IMPROVING DATA ANALYSIS THROUGH DIVERSE DATA SOURCE INTEGRATION

Jennifer Casper, Ronald Albuquerque, Peter Leveille, Jeremy Hyland, Jing Hu, Ed Cheung, Barry Lai, and Don Landing
The MITRE Corporation
202 Burlington Rd
Bedford MA

ABSTRACT

Daily sensor data volumes are increasing from gigabytes to multiple terabytes. The manpower and resources needed to analyze the increasing amount of data are not growing at the same rate. Current volumes of diverse data, both live streaming and historical, are not fully analyzed. Analysts are left mostly to analyzing the individual data sources manually. This is both time consuming and mentally exhausting. Expanding data collections only exacerbate this problem.

Improved data management techniques and analysis methods are required to process the increasing volumes of historical and live streaming data sources simultaneously. Analysts need improved techniques are needed to reduce an analyst's decision response time and to enable more intelligent and immediate situation awareness. Faster analysis of disparate information sources may be achieved by providing a system that allows analysts to pose integrated queries on diverse data sources without losing data provenance.

KEYWORDS

Sensor, streaming, historical, database, framework, service oriented architecture, GMTI, analysis

INTRODUCTION

Overwhelmed analysts are spending precious time managing the logistics of accessing diverse data sources and manually integrating data for basic analysis. The number and size of the data sources are continually growing, only making this problem worse. The number of analysts and resources available for processing is not growing at that same rate – this is neither a feasible nor an economical solution. The time analysts have to spend accessing and manually integrating data reduces the amount of time and effort left for needed data analysis. Improved data management techniques are needed to reduce the analysis timeline and enable more rapid and flexible analysis of diverse live streaming and historical data sources.

The goal of this effort is to make it easier for an analyst to process large amounts of diverse data, both live streaming and historical, by investigating how to integrate multiple data sources. The first step in moving towards the goal is to facilitate the ability of analysts to easily query numerous data sources simultaneously.

The Sensor Data and Analysis Framework (SDAF) system provides analysts with the ability to pose integrated queries to all available live streaming and historical data sources concurrently. The core of SDAF is the framework – an event-driven service oriented architecture (SOA). A SOA approach has enormous benefits, including scalability and flexibility. These benefits satisfy the needs of analysts. The system needs to scale more easily for new live streaming and historical data sources, algorithms, and desired clients, while still providing the integrated query capability.

This effort is beneficial for analysts because: (a) analysts gain the ability to pose integrated queries to desired live streaming and historical data sources without having to access each individually – this saves them time and effort, (b) analysts may keep using their current exploitation tools or use tools with features that take advantage of framework capabilities, and (c) Analysts can quickly employ algorithms for event detection alerting.

This work was successful in revealing that: (a) Analysts found the ability to pose integrated queries on live streaming and historical data sources useful, and (b) SOAs are useful in implementing the integrated query capability.

The purpose of this paper is to communicate: (a) how the integrated query capability was successfully implemented via a SOA, (b) how well the system performed, (c) how useful the system was found to be by users, and (d) lessons derived from analysts' feedback, in an effort to provide guidance for future data analysis tools.

This paper is organized as follows. First, related work is discussed. The evolution of SDAF describes the system design, tradeoffs, and implementation. The framework performance reveals how well it works under intense data loads. A narrative describing how SDAF was applied in three different fields and initial user feedback is given. Finally, lessons learned from the system development and

field applications are revealed, followed by conclusions and future work.

BACKGROUND

Data-intensive applications encounter "a continuous, unpredictable and unbounded flow of data as input, referred as streams" [1]. Data-intensive applications, such as sensor networks, operational testing, and network monitoring, would benefit from a general-purpose data management system. Data stream management systems handle the logistics and optimization of stream data input, routing, and result output. The application-driven customization can then be focused on the type of services, filters, or algorithms that need to be applied to the streams.

