Enabling Persistent Queries for Cross-Aggregate Performance Monitoring

Anirban Mandal, Ilya Baldin, Yufeng Xin, Paul Ruth, and Chris Heerman

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 make 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. The 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. The 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 PQA, and describe some relevant use cases.

INTRODUCTION

Advanced multi-layered networks allow widely distributed computational and storage resources to be connected 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 have 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. Large ensembles of network, computing, and storage resources inevitably experience performance degradations and failures, and applications must be informed of 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, and so on. 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 applications spanning multiple domains without a central point of control. In order to effectively analyze end-to-end bottlenecks (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, e.g., perfSONAR services, network infrastructure monitors, XMPP-based monitoring entities, on-node performance information — OS, system, and application counter data) 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

Anirban Mandal, Ilya Baldin, Yufeng Xin, Paul Ruth, and Chris Heerman are with Renaissance Computing Institute, University of North Carolina at Chapel Hill.

0163-6804/14/$25.00

The applications/ clients register their events of interest using queries and are notified by the PQA when those events occur. The PQA does not store monitoring data. It processes performance event streams in real time using persistent client queries.

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. Thus, 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. The 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 are 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, the 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 article is structured as follows. We describe related work. We present the motivation, design and architecture of PQA. We describe some relevant use cases and conclude the article.

Related Work

perfSONAR [1] 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 that is relevant to a particular distributed application.

There are several other multi-domain monitoring tools. MonALISA [2] is a framework for distributed monitoring. MonALISA 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 [3] is another multi-domain network monitoring framework, which focuses on inter-domain quality of service (QoS) monitoring and large-scale network traffic analysis. Other notable multi-domain network monitoring frameworks are ENTHRONE and EuQoS. In [4], 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-domain monitoring, and to configure the parameters of the measurement points selected.

OMF [5] 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) [6] to collect and store any type of measurements from applications.

There has been some work on automated ways of using and analyzing perfSONAR data. OnTimeDetect [7] 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 [8] is a data analysis tool that makes use of perfSONAR data to detect, localize, and diagnose wide area network performance problems. Kissel *et al.* [9] have developed a measurement and analysis framework to automate troubleshooting of end-to-end network bottlenecks. They integrate measurements from the network, hosts, and application sources using a perfSONAR-compatible common representation, and an extensible session protocol for transport of measurement data, which enables tuning of monitoring frequency and metric selection. They leverage measurement data from perfSONAR, NetLogger traces, and BLiPP for collecting host metrics.

Persistent Query Agent

Although there are 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 cannot 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 always be 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 for providing real-time performance feedback to applications or clients so as to enable steering. The 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 are notified by the PQA when those events occur. The PQA does not store monitoring data. It processes performance event streams in real time using persistent client queries.

The PQA uses an off-the-shelf CEP [10] engine for managing and executing the queries expressed in a standard SQL-like query language. The queries enable complex matching conditions to be expressed, including 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, but are rather "watched" and analyzed as they pass by. In the following sections, we present the design, architecture, and current implementation status of the PQA.

PQA Architecture

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

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

Trigger listeners: They are responsible for publishing events of interest when a query is satisfied. Applications/clients 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. Using the query management interface, applications can add and delete new metrics to the measurement registry, and register and delete queries based on those metrics. The query manager 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 event 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.

Esper Persistent Query CEP Engine

Esper is a framework for performing complex event processing, available open source from EsperTech [11]. 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, service level agreement [SLA] monitoring), and sensor network applications (RFID reading, scheduling and control of fabrication lines) [11].

