Enabling Persistent Queries for Cross-aggregate Performance Monitoring

TR-13-01

Anirban Mandal, Ilia Baldine, Yufeng Xin,
Paul Ruth, Chris Heerman

April 2013

RENCI Technical Report Series
http://www.renci.org/techreports

Enabling Persistent Queries for Cross-aggregate Performance Monitoring

Anirban Mandal, Ilia Baldine, Yufeng Xin, Paul Ruth, Chris Heerman
Renaissance Computing Institute, UNC - Chapel Hill

Abstract

It is essential for distributed data-intensive applications to monitor the performance of the underlying network, storage and computational resources. Increasingly, distributed applications need performance information from multiple aggregates, and tools need to take real-time steering decisions based on the performance feedback. With increasing scale and complexity, the volume and velocity of monitoring data is increasing, posing scalability challenges. In this work, we have developed a Persistent Query Agent (PQA) that provides real-time application and network performance feedback to clients/applications, thereby enabling dynamic adaptations. PQA enables federated performance monitoring by interacting with multiple aggregates and performance monitoring sources. Using a publish-subscribe framework, it sends triggers asynchronously to applications/clients when relevant performance events occur. The applications/clients register their events of interest using declarative queries and get notified by the PQA. PQA leverages a complex event processing (CEP) framework for managing and executing the queries expressed in a standard SQL-like query language. Instead of saving all monitoring data for future analysis, PQA observes performance event streams in real-time, and runs continuous queries over streams of monitoring events. In this work, we present the design and architecture of the persistent query agent, and describe some relevant use cases.

1 Introduction

Advanced multi-layered networks allow to connect widely distributed computational and storage resources to scientific instruments to pursue the goals of data-driven computational science. The increasingly dynamic behavior of networks and the new connectivity options at different layers enabled by new technologies has revolutionized the way computational activities are structured. They permit a move from static arrangements of resources that persist over long periods of time to highly dynamic arrangements that respond to the needs of the scientific applications by dynamically provisioning necessary network and edge resources with some notion of optimality. Critically, no resource provisioning and allocation mechanism can operate on behalf of the application unless it is capable of providing feedback to the application. The feedback describes the performance and state of the allocated resources, and the performance of the application on the allocated resources. Large ensembles of network, compute and storage resources inevitably experience performance degradations and failures, and applications must be informed about them. Providing feedback about resource performance to the application, to enable closed-loop feedback control and dynamic adjustments to resource allocations is of utmost importance. Many monitoring solutions exist today that can provide such feedback including perfSONAR, Ganglia, MonALISA etc. However, presenting this information to an application in a sufficiently abstract and useful fashion still remains a challenge.

The challenge is even greater when one has to monitor distributed infrastructure and distributed application executions spanning multiple domains, and there is no central point of control. In order to effectively analyze end-to-end bottlenecks with respect to several aspects of application execution (network congestion, high latency, compute load, storage system bottlenecks), we need a mechanism to federate performance information from these diverse aggregates and derive useful insights in an application specific manner. The focus should be on gaining high-level insights important to application performance. This entails taking a cross-aggregate view of computational, network and storage performance, gathering performance metrics (from several measurement sources like perfSONAR services, network infrastructure monitors, XMPP based monitoring entities, on-node performance information - OS, system, application counter

data etc.) and reasoning about them in the context of a particular application execution.

The volume and velocity of monitoring data are increasing rapidly with increased scale and complexity of the substrate and increased availability of monitoring data from various sources, each capable of generating lots of monitoring data at a rapid rate. Often, monitoring data is stored for future analysis to analyze past performance. With high volume performance monitoring data, we can no longer afford to store all performance data for post-processing and analysis. Since steady state performance is seldom interesting, not all performance data tends to be useful. Also, current applications and tools managing application executions need dynamic real-time feedback of application performance so as to enable real-time steering based on observed performance. So, we are facing scalability challenges in dealing with high volume performance data and increasingly need to provide real-time feedback to tools.