Streaming systems, such as those discussed in [2][3][4][5][6], offer users several advantages over static databases. For example, streaming systems can receive new data at any time during execution and process it in real-time through main memory, respond to updates, and feed the results to many corresponding applications. They can quickly report new results derived from the incoming data, which enables them to handle time-sensitive data with minimal delay. Some results must be bound to only a portion of the data, referred to as windowing, rather than running over all data that has ever passed into the system. In other words, streaming systems do not always give complete results. Relational databases, on the other hand, do not have to make this sacrifice of accuracy for the sake of quickly generating results. Static databases optimize for the overall throughput of the system, while streaming systems aim to maximize continuous output rate.

A general, scalable, hybrid approach is needed to properly support integrated queries and algorithm customization for data-intensive applications. Neither stream management systems nor static databases provide the optimal solution. Stream management systems are useful for processing live streams, while Databases are ideal for storing and processing historical data. However, some events, like abnormalcy, require querying live streaming and historical data simultaneously. The authors in [7] employed a data stream management system with access to a database in order to compare live events with similar historical events. Other methods employ combinations of data stream management and database systems with distributed sensor networks for query processing [8][9][10][11][12].

SOAs are advantageous for this effort due to their inherent scalability and ease of service additions. SOAs have been successfully employed processing streaming weather data [13]. Despite installation consistency challenges, the service interface was found to be a huge advantage. The authors in [14] are investigating a grid-based service architecture for video streaming.

EVOLUTION OF THE SENSOR DATA & ANALYSIS FRAMEWORK

SDAF's initial framework was a standalone Java application. As a result of discussions with prospective users, SDAF was completely refactored into its current form of an intelligent information broker. The new open-standards implementation is built on JBoss Application Server. It utilizes the latest Java Enterprise Edition (Java EE) and Web Services standards – implementing a set of service providers that comprise the event-driven SOA. Figure 1 shows an overview of the SDAF architecture.

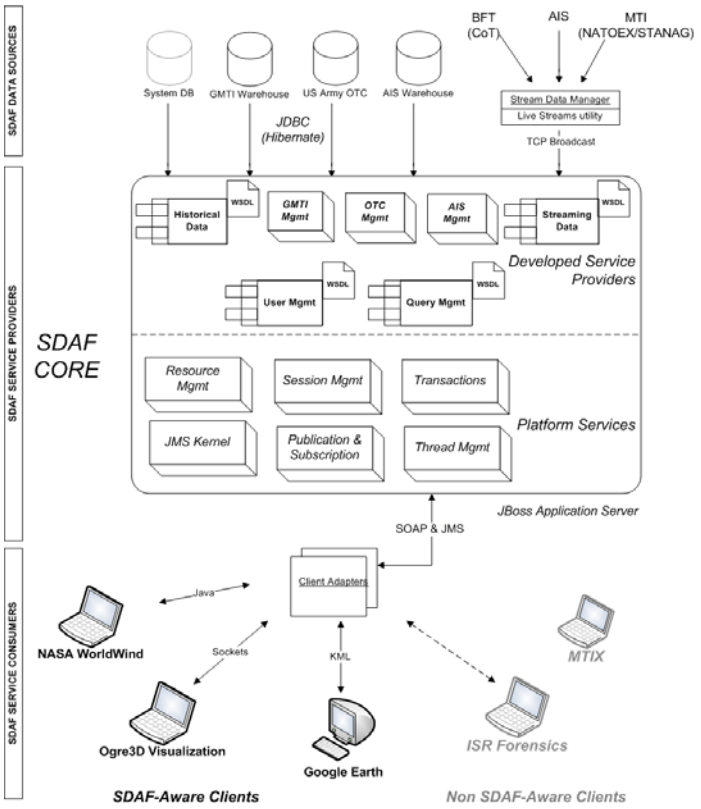

Figure 1 - SDAF SOA Overview

SDAF comprises distinct layers – from the data sources through to the Service Providers and Service Consumers (clients). At the heart of SDAF is the service provider infrastructure, which enables clients to pose queries that span data sources across windows of time. The framework provides a standards-compliant platform to add analysis algorithms as services. It also supports basic security for sharing queries amongst connected analysts – accomplishing this through user logins, query ownership and access control. The architecture is geared towards operational capability and situational awareness. However,

it also incorporates a level of forensic analysis for historical information.