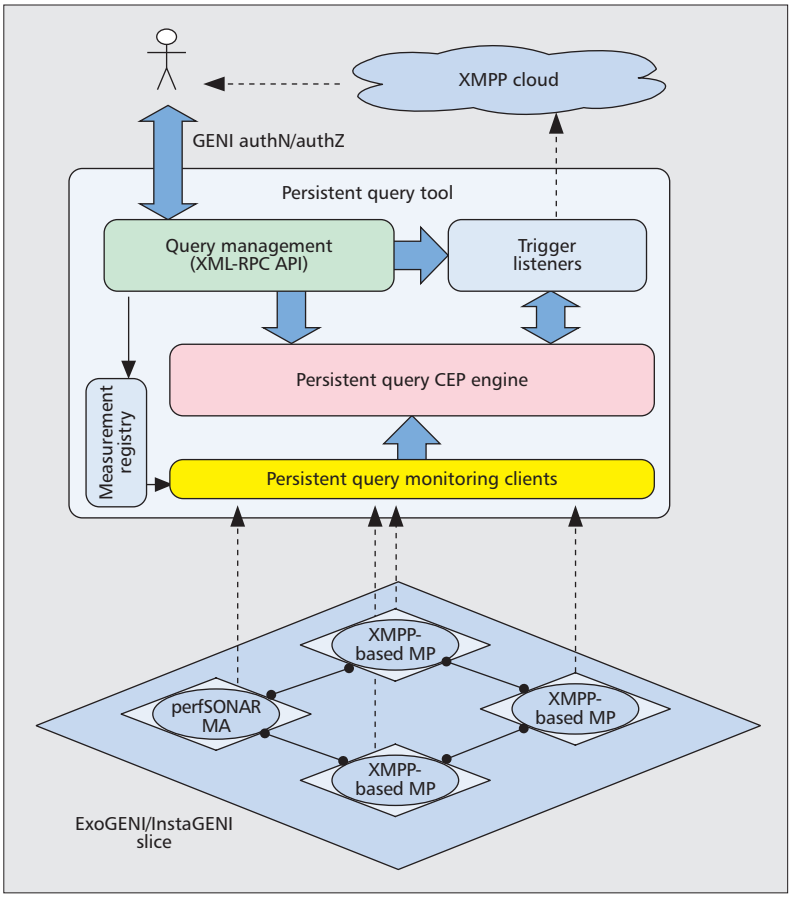

Figure 1. PQA architecture.

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

The Esper Event Processing Language (EPL) allows queries to be registered 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 the expression of complex matching conditions including 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 documents, or

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.

java.util.Map, which promotes reuse of existing systems acting as message publishers. Esper offers a mature application programming interface (API) with features like

- Event stream processing and pattern matching. Esper provides:
  - –*Sliding windows*: time, length, sorted, ranked, accumulating, and so on.
  - –*Named windows* with explicit sharing of data windows between statements.
  - –*Stream operations* like grouping, aggregation, sorting, filtering, merging, splitting, or duplicating of event streams.
  - –*Familiar SQL-standard-based continuous query language* using insert into, select, from, where, group-by, having, order-by, and distinct clauses.
  - –*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 the 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.

XMPP Publish Trigger Listeners

The XMPP pubsub specification [12] 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 broadcast to all entities that have subscribed to the node and are authorized to learn about the published information. The XMPP pubsub clients can authenticate with the XMPP server securely using X.509 certificates. For authorization of clients, we extended an existing XMPP server (Openfire) code to enable verification of XMLSEC credentials, which are signed XML documents generated using the XMLSEC library. This allows authenticated clients to publish/subscribe to specific pubsub nodes based on the rights in their credentials. The XMPP client authorization work was done as part of a separate project [13].

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:

- Static UpdateListeners, which are tightly integrated with the server side of the Esper engine
- Dynamic client side UpdateListeners,which 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 the PQA, are sufficient for simple use cases.

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. The PQA allows the clients/applications to express these in terms of queries into the PQA system.

The 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 the PQA, and the PQA provides a pubsub node handle to the clients corresponding to the registered query. The query management system in the 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 are 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 [14].