In this work, we address some of the above challenges. We have developed a persistent query agent (PQA) that enables persistent queries on application and system performance. Applications or clients managing application execution are able to express important performance metrics, threshold conditions, or event condition combinations using declarative queries. PQA enables federated performance monitoring by interacting with multiple aggregates and performance monitoring sources. By leveraging a publish-subscribe framework, it asynchronously sends triggers to applications/clients when relevant performance events occur. The applications/clients register their events of interest using queries and get notified by the PQA when those events occur. Our work presents a novel use of an open source complex event processing (CEP) framework to manage and execute these queries expressed in a standard SQL-like query language. Instead of saving all monitoring data for future analysis, PQA observes performance event streams in real-time, and runs continuous queries over streams of events generated from the various performance monitoring sources.

The remainder of the paper is structured as follows. Section 2 describes related work. Sections 3 and 4 present the motivation, design and architecture of PQA. Section 5 describes some relevant use cases and section 6 concludes the paper.

2 Related Work

perfSONAR [13, 15, 4] offers a web-services based infrastructure for collecting and publishing network performance monitoring. It consists of a protocol, architecture and set of tools developed specifically to work in a multi-domain environment with the goal of solving end-to-end performance problems on network paths crossing multiple domains. perfSONAR provides hooks for delivering performance measurements in federated environments. However, it is the responsibility of higher level tools to make use of perfSONAR data in a way relevant to a particular distributed application.

There are several other multi-domain monitoring tools. MonALISA [11] is a framework for distributed monitoring. It consists of distributed agents that handle metric monitoring for each configured host at its site and all the wide area links to other MonALISA sites. MonALISA provides distributed registration and discovery, and is designed to easily integrate existing monitoring tools and procedures to provide metric information in a dynamic, customized way to other services or clients. The underlying conceptual framework is similar to that of perfSONAR. INTERMON [5] is another multi-domain network monitoring framework, which focuses on inter-domain QoS monitoring and large scale network traffic analysis. They model abstractions based on traffic and QoS parameter patterns and run simulations for planning network configurations. Their approach is centralized, where flow, topology and test information are collected and stored in a central location for running the analysis. Other notable multi-domain network monitoring frameworks are ENTHRONE and EuQoS. In [3], Belghith et. al present a case for a configurable multi-domain networking architecture, and discuss collaboration schemes used to select measurement points that participate in multi-domian monitoring, and to configure the parameters of the measurement points selected.

OMF [8] provides a set of software services to run repeatable experiments on network testbeds, and to gather measurements from those experiments that are potentially running across several domains. OMF enabled experiments can use the OMF measurement library (OML) [14] to collect and store any type of measurements from applications. OML provides an API to add user defined measurement points and to inject the measurement streams into the library. These streams are processed by the library as defined by the user, including filtering etc. and results are pushed to local files, or to OMF control servers that store the results in a database.

There has been some work on automated ways of using and analyzing perfSONAR data. OnTimeDetect [6] does network anomaly detection and notification for perfSONAR deployments. It enables consumers of perfSONAR measurements to detect network anomalies using sophisticated, dynamic plateau detection algorithms. Pythia [9] is a data analysis tool that makes use of perfSONAR data to detect, localize and diagnose wide-area network performance problems. Kissel et. al. [10] have developed a measurement and analysis framework to automate troubleshooting of end-to-end network bottlenecks. They integrate measurements from network, hosts

and application sources using a perfSONAR compatible common representation, and an extensible session protocol for measurement data transport, which enables tuning of monitoring frequency and metric selection. They leverage measurement data from perfSONAR, NetLogger traces and BLiPP for collecting host metrics.

