clients, in addition to Java, is available here • Python • Go (AKA golang) • **C** • **C++** • Clojure • Ruby • Node.js • **Storm** • **Scala DSL** • HTTP REST • Jruby
GridGain In-Memory Streaming	Combines distributed stream processing with CEP, advanced workflow	Primary: Java, Groovy, and Scala	GridGain White Paper: "GridGain also provides many general

Technology	Programming Models	Programming Language	Third-Party Integration Capability
	management, windowing, user-defined indexes and more. Current key technologies include: • Programmatic Querying • Customizable Event Workflow • At-Least-Once Guarantee • Management • Sliding Windows • Data Indexing • Distributed Streamer Queries • Co-location With In-Memory Database	Secondary: C, C++, R Other: Python, others	features that make development of distributed applications easier and productive such as zero provisioning and zero deployment model, support for aspect-oriented and functional programming, streaming MapReduce processing, and integration with a **wide variety of third-party projects**."
LIFT	Focuses on two forms of programming model: • Traditional enterprise integration patterns and enterprise service bus • Asynchronous event driven architecture including event/message queuing and propagation LIFT also supports any programming model that is required by an underlying, integrated analytic capability.	Primary: Java Secondary: Groovy, Scala Other: C, C++, R, Python, others	LIFT was specifically designed to support integration of third-party tools and libraries. Any code that is available via JNI/JNA, stdin/stdio, or web services can be integrated into these systems.
Yahoo Big Data Uses: Apache Storm Apache Spark Apache Hadoop Optional: Apache Mesos LIFT	There is not a single programming model for these technologies; they each aim to solve a different problem using a different programming paradigm. Storm uses a one-way parallel execution model where execute is determined by topologies, which process streams of tuples (data). Each topology is a graph consisting of spouts (that produce tuples) and bolts (that transform tuples). This supports both "pipeline" analytics as well as "map/reduce" style analytics. Input can be of any type where a spout exists (or	Primary: Java, Groovy, Scala Secondary: C, C++, R Other: Python, others	Any code that is available via JNI/JNA, stdin/stdio, or web services can be integrated into these systems. The main caveat is how performant the resultant integration type is overall (native vs. shell vs. socket).

Technology	Programming Models	Programming Language	Third-Party Integration Capability
	can be written) and output can be exported anywhere accessible to Storm. Hadoop and HDFS are the underlying backbone of this technology stack. Hadoop is a distributed batch processing framework that uses a MapReduce design pattern to perform analytics. For the purposes of this initiative, we are only interested in HDFS and possibly Hadoop YARN and ZooKeeper, which are technologies that Hadoop uses to manage clusters and job execution. A popular and growing alternative to YARN is Apache Mesos. While YARN is limited to Hadoop-only job execution, Mesos works across multiple different cluster applications including Hadoop. *"Apache Mesos is a cluster manager that provides efficient resource isolation and sharing across distributed applications. Mesos can run Hadoop, Jenkins, Spark, Aurora, and other applications on a dynamically shared pool of nodes."* Spark is an in-memory compute framework that provides a distributed, shared-memory programming paradigm. Its primary programming language exposes functional idioms but they are not enforced, for example imperative programming is also supported (through Java). A number of frameworks exist that support streaming, graph analytics, in-memory database, and more via the Berkeley Data Analytics Stack (BDAS) LIFT could be used for instrumentation, reporting, data and user manipulation/ management.		

Technology	Programming Models	Programming Language	Third-Party Integration Capability
NPMs: MPI, OpenMP and Others	Message-passing programming model MPI is the basic model to be used. An abstraction of distributed shared-memory model can be used with Global Arrays (optional). Each of these programming models works on native hardware, with no virtualization of any other indirection. For performance analysis, there are several software stacks available. HPCToolkit can be used for performance analysis and source code attribution (which parts of the source code are slow). The very generic nature of MPI makes it an attractive and difficult solution at the same time. For example, the user still has to worry about communication and defining compute structure. At the same time, it provides maximum flexibility, which is fairly restricted in functional programming models.	MPI Primary: C, C++ MPI Secondary: Java. While inter-JVM Remote Method Invocation (or RMI) is traditionally used in place of MPI, there do exist MPI solutions: • Reference implementations such as OpenMPI • Open-source libraries such as Message-Passing in Java (MPJ) • Commercial products such as FastMPJ Provides support for shared memory and InfiniBand systems.	The premier flexibility provided by MPI would make it an attractive choice for designing the solutions.