In the 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 its 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 s and generates an event of interest when the average memory utilization exceeds 70 percent.

```
"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.met-
ricValue as metricValue1,
m.metricName as metricName2, m.met-
ricValue 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 endpoints and memory utilization at an endpoint. The trigger is raised when both the conditions are met.

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.

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.

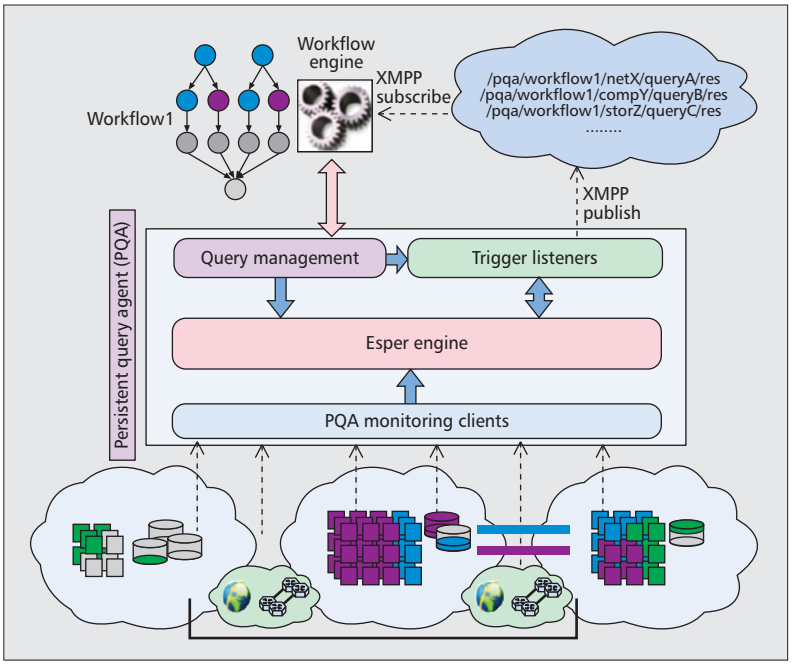

Figure 2. PQA scientific workflow use case.

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 (MPs). This includes active measurements such as bandwidth and latency, and passive measurements such as Simple Network Management Protocol (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-services 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.

XMPP-based Clients — Measurement information can be published by applications or system monitoring entities using the XMPP pubsub mechanism 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

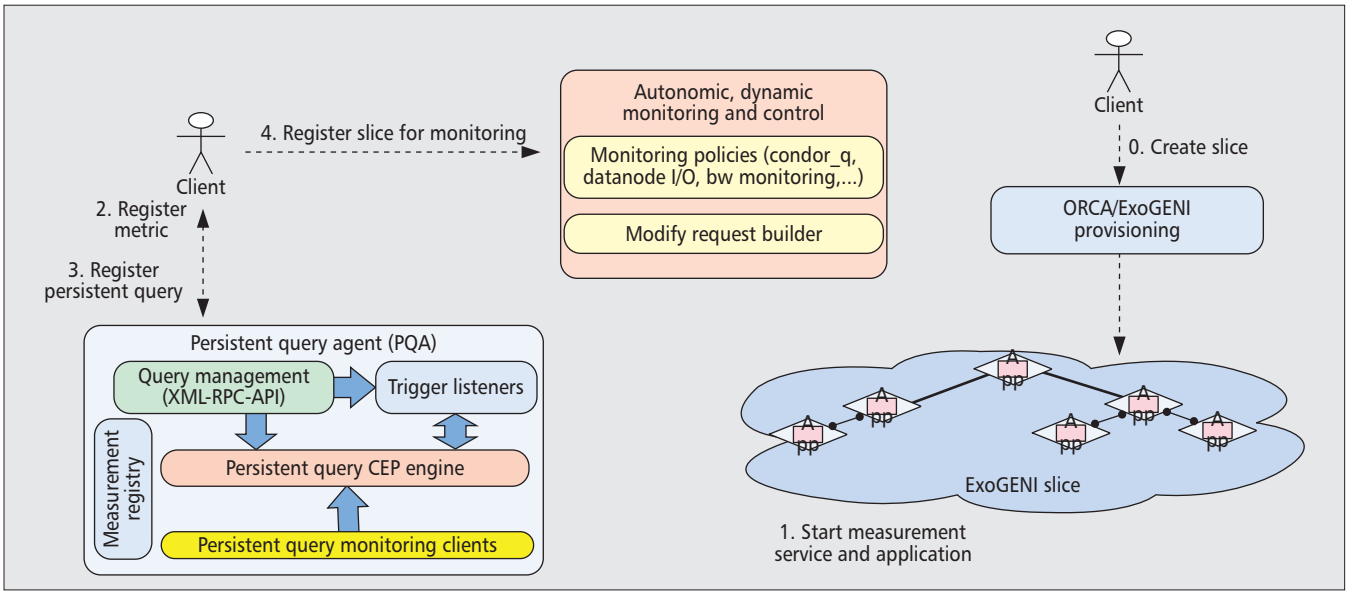

Figure 3. Register metric and persistent query with PQA; register slice for monitoring.

into the Esper engine. Note that these XMPP-based PQA monitoring clients are different from application clients, which query the PQA and subscribe to XMPP pubsub node handles corresponding to events of interest.

Current Status of PQA Software

The current version of the PQA software is available for download from the following website [15]. The software includes the features described above — query management, dynamic registration of new metrics, simple trigger listeners, and an integrated Esper CEP engine. It also includes two types of example monitoring clients — a perfSONAR-based client and an XMPP-based client. Future extensions are envisioned to support new types of PQA monitoring clients as we start ingesting monitoring data from a multitude of sources. We have discussed our work on the PQA at GENI engineering conferences, and are actively engaging with perfSONAR and other communities interested in multi-domain distributed monitoring capabilities.

Use Cases

The PQA can be used in a multitude of scenarios that require monitoring and distributed, scalable, asynchronous notifications. These include data-intensive distributed scientific workflow applications running on networked clouds, as in Fig. 2, where it is important to monitor the performance of the application to determine anomalies, both inside the application and at the infrastructure level. In this section, we present this use case, which was demonstrated as part of the SCInet Network Research Exhibition program during the Supercomputing 2013 conference.