3 Persistent Query Agent (PQA)

Although there exist tools that analyze monitoring data from multi-domain measurement sources, they are mostly targeted toward solving one particular problem. It is difficult to configure or customize these tools to diagnose cross-aggregate performance problems. Clients can't programmatically ask questions about metrics, nor can they be automatically notified. Also, most of the tools do an after-the-fact analysis to determine what went wrong post-mortem, which might not be always possible with proliferation of available monitoring data. The requirements of applications and clients to obtain dynamic, real-time, cross-aggregate performance feedback pose challenges not addressed by existing tools. So, we have developed a persistent query agent (PQA) for providing real-time performance feedback to applications or clients so as to enable steering. PQA interacts with multiple aggregates and performance monitoring sources and asynchronously sends triggers to applications/clients when relevant performance events occur. The applications/ clients register their events of interest using queries and get notified by the PQA when those events occur. PQA doesn't store monitoring data. It processes performance event streams in real-time using persistent client queries.

PQA uses an off-the-shelf complex event processing (CEP) [12] engine for managing and executing the queries expressed in a standard SQL-like query language. The queries enable expressing complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, and sorting. The CEP engine behaves like a database turned upside-down. Queries "persist" in the CEP system. Data or events are not stored, rather "watched" and analyzed as they pass by. In the following sections, we present the design, architecture and current implementation status of the persistent query agent.

4 PQA Architecture

There are various components of PQA as in Figure 1, which are described in more detail in the following sections.

- Esper engine: This is the complex event processing engine that processes performance measurement events injected, and triggers actions when queries get satisfied. The various PQA monitoring clients inject events into the Esper engine.

- Trigger listeners: They are responsible for publishing events of interest when a query is satisfied. Applications/clients that are interested in those events can subscribe to events of interest. Typically, events of interest would correspond to queries submitted by the applications. Applications would automatically be notified when such events occur.

- Query manager: It is responsible for managing application queries through an XML-RPC interface. Applications can register and delete queries with it. It injects new queries and associated triggers into the Esper engine.

- PQA monitoring clients: A perfSONAR web services (pS-WS) client obtains measurement data by querying available perfSONAR measurement archives (MA) services. This client injects events streams into the Esper engine. XMPP pubsub subscriber clients obtain measurement data by subscribing to pubsub nodes where measurements are published periodically. Whenever new items are published on the pubsub node, this client injects a corresponding event stream into the Esper engine.

4.1 Esper Engine

Esper is a framework for performing complex event processing, available open source from EsperTech [7]. Esper enables rapid development of applications that process large volumes of incoming messages or events, regardless of whether incoming messages are historical or real-time in nature. Esper filters and analyzes events in various ways, and responds to conditions of interest with minimal latency. CEP delivers high-speed processing of many events, identifying the most meaningful events within the event cloud, analyzing their impact, and taking subsequent action in real time. Some typical examples of applications of CEP are in finance (algorithmic trading, fraud detection, risk management), business process management and automation (process monitoring, reporting exceptions, operational intelligence), network and application monitoring (intrusion detection, SLA monitoring), and sensor network applications (RFID reading, scheduling and control of fabrication lines) [7].

Relational databases or message-based systems such as JMS make it very difficult to deal with temporal data and real-time queries. By contrast, Esper provides a higher abstraction and intelligence and can be thought of as a database turned upside-down: instead of storing the