Table A3. Performance Capability, Learning Curve, and Instrumentation

Technology	Performance Capability	Learning Curve	Instrumentation
IBM InfoSphere Streams	High Expected performance based on published use cases. See Deployments and Use Cases Example application 1600 streams, 3.5 data objects/sec. What sets Streams apart is its processing speed. Applications in Streams aim to execute in microseconds instead of milliseconds over unbounded streams of events.	High All development teams will have to implement their solutions using SPADE or provided IBM API.	Medium/Low See Performance Monitoring Framework The current performance monitoring capabilities appear to directly relate to internal job/queue management within Streams itself exposed via web interface. The ability to instrument down to the algorithm level does not appear to be present, at least not exposed in the interface. It is currently unknown if this information is available in a separate API that we can tap into via custom code to develop an appropriate solution for AIM. "Once developed, the applications are deployed to Streams Runtime environment. Streams Live Graph then enables you to monitor performance of the runtime cluster, both from the perspective of individual machines and the communications between them."
Apache Storm	High/Medium "Twitter's stream system, but also used in lots of other places, clocked at 1M tuples per second per node" - *Unknown* Storm has been integrated into online services widely considered to be some of the largest data streams currently available (e.g. Twitter). However, implementation benchmarks are dependent on the task, database intensive vs. CPU, and deployment hardware architecture. **Note:** It is known that Storm's default	Medium Storm is based on Java as a high-level programming language but uses a mix of Java and C/C++ libraries for its underlying messaging system. Storm can interact with existing libraries directly via JNI/JNA or indirectly via traditional networking protocols or stdin/stdout.	Medium Storm logs/nimbus server both run on default ports within the JVM. Hooks between Storm and secondary applications (SAMOA) are being created by independent actors as the project matures. Since Storm uses Java as its primary platform, Storm and Nimbus can both be instrumented with Java Management Extensions (JMX), allowing these technologies to provide very fine-grained instrumentation information via a well-defined API.

Technology	Performance Capability	Learning Curve	Instrumentation
	message size is incredibly small, 140 characters, so benchmarking on larger data sizes in less well known.		
Apache Kafka	High/Medium Kafka performance has small numbers of published results and is used as a backing technology for Storm, so performance is expected to be on-par with Storm. One significant benefit of Kafka is that message sizes are only limited by the underlying hardware. The framework itself was not designed to be used only by Storm.	Medium/Low Setting up a Kafka framework will need to be accomplished by the same team setting up the entire AIM infrastructure, so learning curve for that task is marked as Medium. Using Kafka from a client perspective is straightforward and well documented; client code already exists for multiple programming languages, so the learning curve for client integration is Low.	Medium Kafka runs on the JVM. Since Kafka uses Java as its primary platform, Kafka can be instrumented with JMX, allowing this technology to provide very fine-grained instrumentation information via a well-defined API.
GridGain In-Memory Streaming	High **Tasks** GridGain **341 2,705 33,700** 372,279 338,310350,744 Tasks / Milliseconds. http://java.dzone.com/articles/comparison-gridcloud-computing "In 2008 GridGain was the first Java-based grid computing middleware that was independently tested to **scale linearly up to 2048 processing cores on Amazon EC2 cloud infrastructure**." - GridGain White Paper	Medium Learning curve is Medium due to the diversity of products offered by GridGain and the advanced concepts the software suite attempts to solve. While the pre-configured packages available within the enterprise version of the software may be easy to configure and install, AIM will require specialized programming and training to enable GridGain.	High GridGain comes with a prebuilt management and monitoring application called "Visor": http://www.gridgain.com/visor/ Visor provides a single unified console for operations, management, and monitoring across all GridGain products and for any applications and systems built with GridGain. Visor comes with GUI and command line interfaces delivering an advanced set of management and monitoring capabilities. The GUI version is based on a standalone application and the command line interface version is built on top of Scala REPL providing a fully scriptable and customizable environment.
LIFT	High Rating is dependent on the published performance metrics of the Java Message	Medium/Low This project follows industry standard design patterns as closely as possible.	Medium Instrumentation is Medium because each framework is ultimately running on top of