Data Sources

SDAF currently supports the following data types: (a) Ground Moving Target Indicator (GMTI) data in NATOEx and Standardization Agreement (STANAG) 4607 formats, (b) Blue Force Tracking (BFT) in Cursor-on-Target (CoT) format, and (c) Automatic Identification System (AIS) messages. This data is obtained via both historical and live streaming sources.

SDAF utilizes Java EE's Java Persistence API (JPA) and Hibernate to interface with multiple historic information sources. Sources include: (a) an Oracle database that stores GMTI information representing past events, (b) an Oracle database that is the repository for historical AIS data, and (c) a Microsoft SQL Server database that is the US Army repository for historical information on test exercises. SDAF uses MySQL for its system database, which is the persistent store for user and query information.

Live streaming data is managed through the Stream Data Manager. SDAF manages multiple streams of GMTI, BFT and AIS data, broadcasting them over sockets to the SDAF service provider infrastructure.

Service Providers

This comprises the core of SDAF, and includes service providers that manage historical and live streaming data, internal messaging, users and queries. Stream data management incorporates the use of Esper. Esper, an open source Stream Processing Engine (SPE), provides the capability to parse and filter large amounts of streaming sensor data in near real-time. Current stream operators (i.e. filters) highlight data with analyst-defined geo-temporal parameters, alert analysts based on ID, and highlight data filtered on key characteristics.

Service Consumers

Service consumers (clients) provide analysts with visual methods to interface with the framework. SDAF is client-agnostic, supporting both thin and thick client architectures. SDAF-Aware clients can take full advantage of the value-added information provided by the framework, such as query and user information, metadata, provenance information, etc. Non SDAF-Aware clients are typically applications that analysts are accustomed to using and cannot easily be modified to visualize these added parameters. The framework supports non SDAF-Aware clients through a client adapter that feeds clients filtered data in their required formats. Currently supported SDAF-Aware clients include NASA World Wind, Google Earth, and a custom built viewer utilizing the Ogre-3D visualization engine.

The system architecture has evolved over the last year. The first version of SDAF's SOA incorporated the Mule open-source Enterprise Service Bus (ESB) for mediation amongst service providers. Initial testing discovered bottlenecks in query processing. The resultant refactoring incorporated removal of the ESB, as well as several other enhancements. A full discussion of the performance enhancements is provided in the FRAMEWORK PERFORMANCE section. SDAF's software architecture is illustrated in Figure 2.

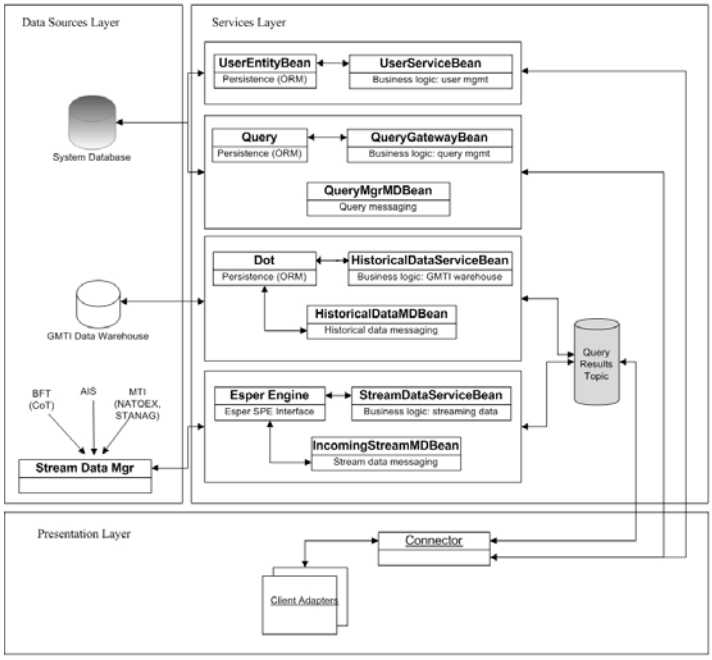

Figure 2 - SDAF Software Architecture

The architecture is flexible, modular and agile – enabling rapid prototyping of new features. It leverages the latest Java EE and Web Services standards and can work on any application server that is compliant with the Java EE specifications (e.g. Sun Glassfish, BEA Weblogic, etc.)