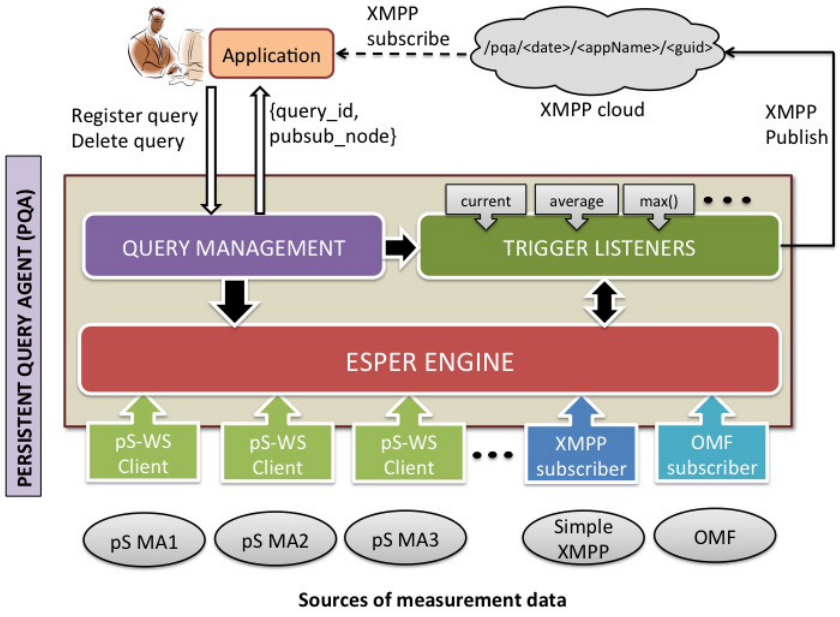

Figure 1: PQA architecture

data and running queries against stored data, Esper allows to store queries and run the data through. Response from the Esper engine is real-time when conditions occur that match user defined queries. The execution model is thus continuous rather than only when a query is submitted. It is for this precise reason we have chosen Esper as our persistent query engine.

Th Esper Event Processing Language (EPL) allows registering queries into the engine. A listener class, which is a plain Java object, is called by the engine when the EPL condition is matched as events flow in. The EPL enables to express complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, and sorting. Esper EPL statements can also be combined together with "followed by" conditions thus deriving complex events from more simple events. Events can be represented as Java classes, JavaBean classes, XML document or java.util.Map, which promotes reuse of existing systems acting as message publishers. Esper offers a mature API with features like

- Event stream processing and pattern matching - Esper provides (a) *Sliding windows*: time, length, sorted, ranked, accumulating, etc. , (b) *Named windows* with explicit sharing of data windows between statements, (c) *Stream operations* like grouping, aggregation, sorting, filtering, merging, splitting or duplicating of event streams, (d) *Familiar SQL-standard-based continuous query language* using insert into, select, from, where, group-by, having, order-by, and distinct clauses, (e) *Joins* of event streams and windows, and so on. Esper provides logical and temporal event correlation, and pattern-matched events are provided to listeners.

- Event representations - Esper supports event-type inheritance and polymorphism as provided by the Java language, for Java object events as well as for Map-type and object-array type events. Esper events can be plain Java objects, XML, object-array (Object[]) and java.util.Map including nested objects and hierarchical maps.

We have leveraged the Esper engine in our design of PQA. The PQA monitoring clients construct simple Java object based Esper events and inject them into the Esper engine. The Esper EPL queries concerning these monitoring events are injected into the Esper engine by the query management module. The trigger listeners are registered with the Esper engine as callbacks for performance event triggers.

4.2 XMPP Publish Trigger Listeners

The XMPP pubsub specification [1] defines an XMPP protocol extension for generic publish-subscribe functionality. The protocol enables XMPP entities to create nodes (topics corresponding to relevant events) at a pubsub service and publish information at those nodes;

an event notification (with or without payload) is then broadcasted to all entities that have subscribed to the node and are authorized to learn about the published information.

We have leveraged the XMPP pubsub mechanism to publish triggers corresponding to events of interest, as registered by client/application queries. UpdateListeners or trigger listeners are Esper entities that are invoked when queries get satisfied. UpdateListeners are pluggable entities in the Esper system, which can peek into event streams and are free to act on the values observed on the event streams. There can be two types of UpdateListeners - (a) static UpdateListeners that are tightly integrated with the server side of the Esper engine, and (b) dynamic client side UpdateListeners that can be provided by clients any time and injected into the Esper system. These ClientSideUpdateListeners can be tailored to queries of interest. When queries get registered into the PQA, the pubsub node handle is passed back to the client, and is used to seed the ClientSideUpdateListener. When the query gets satisfied, the ClientSideUpdateListener uses the pubsub node handle to publish values observed on the event streams. Depending on the design of the ClientSideUpdateListener, it might choose to apply any function (max, current, average etc.) on these values, or ignore some of them. When new values are published on the pubsub nodes, the clients are notified because they subscribe to the same pubsub node handle. The clients/applications can take adaptation actions based on occurrences of event notifications. The ClientSideUpdateListeners have publishing rights on the pubsub nodes and the clients are granted subscribe rights on the nodes. New ClientSideUpdateListeners can be implemented using existing templates in a reasonably straightforward manner, although the currently available set of UpdateListeners, as implemented in PQA, are sufficient for simple use cases.