As part of this work, we developed a capability for closed-loop feedback control using persistent queries on monitoring data. We utilized the ExoGENI testbed to execute *Montage*, a data-intensive scientific workflow application, using the *Pegasus* workflow manager. The slice included dynamically provisioned connections over I2, ESnet, NLR, and BEN, as well as a connection over SCInet to a data storage host located in RENCI's SC '13 booth. Montage was deployed on a virtualized HTCondor environment provisioned dynamically from resources on the ExoGENI testbed. Slices were provisioned from the ExoGENI testbed by sending requests for virtual topologies consisting of a set of virtual machines (VMs) connected via a broadcast link with a specified bandwidth.

We showed dynamic scaling of the virtualized HTCondor cluster based on measurements of HTCondor idle job queue length. The measurements were continuously published into an XMPP space by an agent running on the HTCondor master VM. The measurements were consumed by an XMPP-based PQA client, and were fed into the Esper engine. The "IdleJobs" metric was registered with the PQA, and an EPL query expressing a constraint was registered with the PQA, as shown in Fig. 3. In this case, we chose a simple constraint that if idle job queue length for a specified period of time in the past exceeds a certain threshold, a notification is sent. The asynchronous notifications were sent to an "autonomic, dynamic monitoring and control" agent. When the "autonomic, dynamic monitoring and control" agent received the triggers from the PQA, it initiated scaling actions (growing or shrinking the virtualized HTCondor cluster) by sending modification requests to ExoGENI, the resource provisioning system. Here, the "autonomic, dynamic monitoring and control" agent was a subscriber of persistent query notifications, and used simple internal policies to send modification requests. The adaptation flow is shown in Fig. 4.

We are currently investigating more advanced versions of this use case, where more complex sets of multi-domain metrics could be used to experiment with various dynamic control policies like adjusting future resource provisioning deci-

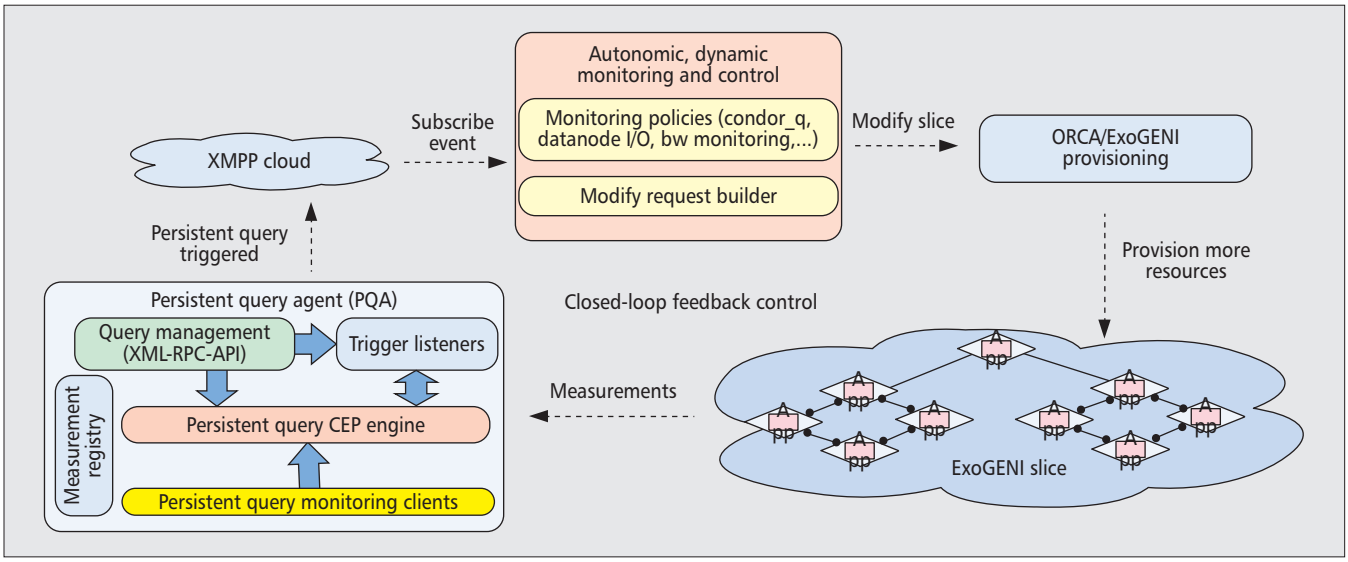

Figure 4. Dynamic monitoring and closed-loop feedback control using persistent queries.

sions. The 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 that analysis happens in real time, and no archives are 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. The PQA would be useful for network monitoring to detect end-to-end bottlenecks, when network paths span multiple domains, and measurement streams are made available to PQA.

Conclusions and Future Work