SDAF DATA WAREHOUSE

The SDAF data warehouse supports and facilitates the storage and retrieval of disparate historical sensor data efficiently in a persistent data store. It is based on the MITRE ISR Forensics data warehouse[1]. Initially, only GMTI data in the NATOEx format was stored. The data warehouse was extended to store BFT in the CoT format, as well as AIS data.

[1] http://www.mitre.org/news/events/tech07/2163.pdf

To support efficient spatial queries of data related to location and time, the data warehouse is partitioned by date and time. There are two advantages to the approach. The first advantage is the automatic partition elimination during the formulation of query execution plans. Partition elimination keeps the number of partitions to search for a query to a minimum. The second benefit is that, as the size of the database grows, the number of partitions to search for the same query remains the same, thereby keeping the response time relatively constant. The response time for a test suite of five spatial queries remains relatively constant as the database grows from 51 to 116 million rows [15].

SDAF-AWARE CLIENTS

Ogre 3D

The first SDAF-aware client developed was built from using the Object-Oriented Graphics Rendering Engine (Ogre3D)[2], written in C++. This enabled a rapid prototyping capability to easily visualize queries and resultant data. Queries were visualized in 3D as physical cylinders and cubes residing on the earth. Pressing a toggle would increase opacity and allow visually resizing and repositioning queries. Incoming sensor data was displayed as spheres of varying color and size, or as 3D models representing known vehicle types – for BFT data.

Google Earth

The Google Earth Client was created to demonstrate the openness of the framework and to highlight its ability to support a wide range of applications. Google Earth[3] was identified as a mapping tool to support because of its wide-spread popularity, high quality imagery and free availability. The application itself could not be modified to integrate SDAF query features, due to it being closed source. Instead, the SDAF client, written in C#[4] under .NET[5], was built as a separate adapter application to translate between the framework and Google Earth.

NASA World Wind

Development of the SDAF World Wind[6] client is organized into two major sections – a framework interface and a custom rendering component. The framework interface is effectively a standalone Java Swing based GUI application that provides users access to the functionality of the framework backend. To shorten the turnaround between feature implementation in the framework and feature availability in the client, certain components of the user interface are dynamically generated, most notably the query creation dialog.

The rendering component handles drawing of dots on the screen. Large GMTI queries could return hundreds of thousands of dots. The initial rendering path using the provided surface drawing API would see severe performance degradation with two hundred dots on a system with a 2.16GHz Core 2 Duo processor and GeForce Go 7400 graphics card. Instead of drawing circles, the new custom renderer drew solid-colored quads, and later discrete points, which have a quarter of the memory footprint of quads. OpenGL calls in the render loop were also reduced from a linear function, with respect to the number of dots, to a constant number. This was accomplished by pre-computing vertex arrays and compiling display lists as the client receives the dots, rather than on each individual render pass. The new renderer **handles in excess of three hundred thousand dots** on the same hardware with acceptable performance.

FRAMEWORK PERFORMANCE

Overall performance was a key consideration during the design and development of the SDAF system. In order to be of use, SDAF had to process very large amounts of data quickly. This requirement presented the SDAF team with a number of interesting problems.

The first issue tackled was to ensure SDAF users a timely query submission process. By nature, registering a query with the framework is a synchronous process: the user creates the query, with the options they desire, and then submits the query to the framework through a web services call. The overall user experience is significantly diminished if the web service call to the framework takes an inordinate amount of time, since the user (depending on the client user interface) will probably have to wait for the call to complete before continuing.

In the initial SDAF 1.0 system, the complete time for a single user to submit a query was about six seconds. This long wait time would be unacceptable in a production system, and present a serious scalability issue as times would increase if more users were added to the system.