Technology	Performance Capability	Learning Curve	Instrumentation
	Service (JMS)/MQ framework used inside of LIFT. Currently LIFT uses ActiveMQ. The use of JMS inside of LIFT does not prevent LIFT from handling large numbers of messages (millions per second) or large messages (multi-GB).	The difficulty of integration and performance falls squarely on the developers writing the tools to be integrated.	the JVM, meaning AIM test and performance monitoring infrastructure will have access to JMX, allowing applications to provide very fine-grained instrumentation information via a well-defined API. AIM could harvest and manage this information via LIFT, if desired. All individual analytics will have to write their own instrumentation code and find a way to provide it via JMX. If the algorithms are written in Java or Scala, integration with JMX is straightforward; if written in a third-party library or non-JVM programming language, integration will not be as straightforward. A number of toolkits provide web-based (HTTP) management consoles, but these are specifically designed for system-level management. Commercial solutions are available for monitoring JMS/MQ such as http://www.hyperic.com.
Yahoo Big Data uses Apache Storm Apache Spark Apache Hadoop Optional: Apache Mesos LIFT	High The numbers from Yahoo published 9 months ago speak for themselves: • 100 billion events (clicks, impressions, email content, metadata, etc.) are collected daily across all of the company's systems. • A subset of collected events get passed to a stream processing engine over a Hadoop/YARN cluster: 133K events/second are processed, using Storm-on-Yarn across 320 nodes. This involves roughly 500 processors and	Medium Each system has its own requirements and learning curve. Storm is based on Java as a high-level programming language but uses a mix of Java and C/C++ libraries for its underlying messaging system. Storm can interact with existing libraries directly via JNI/JNA or indirectly via traditional networking protocols or stdin/stdout. Hadoop is based on Java as a high-level programming language and uses Java as a primary language for development.	Medium Instrumentation is Medium because each framework is ultimately running on top of the JVM, meaning AIM test and performance monitoring infrastructure will have access to JMX, allowing applications to provide very fine-grained instrumentation information via a well-defined API. AIM could harvest and manage this information via LIFT, if desired. All individual analytics will have to write their own instrumentation code and find a

Technology	Performance Capability	Learning Curve	Instrumentation
	12,000 threads. • Iterative computations are performed with Spark-on-YARN across 40 nodes. • Sparse data store: 2 PB of data stored in HBase, across 1,900 nodes. This is one of the largest HBase deployments in production. • 365 PB of available raw storage on HDFS, spread across 30,000 nodes (about 150 PB is currently used). • About 400,000 jobs/day run on YARN, corresponding to about 10M hours of compute time per day.	Hadoop can interact with existing libraries directly via JNI/JNA or indirectly via traditional networking protocols or stdin/stdout (Hadoop "Streaming" API). Note the "streaming" API is misnamed and is not a streaming API at all, but a way to interact with command-line-driven programming languages such as Python. Spark is based on Scala, a next-generation functional programming language written for the JVM. Spark also supports Java and C/C++ via JNI/JNA or indirectly via traditional networking protocols or stdin/stdout.	way to provide it via JMX. If the algorithms are written in Java or Scala, integration with JMX is straightforward; if written in a third-party library or non-JVM programming language, integration will not be as straightforward. A number of toolkits provide web-based (HTTP) management consoles, but these are specifically designed for system-level management.
NPMs: MPI, OpenMP and Others	High Since MPI uses native communication networks and combined with C++, it would result in minimal overhead in terms of indirection and performance loss. **NOTE:** This column should be defined in terms of what should be expected in execution time of an algorithm, if san "n" nodes were used. Of course using 2-3 times the number of nodes to get similar result may be possible, but that would defeat the purpose. The rest of the rows in this table should determine the input, what algorithms were executed, and missed performance. In other cases, the interconnectivity is with Ethernet and using sockets (TCP/IP), which is not suitable for high-end systems like PIC, the primary computational target for AIM.	Medium/Low While most of the teams have at least someone with a basic knowledge of MPI; most developers, unless they are familiar with C/C++ distributed programming are not going to be familiar with MPI on an expert level. PIC's Olympus cluster currently has installations of the MPI and OpenMP ecosystem. It is expected that many of the large-scale systems deployed on Olympus would leverage MPI for scalability.	High/Medium There are several tools that provide performance analysis at parallel scale, without requiring source code changes. For example, HPCToolkit, which automatically samples messages. Similarly PAPI provides low-level information on performance lost and wrappers to performance counters provided by hardware. It is widely used in HPC systems and can be used on desktops/clouds as well. An important aspect of using the MPI ecosystem is the abundance of debugging large-scale parallel programs. License-based tools use similarity analysis and other measures to provide graphical interface for debugging large-scale programs. This functionality would be critical for the AIM teams since they would eventually write large-scale analysis algorithms.