4.3 Query management

In the PQA architecture, the clients or applications are interested in specific patterns of events. They might be interested in events where values of certain metrics exceed or drop below a threshold, or where a complex condition is met with respect to values of multiple metrics. PQA allows the clients/applications to express these in terms of queries into the PQA system.

PQA exposes a simple API for registering and deleting such queries. The current implementation uses a simple XML-RPC mechanism to expose this API to the clients. The clients/applications can register their queries of interest with PQA and PQA provides a pubsub node handle to the clients corresponding to the registered query. The query management system in PQA hashes these queries and pushes them onto the Esper engine for continuous monitoring of event streams. The queries are injected using a management interface provided by Esper. The clients/applications can then subscribe to the provided pubsub node handle and be notified by the XMPP pubsub mechanism when their queries get satisfied. The query management system is responsible for managing queries from multiple clients. Although not implemented in the current prototype, query management can be extended to handle client authentication over SSL using certificates, as implemented in a separate context by the same authors [2].

In PQA, the queries are expressed using the Esper Event Processing Language (EPL), which is a declarative language for dealing with high frequency time-based event data. EPL statements derive and aggregate information from one or more streams of events, to join or merge event streams, and to feed results from one event stream to subsequent statements. EPL is similar to SQL in it's use of the "select" clause and the "where" clause. However EPL statements use event streams and views instead of tables. Similar to tables in an SQL statement, views define the data available for querying and filtering. Views can represent windows over a stream of events. Views can also sort events, derive statistics from event properties, group events or handle unique event property values.

The following is an example EPL statement that computes the average memory utilization on a node for the last 20 seconds and generates an event of interest when the average memory utilization exceeds 70%.

```
"select avg(memutil) as avgMemUtil
from MemUtilEvent.win:time(20 sec)
where avgMemUtil > 70"
```

When a client registers a query with PQA, it is coupled with a ClientSideUpdateListener that publishes relevant metrics from the event stream when the query is satisfied. In the previous example, the ClientSideUpdateListener may choose to publish the *avgMemUtil* value, or the instantaneous value that triggered the threshold to go above 70.

A more complex example would be a query using joins of several performance metrics from multiple domains.

```
"select
b.metricName as metricName1,  b.metricValue as metricValue1,
m.metricName as metricName2,  m.metricValue as metricValue2
from
BWUtilization.win:length(1) as b,
MemoryUtilization.win:length(1) as m
where b.metricValue > 1.40012E9 and m.metricValue > 70"
```

Here the query concerns instantaneous metric values for bandwidth between two end points and memory utilization at an endpoint. The trigger is raised when both the conditions are met.

4.4 PQA Monitoring Clients

Distributed application execution entails cross-aggregate performance monitoring because a global insight is required to identify performance bottlenecks. It is important to monitor the performance of not only the system and network entities in the different aggregates, but also specific application performance metrics as observed when applications are executing. One of the goals of the PQA tool is to be able to gather these diverse performance metrics from multiple measurement sources belonging to different aggregates. This makes it possible to correlate and filter different observed metrics in an application specific manner through use of queries into PQA.