The solution was to go back over the system architecture looking for areas that were redundant or that added unnecessary amounts of time to the query submission process. As discussed in EVOLUTION OF THE SENSOR DATA & ANALYSIS FRAMEWORK, the resulting second generation SDAF system removed the ESB and provided a more direct line from services to the client. The

[2] http://www.ogre3d.org/
[3] http://earth.google.com/
[4] http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx
[5] http://www.microsoft.com/net/
[6] http://www.worldwind.arc.nasa.gov/

initial system always routed queries through the ESB between services. This created a large amount of unnecessary latency, given that all the services existed within the same Java application server instance. By removing the ESB and simply routing the queries using methods within the application server, the overall number of Java object to Simple Object Access Protocol (SOAP) object **transformations were reduced from fourteen down to two –** once when the SOAP request enters the system, and then again when the response is sent back. This improvement drastically reduced the amount latency incurred during the query submission process.

Along with streamlining the system architecture, many other enhancements were made to improve the performance of the JBoss. This included: (a) adding native complied Apache[7] HTTP handlers[8] for better web services request/response times, (b) tuning JBoss logging according to recommended best practices[9,10], (c) switching to the 64 bit version of the Java Virtual Machine, and (d) increasing the RAM available to JBoss from 512 MB to 2 GB. All of these improvements, in addition to streamlining the system architecture, resulted in a radically improved average query submission time – **down from nearly six seconds to around eight hundredths of a second.**

The query submission times in SDAF 1.0 fluctuated throughout the entire test – where as the query submission times in SDAF 2.0 remained relatively stable. Much of the variability in SDAF 1.0 was due to the large number of transitions between the ESB and the application server hosting the services. Each object passed between the two systems required a web services network call – even though the ESB and application server were running on the same physical system. This introduced a large amount of variability in response times that is no longer seen in the second generation SDAF. The lesson here is that while ESBs are advantageous when integrating disparate loosely coupled diverse services, they are not necessarily needed when connecting a homogeneous collection of controlled services. In fact, performance gains can be realized by exploiting more specialized and efficient methods of communication possible in the homogeneous environment. In the end, it always comes back to using the best design for the specific situation.

Another very important performance area is the ability to process very large amounts of live streaming data. The SDAF query functionality on live streaming data falls under the category of near real-time – which means live events that took place at time *t* will be presented to the end users at time t_1 where $t_1 = t$ + processing time[11]. This means that in order to preserve the freshness of events, and therefore the usefulness of the system, processing time must be kept to a minimum [16].

However, there was one other problem. If events come in faster than they can be processed then they must somehow be buffered. This starts a downhill spiral if the system is never able to clear the buffer before more events are queued. Allowed to continue, this would eventually cause the system to run out of memory. Testing was done to determine the highest level of speed at which events could come in and still be successfully consumed.

Initially, these speed tests revealed an interesting issue with the standard JBoss configuration that the SDAF team previously overlooked. By default, JBoss internally uses an older version of the in-memory database Hypersonic[12], which has documented problems running under heavy loads[13]. This was causing speed tests to crash the framework unexpectedly after about an hour and a half of testing. This issue was easily fixed by replacing[14] Hypersonic with MySQL[15] (or any other more stable database implementation).

The results of the speed tests showed that the framework running on a quad core Xeon 2.33 GHz system with 4 GB of RAM could process around 3.8 million events an hour, with an average event throughput of about one thousand events per second and **peak bursts at a maximum of two thousand events per second.**

While these results are promising, more testing is needed to determine the effect more complex queries have on overall performance. The SDAF team currently assumes that event throughput will decrease proportionally to the query complexity, but further testing is required to test this hypothesis. It is clear that a more complex query will add processing time, thus increasing the time between *t* and t_1. The question then becomes how much processing time can be added to *t* before the event at t_1 is no longer valuable to the user. Much of the answer to this question will be domain specific – some users in some domains will be able to tolerate less fresh events then others. Given a specific upper bound on freshness, the SDAF team should be able

[7] http://apr.apache.org/
[8] http://wiki.jboss.org/wiki/HowToAddAprToJBoss
[9] http://wiki.jboss.org/wiki/JBossASTuningSliming
[10] http://www.jboss.com/pdf/jbwb/linuxjbperf-2.pdf
[11] http://www.its.bldrdoc.gov/fs-1037/dir-024/_3492.htm
[12] http://www.hsqldb.org/
[13] http://wiki.jboss.org/wiki/HypersonicProduction
[14] http://wiki.jboss.org/wiki/ConfigJBossMQDB
[15] http://www.mysql.com/

to devise a way to reach the goal using better hardware and existing load balancing and clustering technologies.

