Maintaining QoS for Publish/Subscribe Middleware via Autonomic Adaptation

Joe Hoffert and Douglas C. Schmidt
Vanderbilt University
Nashville, TN, USA
{jhoffert, schmidt}@dre.vanderbilt.edu

***Abstract*—Quality of Service (QoS)-enabled Publish/Subscribe (pub/sub) middleware provides powerful support for data dissemination. It is hard, however, to maintain specified QoS properties (such as reliability and latency) in dynamic environments (such as disaster relief operations or power grids). For example, managing QoS manually is not feasible in dynamic systems due to slow human response times and the complexity of managing multiple interrelated QoS settings.**

Autonomic adaptation provides a promising approach to maintaining QoS properties of QoS-enabled pub/sub middleware in dynamic environments. By monitoring key system resources and environment aspects, changes to middleware mechanisms (*e.g.*, associations of publishers and subscribers to transport protocols) can be made autonomically to maintain specified QoS. This paper describes the research we are conducting on highly configurable QoS-enabled middleware, adaptive transport protocols, system monitoring, supervised machine learning techniques, and autonomic adaptation to maintain specified QoS as the system environment and configuration dynamically changes.

I. Introduction

Emerging trends and challenges. The number and type of distributed systems that utilize publish/subscribe (pub/sub) technologies have grown due to the advantages of performance, cost, and scale as compared to single computers [1], [2]. Examples of pub/sub middleware include Web Services Brokered Notification (www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn), the Java Message Service (JMS) (java.sun.com/products/jms), the CORBA Event Service (www.omg.org/technology/documents/formal/event_service.htm), and the Data Distribution Service (DDS) (www.omg.org/spec/DDS). These technologies support data propagation throughout a system using an anonymous subscription model that decouples event suppliers and consumers.

Pub/sub middleware is used in a wide spectrum of application domains, ranging from shipboard computing environments to fractionated spacecraft constellations. The middleware supports policies that affect the end-to-end QoS of the system. Common policies across different middleware include *persistence* (*i.e.*, saving data for current subscribers), *durability* (*i.e.*, saving data for subsequent subscribers), and *grouped data transfer* (*i.e.*, transmitting a group of data as an atomic unit).

While tunable policies provide fine-grained control of system QoS, several challenges emerge when developing pub/sub systems deployed in dynamic environments. Mechanisms used by the middleware to ensure certain QoS properties for a given environment configuration may not be applicable for a different environment configuration. For example, a simple unicast protocol, such as UDP, may provide adequate QoS regarding latency when a publisher sends to a small number of subscribers. UDP could incur too much latency, however, when used for a large number of subscribers due to the publisher needing to send to each individual subscriber.

Challenges also arise when managing multiple QoS policies that interact with each other. For example, a system might specify low latency QoS and reliability QoS, which can affect latency due to data loss discovery and recovery. Certain transport protocols, such as UDP, provide low overhead but no end-to-end reliability. Other protocols, such as TCP, provide reliability but unbounded latencies due to acknowledgment-based retransmissions. Still other protocols balance reliability and low latency, but provide benefit over other protocols only for specific environment configurations. Determining when to modify parameters of a particular transport protocol or switch from one transport protocol to another can be a complex decision. Moreover, human intervention might not be responsive enough for the timeliness requirements of the system.

Research approach. The remainder of this document describes the research we are conducting to address the challenges of maintaining QoS in dynamic environments by integrating and enhancing the following technologies: (1) QoS-enabled pub/sub middleware, (2) adaptive transport protocols, (3) environment monitoring, (4) supervised machine learning, and (5) autonomic adaptation of transport protocols to manage specified QoS within dynamic environments.

II. Motivating Example - Search and Rescue (SAR) Operations for Disaster Recovery

To motivate the need for autonomic adaptation of QoS-enabled pub/sub middleware, this section describes the research challenges associated with search and rescue (SAR) operations. These operations help locate and extract survivors in a large metropolitan area after a regional catastrophe, such as a hurricane, earthquake, or tornado. SAR operations utilize unmanned aerial vehicles (UAVs), existing operational monitoring infrastructure (*e.g.*, building or traffic light mounted cameras intended for security or traffic monitoring), and (temporary) datacenters to receive, process, and transmit event stream data from sensors and monitors to emergency vehicles that can be dispatched to areas where survivors are identified.