Table A4. Visualization, Risk Opportunities, Expertise at PNNL, and Notes

Technology	Visualizations	Risk Opportunities	Expertise at PNNL	Notes
IBM InfoSphere Streams	Low See Visualization and Dashboards Built-in visualizations appear to be limited, displaying basic analytic and data stream information via a web console. New visualizations would have to be developed. It is unclear if there is an API to develop custom visualizations, although the "Surveillance and Physical Security: TerraEchos" project shows advanced standalone visualizations.	High/Medium Risks are related more to licensing costs and vendor lock-in rather than ability to process and analyze streams. For example, even if the licensing is not prohibitively expensive for government customers, will traditional academic or scientific communities support commercial licensing costs? There is also risk that project teams will have to ramp up on a new programming language, programming paradigm(s), and ultimately learn how to interact with and optimize execution of their algorithms within Streams. Opportunities lie in the ability to have a fully tested, reliable, and commercially supported platform to inject analytics/algorithms. Many of our National Security Directorate customers, including in the Intelligence Community, are already using Streams or would like to use Streams (**NEED TO VERIFY**) so a clear path to funding is available.	No	Streams is a CEP system and has a direct comparison table in the Red Book to other CEP frameworks. **See Pages 38-41 in the IBM Streams Red Book.** **Complex Event Processing** Analysis on discrete business events Rules-based processing using if/then/else) with correlation across event types Only structured data types are supported Modest data rates **InfoSphere Streams** Analytics on continuous data streams. Supports simple to extremely complex analytics and can scale for computational intensity. Supports an entire range of relational and non-relational data types. Extreme data rates (often an order of magnitude faster).
Apache Storm	Low There are no visualization capabilities directly related to Storm, but several companies are building analytics packages for integration with Storm.	Medium Risks of implementing Storm coincide with the project being a community supported and open source. Depending on contributors, competing projects, and adoption by	Yes	

Technology	Visualizations	Risk Opportunities	Expertise at PNNL	Notes
	An example of integrating analytics with Storm: • http://www.slideshare.net/ Hadoop_Summit/ realtime-analytics-with-storm	larger commercial interests projects can be enormously successful (Apache web server) or fall into obscurity.		
Apache Kafka	Low Since Kafka is a supporting technology for higher-level processing and analytics, it does not provide any visualizations.	Medium The risks of implementing Storm coincide with the project being community supported and open source. Depending on contributors, competing projects, and adoption by larger commercial interests projects can be enormously successful (Apache web server) or fall into obscurity.	Yes/No	There is expertise at PNNL for using Apache ActiveMQ and Code Connected ZeroMQ, which provide similar message queuing functionality as Kafka. There are also well-supported integration tools for the Java Spring Framework, so integration with Kafka is fairly straightforward. The risk and expertise ratings were given because Kafka is much more than simply a message queuing platform, as described here, so configuring and learning how to properly use Kafka with other technologies like Storm and Hadoop will require training.
GridGain In-Memory Streaming	Low Analytic visualizations are not included in GridGain but can be easily integrated through GridGain's client-facing APIs.	Medium The risk for this product falls into two main categories: • High-value features of product may only be available via commercial licensing. • Supporting documentation suggests these features may not be of high value to AIM, such as HA in-memory databases and the Visor GUI. Researchers will need to write all of	Yes	GridGain is an industry leader in the in-memory compute and data grid space, and one of the first to produce a viable commercial product based on its technology stack.