FIELD APPLICATIONS

Several interesting uses of SDAF were quickly identified while working with MTI analysts. The ability to display and query all data provided over a certain time period and simultaneously correlate that with all live streaming MTI feeds, without the need to know specific mission names and locations proved to be very useful. This also enabled users to quickly correlate historical activity to live field activity, which helped identify abnormal activity.

As users became more accustomed to SDAF's upstream filtering and querying capabilities, they requested specific features. One example of this was a size and speed filter. Utilizing SDAF, it was simple to add upstream filtering of live streaming and historic data for specific size and speed ranges. Users focused on identifying objects with certain speed and size characteristics were then able to greatly de-clutter their displays and more easily focus on specific MTI of interest.

This is potentially an even greater benefit to disadvantaged users at the tactical edge. MTI intelligence data is not provided to disadvantaged users due to bandwidth limitations. The upstream filtering applied by SDAF means only a subset of the full data stream needs to be sent to an end user. With restrictive enough filters applied to streaming MTI, the bandwidth requirement can be greatly reduced making MTI a possibility for disadvantaged users.

Personnel working in the area of Maritime Domain Awareness (MDA) have also expressed interest in the use of SDAF to help in the correlation of MTI and AIS. This would be used to detect when and where a ship was located if its AIS feed was interrupted or spoofed. Combined with SDAF's built in alerting capabilities, users could receive automatic notification if this occurs and immediately reposition their view to the event.

Members of the operational test community have also found a novel use for SDAF. Utilizing SDAF's capability to easily plug in additional services, they have sponsored the creation of an information fusion service to help in the evaluation and scoring of equipment and system field tests. SDAF's design enables rapid integration of new data sources. An added service can subscribe to any of the data sources SDAF has connected and utilized the information from these various data sources to create its own data product. End users of SDAF can then subscribe to the data being delivered by this new service. Since the service will be utilizing established SOA protocols to send and receive data, developers are able to concentrate primarily on the algorithmic implementations and not be concerned with a multitude of stove piped data connections.

LESSONS LEARNED

In terms of initial field applications (see FIELD APPLICATIONS), SDAF was successful in providing users with a useful means of posing integrated queries on live streaming and historical data sources simultaneously. This is evident by the three programs currently in collaboration. In addition, users naturally began adapting to the idea of customized algorithms and filters. They began making requests and were delighted at the quick turnaround.

From a system design perspective, SDAF was also successful. The system was able to process and manage an intense amount of live streaming data using a SOA as the core, with the aforementioned changes (see FRAMEWORK PERFORMANCE).

With the success of this work, several valuable lessons have been identified:

SOAs may be configured to handle intense data loads. Many alterations were made to the SDAF framework for this purpose (see FRAMEWORK PERFORMANCE). Ultimately, the SOA design is an art where adjustments are needed to meet user requirements.

ESBs are advantageous in some scenarios, but disadvantageous in others. The ESB used in SDAF 1.0 actually increased the query submission time. The queries within the SDAF system configuration could be routed using methods within the application server. This greatly decreased query submission time so the framework was useable.

Cosmetic attributes in clients may be unnecessary and costly when dealing with live data streams. World Wind's renderer created attractive shapes and was flexible, but did not perform well under intense data loads. A simpler renderer was written in order to handle in excess of three hundred thousand dots.

Flexible and general-purpose client tools struggle with intense data loads. Google Earth is a useful mapping tool. However, its general purpose design was found not to be beneficial when dealing with large data loads. Work-arounds were needed to achieve reasonable performance.