Figure 1 shows an example SAR scenario where infrared scans along with GPS coordinates are provided by UAVs and video feeds are provided by existing infrastructure cameras. These infrared scans and video feeds are then sent to a datacenter, where they are processed by fusion applications to detect survivors. Once a survivor is detected the application can develop a three dimensional view and highly accurate position information so that rescue operations can commence.

There are several key challenges that arise with SAR operations in dynamic environments.

Fig. 1. **Search and Rescue Motivating Example**

A. Challenge 1: Timely Adaptation to Dynamic Environments

Due to the dynamic environment inherent in the aftermath of a disaster SAR operations must adjust in a timely manner as the environment changes. If SAR operations cannot adjust quickly enough they will fail to perform adequately given a shift in resources. If resources are lost or withdrawn—or demand for information increases—SAR operations must be configured to accommodate these changes with appropriate responsiveness to maintain a minimum level of service. If resources increase or demand decreases, SAR operations should take advantage of these as quickly as possible to provide higher fidelity or more expansive coverage. Manual modification is often too slow and error-prone to maintain QoS.

B. Challenge 2: Managing Interacting QoS Requirements

SAR operations must manage multiple QoS requirements that interact with each other, *e.g.*, data reliability so that enough data is received to be useful and low latency for soft realtime data so that infrared scans from UAVs or video from cameras mounted atop traffic lights do not arrive after they are needed. The streamed data must be received soon enough so that successive dependent data can be used as well. For example, MPEG I frame data must be received in a timely manner so that successive dependent B and P frame data can be used before the next I frame makes them obsolete. Otherwise, not only is the data unnecessary, but sending and processing the data has consumed limited resources.

C. Challenge 3: Scaling to Large Numbers of Receivers

For a regional or national disaster, a multitude of organizations would register interest not only in the individual video and infrared scans for various applications, but also in the fused data for the SAR operations. For example, fire detection applications and power grid assessment applications can use infrared scans to detect fires and working HVAC systems respectively. Likewise, security monitoring and structural damage applications can use video stream data to detect looting and unsafe buildings respectively. Moreover, federal, state, and local authorities would want to register interest in the fused SAR data to monitor the status of current SAR operations.

D. Challenge 4: Specifying Standardized and Robust QoS

SAR applications should be developed with the focus on application logic rather than on complex or custom formats for specifying QoS. Time spent learning a customized or complex format for QoS is time taken from developing the SAR application itself. Moreover, learning a custom format will not be applicable for other applications that use a different QoS format. Application developers also need support for a wide range of QoS to handle dynamic environments.

III. Overview of Related Work

This section analyzes related research efforts in light of the challenges presented in Section II.

Machine learning in support of autonomic adaptation. Vienne and Sourrouille [3] present the Dynamic Control of Behavior based on Learning (DCBL) middleware that incorporates reinforcement machine learning in support of autonomic control for QoS management. Reinforcement machine learning not only allows DCBL to handle unexpected changes but also reduces the overall system knowledge required by the system developers. System developers provide an XML description of the system, which DCBL then uses together with an internal representation of the managed system to select appropriate QoS dynamically.

DCBL provides a customized QoS specification that does not address Challenge 4 in Section II-D. DCBL focuses on single computers rather than addressing scalable distributed systems, as outlined with Challenge 3 in Section II-C. Moreover, DCBL requires developers to specify in an XML file the selection of operating modes given a QoS level along with execution paths, which leaves handling Challenge 2 in Section II-B to developers.

Infrastructure for autonomic computing. Grace *et al.* [4] describe an architecture metamodel for adapting components that implement coordination for reflective middleware distributed across peer devices. This work also investigates supporting reconfiguration types in various environmental conditions. The proposed architecture metamodel, however, only provides proposed infrastructure for autonomic adaptation and reconfiguration and does not directly address the challenges in Section II.

Valetto *et al.* [5] developed network features in support of service awareness to enable autonomic behavior. Their work targets communication services within a Session Initiation Protocol (SIP) enabled network to communicate monitoring, deployment, and advertising information. As an autonomic computing infrastructure, however, this work does not directly address any of the challenges in Section II.