We have presented the design, architecture, and implementation of a persistent query agent. A 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. The PQA processes event streams and asynchronously sends triggers to applications/clients using an XMPP pubsub mechanism when relevant performance events occur. A PQA is scalable — instead of storing all monitoring data for future analysis, a 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 provisioning resources for scientific workflows, and anomaly and failure detection.

In the 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 and hence manage which 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 and discover new metrics.

Acknowledgments

This work is supported by the DOE DROPS project, award # DE-FG02-10ER26016/DE-SC0005286, and the DOE SUPER project, award # DE-FG02-11ER26050/DE-SC0006925.

References

[1] B. Tierney *et al.*, "Instantiating a Global Network Measurement Framework," tech. rep. LBNL-1452E, Lawrence Berkeley Nat'l. Lab., 2009.
[2] I. Legrand *et al.*, "MonALISA: An Agent-Based, Dynamic Service System to Monitor, Control and Optimize Distributed Systems," *Computer Physics Commun.*, vol. 180, Dec. 2009, pp. 2472–98.
[3] E. Boschi *et al.*, "INTERMON: An Architecture for Inter-Domain Monitoring, Modelling and Simulation," *NETWORKING 2005, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, R. Boutaba *et al.*, Eds., vol. 3462 of LNCS, Springer, 2005, pp. 1397–1400.
[4] A. Belghith *et al.*, "Proposal for the Configuration of Multidomain Network Monitoring Architecture," *2011 Int'l. Conf. Info. Networking*, Jan. 2011, pp. 7–12.
[5] G. Jourjon, T. Rakotoarivelo, and M. Ott, "A Portal to Support Rigorous Experimental Methodology in Networking Research," *7th Int'l. ICST Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities*, Shanghai, China, Apr. 2011, p. 16.
[6] J. White *et al.*, "Measurement Architectures for Network Experiments with Disconnected Mobile Nodes," *Int'l. ICST Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities*, Berlin, Germany, Springer-Verlag, May 2010, pp. 315–30.
[7] P. Calyam *et al.*, "Ontimedetect: Dynamic Network Anomaly Notification in Perfsonar Deployments," *2010 IEEE Int'l. Symp. Modeling, Analysis Simulation of Computer and Telecommunication Systems*, Aug. 2010, pp. 328–37.
[8] P. Kanuparthy *et al.*, "Pythia: Detection, Localization, and Diagnosis of Performance Problems," *IEEE Commun. Mag.*, vol. 51, no. 11, 2013, pp. 55–62.
[9] E. Kissel *et al.*, "Scalable Integrated Performance Analysis of Multi-Gigabit Networks," *2012 IEEE NOMS*, Apr. 2012, pp. 1227–33.

[10] A. Margara and G. Cugola, "Processing Flows of Information: From Data Stream to Complex Event Processing," *Proc. 5th ACM Int'l. Conf. Distributed event-based System*, *DEBS '11*, 2011, pp. 359–60.