Historical data sources need to be constructed to support user queries. Changing the partitioning scheme

in the SDAF data warehouse to support geo-temporal queries kept the query response time relatively constant despite the growing amount of data stored. Optimizing the data sources ultimately benefits the users.

CONCLUSIONS AND FUTURE WORK

Analysts have found the ability to pose integrated queries on live streaming and historical data sources useful. Other government programs are recognizing the ways in which they can leverage this capability. The SDAF team is adapting the framework for their data sources, algorithms, and client needs. This expands the suite of data types SDAF can process and the clients that it can connect to it. The team will continue to improve SDAF, through refining and testing, as they apply it to different fields.

SOAs are useful in implementing the integrated query capability. The SDAF team was able to take advantage of SOAs innate scalable and flexible characteristics, while still managing live streaming data. This was made possible via the modifications outlined in the FRAMEWORK PERFORMANCE.

In addition to the field application direction noted above, the SDAF team is initiating future research work to focus on algorithm development for pattern recognition and alerting – both in the MTI analysis and network monitoring fields. Current collaborations inspired this follow-on research direction. This work will leverage SDAF as the foundation to easily employ developed algorithms for testing, manage data sources and queries, and visualize results.

ACKNOWLEDGEMENTS

The authors would like to thank William Harris for his motivating passion, long-standing support and continuous system feedback. Also, thanks to Phil Hallenbeck for his inspirational visions and strong support. This work is financially supported by the MITRE Innovation Program.

REFERENCES

[1] Gilani, S., Sonune, B., Kendai, B. and S. Chakravarthy. The Anatomy of a Stream Processing System. *Lecture Notes in Computer Science: Flexible and Efficient Information Handling*, vol. 4042, pp. 232-239, 2006.

[2] Abadi, D. et al. Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal*, vol. 12, no. 2, pp. 120-139, 2003.

[3] Plale, B. and Schwan, K. Dynamic Querying of Streaming Data with the dQUOB System. *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, April 2003.

[4] Widom, J. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. *Proceedings of the 2005 CIDR Conference*, January 4-7, 2005, Asilomar, CA.

[5] Vijayakumar, N. et al. Calder Query Grid Service: Insights and Experimental Evaluation. *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006.

[6] Rundensteiner, E. et al. CAPE: A Constraint-Aware Adaptive Stream Processing Engine. *Advances in Database Systems: Stream Data Management*, Springer Verlag, pp. 83-111, 2005.

[7] Kwon, Y. et al. Identifying Similar Past Events in a Continuous Monitoring System. *Proceedings of the 2007 VLDB*, September 23-28, 2007, Vienna, Austria.

[8] Abadi, D. et al. An Integration Framework for Sensor Networks and Data Stream Management Systems. *Proceedings of the 30th VLDB Conference*, 2004, Toronto, Canada.

[9] Agre, J. and Clare, L. An Integrated Architecture for Cooperative Sensing Networks. *IEEE Computer*, vol. 33, no. 5, pp. 106-108, May, 2008.

[10] Mokbel, M. et al. Continuous Query Processing of Spatio-temporal Data Streams in PLACE. *Proceedings of the Second Workshop on Spatio-Temporal Database Management*, August 30, 2008, Toronto, Canada.

[11] Madden, S. et al. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122-173, March 2005.

[12] Yao, Y. and Gehrke, J. Query Processing for Sensor Networks. *Proceedings of the 2003 CIDR Conference*, January 5-8, 2003, Asilomar, CA.

[13] Droegemeier, K. et al. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *IEEE Computing in Science and Engineering*, vol. 7, no. 6, pp. 12-29, December 2005.

[14] Hummel, K. and Juhasz, Z. Towards a Service-Oriented Architecture (SOA) for Performance-Aware Mobile Grid Service. Technical Report, Universität Wien, April 2008.

[15] Cheung, E. et al. Sensor Data & Analysis Framework (SDAF) Data Warehouse. Technical report, The MITRE Corporation, February 2007.

[16] Bouzeghoub, M. and Peralta, V. A Framework for Analysis of Data Freshness. *Proceedings of the International Workshop on Information Quality in Information Systems*, 2004, Paris, France.