Autonomic adaption of service level agreements. Herssens *et al.* [6] describe work that centers around autonomically adapting service level agreements (SLAs) when the context of the specified service changes. This work acknowledges that both offered and the requested QoS for Web services might vary over the course of the interaction and accordingly modifies the SLA between the client and the server as appropriate. This work does not address Challenge 1 in Section II-A, but rather negotiates the QoS agreement to fit the dynamic environment.

Autonomic adaption of networks. The Autonomic Realtime Multicast Distribution System (ARMDS) [7] is a framework that focuses on decreasing excessive variance in service

quality for multicast data across the Internet. The framework supports the autonomic adaptation of the network nodes forming the multicast graph so that the consistency of service delivery is enhanced. The framework includes (1) high level descriptions of policies and objectives, (2) a multicast topology management protocol supported by network nodes, (3) measurement and monitoring infrastructure, and (4) a control component that autonomously manipulates the protocol and infrastructure to reduce variance. ARMDS does not address Challenge 2 in Section II-B, however, nor does it address Challenge 4 in Section II-D.

IV. Proposed Research

A. Approach

The proposed research combines and enhances the following technologies to resolve the challenges presented in Section II and address the gaps in related work described in Section III.

- Standard QoS-enabled pub/sub middleware addresses the scalability of Challenge 2 in Section II-C by decoupling data senders from data receivers. Applications interested in published data can receive it any time without knowledge of the data sender. Moreover, standard QoS-enabled middleware addresses the QoS standardization of Challenge 4 in Section II-D.
- Supervised machine learning helps address Challenge 1 in Section II-A and Challenge 2 in Section II-B by selecting in a timely manner an appropriate transport protocol and protocol parameters given specified QoS and a particular environment configuration. The machine learning component includes features for several different environment configurations and supervised training to learn the correct protocol and parameters. The machine learning will interpolate and extrapolate its learning based on the current environment configuration, which might not have been included in the supervised training.
- Adaptive network transports works to address Challenge 1 in Section II-A and Challenge 2 in Section II-B by providing the infrastructure flexibility to maintain interrelated QoS even within dynamic environments. For some environment configurations one particular transport protocol provides the required QoS. For other environment configurations another transport protocol provides the specified QoS. Adaptive network transports that support not only fine tuning a protocol's parameters but also switching from one protocol to another provide functionality needed in dynamic environments.
- Environment monitoring works to address Challenge 1 in Section II-A by providing environment configuration information. Relevant environment configuration values are monitored as needed such as the number of subscribers, the percentage of network packet loss, and the sending rate of the data. These monitored values are input to the machine learning component to determine an appropriate network transport and accompanying parameters.
- Autonomic adaptation addresses Challenge 1 in Section II-A and Challenge 2 in Section II-B by (1) querying relevant values from the environment monitoring, (2) activating the machine learning component which will determine an appropriate transport protocol and parameters, (3) retrieving the recommended protocol settings, and (4) transitioning the adaptive network transports to use the recommended settings.

B. Current Status and Plan

Our current work has integrated and enhanced the OpenDDS implementation (www.opendds.org) of the OMG Data Distribution Service (DDS) standard (www.omg.org/spec/DDS) with the Adaptive Network Transports (ANT) framework. OpenDDS is standards-based anonymous QoS-enabled pub/-sub middleware for exchanging data in event-based distributed systems. It provides a global data store in which publishers and subscribers write and read data, respectively, so applications can communicate by publishing information they have and subscribing to information they need in a timely manner.

We chose the OpenDDS implementation due to its (1) source code being freely available, which facilities modification and experimentation, and (2) support for a *pluggable transport framework* that allows application developers to create custom transport protocols for sending/receiving data. We chose the ANT framework due to its infrastructure for composing transport protocols. ANT builds upon the properties provided by the scalable reliable multicast-based Ricochet transport protocol [8]. ANT also provides a modular framework whereby protocol modules can be tuned, enhanced, and replaced to maintain specified QoS.