[11] EsperTech, http://www.espertech.com, 2013.
[12] "The XMPP Standards Foundation XEP-0060: Publish-Subscribe http://xmpp.org/extensions/xep- 0060.html."
[13] GENI Instrumentation and Measurement Framework (IMF) Project Wiki Document on XMPP Client Authorization, http://groups.geni.net/geni/attachment/wiki/IMFGEC13- QSR/XMPPAuthCred-IMF.docx.
[14] I. Baldine *et al.*, "ExoGENI: A Multi-Domain Infrastructure-as-a-Service Testbed," *TRIDENTCOM*, 2012, pp. 97–113.
[15] DOE DROPS project website, https://code.renci.org/gf/project/drops/.

BIOGRAPHIES

ANIRBAN MANDAL (anirban@renci.org) received both his M.S. (2002) and Ph.D. (2006) degrees in computer science from Rice University. He currently works as a research scientist at RENCI, University of North Carolina at Chapel Hill. His research interests include provisioning, scheduling, performance analysis, and measurements for distributed computing systems, cloud computing, and scientific workflows.

ILYA BALDIN (ibaldin@renci.org) received both his M.S. (1995) and Ph.D. (1998) degrees in computer science from North Carolina State University. He is currently the director of the Network Research and Infrastructure group at RENCI, University of North Carolina at Chapel Hill. His research interests include federated distributed networked systems and architectures, high-speed optical network architectures, cross-layer interactions, novel signaling schemes, and network security.

YUFENG XIN (yxin@renci.org) is a senior networking researcher at RENCI, University of North Carolina at Chapel Hill. He was a senior scientist at MCNC, RTP, NC, and a research associate at the University of Maryland, College Park. His research focuses on high-speed networks, cloud computing, and smart grid communications. He obtained his Ph.D. in operations research and computer science from North Carolina State University in 2002.

PAUL RUTH (pruth@renci.org) is a senior distributed systems researcher at RENCI, University of North Carolina at Chapel Hill. His research interests include machine and network virtualization with emphasis on increasing the performance of scientific applications. He received his Ph.D. in computer science from Purdue University in 2007.

CHRIS HEERMAN (ckh@renci.org) is a senior network research scientist at RENCI, University of North Carolina at Chapel Hill, where he manages and operates a metro area dark fiber based research facility. His primary focus is to enable network research and develop software defined networking solutions. Past experience includes network engineering, project management, and software development for large service providers in both higher education and industry. He holds an M.S. degree in electrical engineering from George Mason University.