Technology	Visualizations	Risk Opportunities	Expertise at PNNL	Notes
		their applications in a supported programming language/ programming paradigm.		
LIFT	Low Analytic visualizations are not included in LIFT but can be easily integrated through LIFT's client-facing APIs.	High/Medium LIFT is a mature, yet growing framework. While support for high-speed event/message processing is available and maturing, support for processing raw byte streams is less mature. In addition, each technology that intends to integrate into LIFT will need to be developed with either remote execution in mind or the ability to be wrapped by Java code for local execution.	Yes	The purpose of LIFT is not to provide a complete streaming analytics framework, but to integrate and orchestrate event/message processing through various remote and/or third-party tools. LIFT would also manage the instrumentation, user-facing API, and additional features required of a higher-level framework. A reasonable option is using LIFT on top of native project implementations, with individual projects scaled as required and integrated using a suite of common APIs such as the Advanced Message Queuing Protocol or a content management system. Ideally, projects would be designed in such a way as to support high-speed asynchronous two-way messaging. We could setup LIFT similarly to a traditional CEP framework, but appropriately decouple the architecture and underlying implementations as needed. LIFT can also support CEP through direct integration of technologies such as Drools Fusion. The Drools 5 Behavioural Modeling Platform moves away from any of the narrow modeling perspectives that see only rules, processes, or

Technology	Visualizations	Risk Opportunities	Expertise at PNNL	Notes
				events as their main modeling concept. To effectively achieve the flexibility and power of behavioral modeling, a platform must understand all of these as primary concepts and allow them to leverage on each other strengths. In this scenario, Drools Fusion is an independent module, but still completely integrated with the rest of the platform, that adds a set of features to enable it: • Understand and handle events as first class citizens of the platform • Select a set of interesting events in a cloud or stream of events • Detect the relevant relationships (patterns) among these events • Take appropriate actions based on the patterns detected Additionally, LIFT is currently undergoing proposal development to become available through PIC as an institutionally supported "cloud" resource. This could add significant value to AIM as additional non-AIM analytics could be integrated and leveraged through this capability.
Yahoo Big Data uses Apache Storm Apache Spark Apache Hadoop	Low There are no visualization capabilities inherent with the provided technologies. All information will need to be visualized with third-party tools.	Medium Maturity is extremely dependent on the technology chosen. Hadoop is mature, the current de facto standard for large-scale batch processing on commodity hardware clusters. Open-source actively	Yes	This is the infrastructure most widely used in industry to solve both batch-based and streaming "big data" analytics. These technologies were also designed to scale by simply adding more commodity hardware, possibly augmented with some

Technology	Visualizations	Risk Opportunities	Expertise at PNNL	Notes
Optional: Apache Mesos LIFT		supported by a number of commercial vendors Storm is maturing and used in production environments at Twitter, Yahoo, and others on mission-critical data at high volume and velocity. The open-source project is still young, although third-party developer support is growing. Spark is young/maturing. It is the open-source equivalent of GridGain, although as of 02/2014 that may be a moot point. Spark is still pretty raw and just recently graduated from academia to an Apache Incubator project. Cloudera has recently picked up commercial support of Spark in CDH 5.x and third-party developer support is growing.		specialty hardware such as graphics or accelerated processing units. The main question is whether this type of infrastructure will work for AIM's specific needs or if a more traditional, custom, low-level HPC style system is needed (see below).
NPMs: MPI, OpenMP and Others	Low This section is slightly unclear. Does it intend to provide visualization of the output, which should be dependent on the choice of application? If there is an existing visualization app that reads from files and does the job, it should be a requirement for parallel analysis algorithm to output the data in that format. The answer to a visualization app is dependent on the application.	Risk - Low Low risk since most teams have someone who could write at least a basic MPI program. The scale to 1000 nodes is standard for MPI, which may be the ultimate target for AIM. Maturity - High MPI has been around for 20 years. The high likelihood of very good performance and scalability on smallish node counts such as 1000 is not a problem.	Yes	Not all clients are happy about using MPI. It has a bare bones feel that is not appealing, but it will do the job.

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

U.S. DEPARTMENT OF
ENERGY

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

www.pnnl.gov