Our recent work [9] has evaluated the performance of the transport protocols integrated with OpenDDS in various environment configurations. We are using the data from these performance evaluations as input for supervised machine learning techniques, *e.g.*, neural networks, decision trees. We will evaluate these techniques to see which most appropriately meets the research challenges.

References

[1] Y. Huang and D. Gannon, "A comparative study of web services-based event notification specifications," *Proceedings of the International Conference on Parallel Processing Workshops*, vol. 0, pp. 7–14, 2006.

[2] S. Tarkoma and K. Raatikainen, "State of the Art Review of Distributed Event Systems," University of Helsinki, Tech. Rep. C0-04, 2006.

[3] P. Vienne and J.-L. Sourrouille, "A middleware for autonomic qos management based on learning," in *SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware*. New York, NY, USA: ACM, 2005, pp. 1–8.

[4] P. Grace, G. Coulson, G. S. Blair, and B. Porter, "A distributed architecture meta-model for self-managed middleware," in *ARM '06: Proceedings of the 5th workshop on Adaptive and reflective middleware (ARM '06)*. New York, NY, USA: ACM, 2006, p. 3.

[5] G. Valetto, L. W. Goix, and G. Delaire, "Towards service awareness and autonomic features in a sip-enabled network," in *Autonomic Communication*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 202–213.

[6] C. Herssens, S. Faulkner, and I. J. Jureta, "Context-driven autonomic adaptation of sla," in *ICSOC '08: Proceedings of the 6th International Conference on Service-Oriented Computing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 362–377.

[7] B. Brynjulfsson, G. Hjalmtysson, K. Katrinis, and B. Plattner, "Autonomic network-layer multicast service towards consistent service quality," in *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications*. Washington, DC, USA: IEEE Computer Society, April 2006, pp. 494–498.

[8] M. B. *et al*, "Ricochet: Lateral error correction for time-critical multicast," in *NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation*, Boston, MA, 2007.

[9] F. Wolf, J. Balasubramanian, A. Gokhale, and D. C. Schmidt, "Component Replication based on Failover Units," in *To Appear in the Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '09)*, Aug 2009.

Joe Hoffert is a Ph.D. student (advanced standing) in the Department of Electrical Engineering and Computer Science at Vanderbilt University. His research focuses on enhancing productivity and flexibility of QoS-enabled pub/sub middleware. As part of this research he has developed the Distributed QoS Modeling Language (DQML) which is a domain-specific modeling language (DSML) to manage the complexities of QoS configurations for distributed pub/sub middleware. DQML addresses key challenges of pub/sub middleware, including (1) managing QoS policy configuration variability, (2) developing semantically compatible configurations, and (3) correctly transforming QoS policy configurations from design to implementation. DQML also reduces developer effort by as much as 54% when compared to manual methods.

Mr. Hoffert previously worked for Boeing in the area of model-based integration of embedded systems and high-fidelity flight simulation. Within Boeing's Phantom Works R&D organization he developed an instrumentation interface to provide metrics feedback from real-time embedded avionics systems. Within Boeing's Center for Integrated Defense Simulation he developed a scalable and high fidelity framework for distributed and network-centric operations simulations. This work included publishing and subscribing of information and intelligent filtering of data to maximize networking resources.

Prior to working at Boeing, he was a research associate at Washington University in St. Louis where he worked in (1) the Distributed Object Computing (DOC) group enhancing the ADAPTIVE Communication Environment (ACE) to portably use a POSIX-like C wrapper for POSIX functionality, (2) the Applied Research Laboratory enhancing ACE to leverage ATM technologies, and (3) the Distributed Programming Environment group developing the Programmer's Playground, a software library and run-time support system to ease development of distributed applications.

He has over 15 years of industry experience in the areas of object-oriented technologies and over 5 years industry experience with distributed, realtime, and embedded (DRE) systems. He has 9 refereed publications. Additional information is available at http://www.dre.vanderbilt.edu/~jhoffert/cv.html.

Mr. Hoffert received his B.A. in Math/C.S. from Mount Vernon Nazarene College (OH) and his M.S. in C.S. from the University of Cincinnati (OH). He entered Vanderbilt's Ph.D. program in the fall of 2006 and expects to receive his Ph.D. in late 2011/early 2012.

SDG