To this end, PQA includes different monitoring clients that continuously gather data from different sources - system and application specific. The monitoring clients follow a simple design. They interact with measurement sources using their respective native APIs, and collect the metric data. They then construct Esper events corresponding to the observed metric and push event streams into the Esper engine. As of current implementation, PQA includes PerfSONAR and XMPP based clients. It is possible to add new kinds of monitoring clients.

4.4.1 perfSONAR clients

The perfSONAR service responsible for storing measurement information is called a measurement archive (MA). MAs contain two types of information: data and metadata. Data represents the stored measurement results, which are mostly obtained by perfSONAR measurement points (MP). This includes active measurements such as bandwidth and latency, and passive measurements such as SNMP counter records. Metadata is an object that has data associated with it. For example, a bandwidth test identified by its parameters (i.e. endpoints, frequency, duration) is the metadata associated with bandwidth measurement. The MA exposes a web-service interface so that web service clients can query for data/metadata stored in the MA. The PQA perfSONAR clients obtain measurement data by querying available perfSONAR MA services, and then construct Esper events that get continuously inserted as event streams into the Esper engine.

4.4.2 XMPP based clients

Measurement information can be published by applications or system monitoring entities using the XMPP pubsub mechnism, so that interested third parties (other applications, decision engines, workflow tools) get notified of those measurements. This is a general method to disseminate instantaneous performance information. The XMPP based PQA monitoring clients subscribe to relevant pubsub nodes for measurement streams based on configured events. On event notifications on the pubsub nodes, these clients construct Esper events and continuously insert event streams into the Esper engine. Note that these XMPP based PQA monitoring clients are different from application clients that query the PQA and subscribe to XMPP pubsub node handles corresponding to events of interest.

5 Use Cases

The persistent query agent can be used in a multitude of scenarios that require distributed monitoring. These include data-intensive distributed scientific workflow applications running on networked clouds, as in Figure 2, where it is important to monitor the computational performance on nodes and network performance to diagnose any performance problems, both inside the application and at the infrastructure level. For example, third party clients like workflow engines would be able to query PQA about existence of a combination of performance metric thresholds, and would get notified when such conditions arise. This enables efficient feedback for the workflow engine to steer the execution of rest of the workflow. PQA can also be used exclusively at the infrastructure level, monitoring health of distributed infrastructure, and triggering events to relevant infrastructure owners when critical events occur. This entails running continuous health queries, so analysis happens real-time and no archival is required. Other cloud based distributed applications like cloud oriented content delivery networks could leverage PQA to monitor different performance metrics with respect to latency and service rates. PQA would be useful for network monitoring to detect end-to-end bottlenecks, when network paths span multiple domains, and measurement events are made available to PQA.

6 Conclusions and Future Work

We have presented the design, architecture and implementation of a persistent query agent (PQA). PQA enables federated performance monitoring by interacting with multiple aggregates and performance monitoring sources. The PQA implementation leverages an open source complex event processing engine called Esper. The applications/clients register their events of interest using declarative queries expressed in EPL, an SQL-like standard query language. PQA processes event streams and asynchronously sends triggers to applications/clients using an XMPP pubsub mechanism when relevant performance events occur. PQA is scalable - instead of

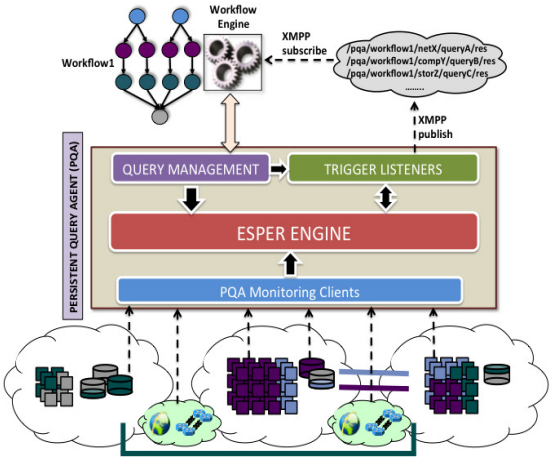

Figure 2: PQA scientific workflow use case

storing all monitoring data for future analysis, PQA observes performance event streams in real-time, and runs persistent queries over streams of events generated from the various performance monitoring sources. The real-time performance feedback is useful in a variety of use cases like workflow scheduling, resource provisioning, anomaly and failure detection etc.

In future, we plan to extend PQA in different directions. We plan to improve the ability to plug in new kinds of monitoring sources dynamically. We are also working on extending the system so that clients are able to add custom update listeners so that they are able to manage what information gets published when an event trigger happens. Our future plans also include coming up with measurement ontologies so that it becomes easier to describe, register and discover new metrics.

Acknowledgments

This work is supported by the Department of Energy award #: DE-FG02-10ER26016/DE-SC0005286.

References

[1] The XMPP Standards Foundation XEP-0060: Publish-Subscribe http://xmpp.org/extensions/xep-0060.html.

[2] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heermann, and J. Chase. Exogeni: A multi-domain infrastructure-as-a-service testbed. In *TRIDENTCOM*, pages 97–113, 2012.

[3] A. Belghith, B. Cousin, S. Lahoud, and S. Ben Hadj Said. Proposal for the configuration of multi-domain network monitoring architecture. In *Information Networking (ICOIN), 2011 International Conference on*, pages 7 –12, jan. 2011.

[4] J. W. Boote, E. L. Boyd, J. Durand, A. Hanemann, L. Kudarimoti, R. Lapacz, N. Simar, and S. Trocha. Towards multi-domain monitoring for the european research networks. In *TNC*, 2005.

[5] E. Boschi, S. DAntonio, P. Malone, and C. Schmoll. Intermon: An architecture for inter-domain monitoring, modelling and simulation. In R. Boutaba, K. Almeroth, R. Puigjaner, S. Shen, and J. Black, editors, *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, volume 3462 of *Lecture Notes in Computer Science*, pages 1397–1400. Springer Berlin Heidelberg, 2005.

[6] P. Calyam, J. Pu, W. Mandrawa, and A. Krishnamurthy. Ontimedetect: Dynamic network anomaly notification in perfsonar deployments. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 328 –337, aug. 2010.

[7] EsperTech. http://www.espertech.com, 2013.

[8] G. Jourjon, T. Rakotoarivelo, and M. Ott. A portal to support rigorous experimental methodology in networking research. In *7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, page 16, Shanghai/China, April 2011.

[9] P. Kanuparthy and C. Dovrolis. Pythia: Distributed Diagnosis of Wide-area Performance Problems. Technical report, Georgia Institute of Technology, 2012.

[10] E. Kissel, A. El-Hassany, G. Fernandes, M. Swany, D. Gunter, T. Samak, and J. Schopf. Scalable integrated performance analysis of multi-gigabit networks. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1227 – 1233, april 2012.

[11] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. Stratan. MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems. *Computer Physics Communications*, 180:2472–2498, Dec. 2009.

[12] A. Margara and G. Cugola. Processing flows of information: from data stream to complex event processing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 359–360, New York, NY, USA, 2011. ACM.

[13] B. Tierney, J. Boote, E. Boyd, A. Brown, M. Grigoriev, J. Metzger, M. Swany, M. Zekauskas, Y.-T. Li, and J. Zurawski. Instantiating a Global Network Measurement Framework. Technical Report LBNL-1452E, Lawrence Berkeley National Lab, 2009.

[14] J. White, G. Jourjon, T. Rakotoarivelo, and M. Ott. Measurement architectures for network experiments with disconnected mobile nodes. In *Internatinonal ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, pages 315–330, Berlin, May 2010. Springer-Verlag.

[15] J. Zurawski, J. Boote, E. Boyd, M. Glowiak, A. Hanemann, M. Swany, and S. Trocha. Hierarchically federated registration and lookup within the perfsonar framework. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 705 –708, 21 2007-yearly 25 2007.