Adapting Distributed Real-time and Embedded Pub/Sub Middleware for Cloud Computing Environments [⋆]

Joe Hoffert[⋆⋆], Douglas C. Schmidt, and Aniruddha Gokhale

Vanderbilt University, VU Station B #1829, 2015 Terrace Place, Nashville, TN 37203

Abstract. Enterprise distributed real-time and embedded (DRE) publish/subscribe (pub/sub) systems manage resources and data that are vital to users. Cloud computing—where computing resources are provisioned elastically and leased as a service—is an increasingly popular deployment paradigm. Enterprise DRE pub/-sub systems can leverage cloud computing provisioning services to execute needed functionality when on-site computing resources are not available. Although cloud computing provides flexible on-demand computing and networking resources, enterprise DRE pub/sub systems often cannot accurately characterize their behavior *a priori* for the variety of resource configurations cloud computing supplies (*e.g.*, CPU and network bandwidth), which makes it hard for DRE systems to leverage conventional cloud computing platforms.

This paper provides two contributions to the study of how autonomic configuration of DRE pub/sub middleware can provision and use on-demand cloud resources effectively. We first describe how supervised machine learning can configure DRE pub/sub middleware services and transport protocols autonomically to support end-to-end quality-of-service (QoS) requirements based on cloud computing resources. We then present results that empirically validate how computing and networking resources affect enterprise DRE pub/sub system QoS. These results show how supervised machine learning can configure DRE pub/sub middleware adaptively in < 10 μsec with bounded time complexity to support key QoS reliability and latency requirements.

Keywords: Autonomic configuration, pub/sub middleware, DRE systems, cloud computing

1 Introduction

Emerging trends and challenges. Enterprise distributed real-time and embedded (DRE) publish/subscribe (pub/sub) systems manage data and resources that are critical to the ongoing system operations. Examples include testing and training of experimental aircraft across a large geographic area, air traffic management systems, and disaster recovery operations. These types of enterprise DRE systems must be configured correctly to leverage available resources and respond to the system deployment environment. For example, search and rescue missions in disaster recovery operations need to configure the image resolution used to detect and track survivors depending on the available resources (*e.g.*, computing power and network bandwidth) [20].

[⋆] This work is sponsored by NSF TRUST and AFRL.

[⋆⋆] Contact author's email address: `jhoffert@dre.vanderbilt.edu`

Many enterprise DRE systems are implemented and developed for a specific computing/networking platform and deployed with the expectation of specific computing and networking resources being available at runtime. This approach simplifies development complexity since system developers need only focus on how the system behaves in one operating environment. Thus considerations of multiple infrastructure platforms are ameliorated with respect to system quality-of-service (QoS) properties (*e.g.*, responsiveness of computing platform, latency and reliability of networked data, etc.). Focusing on only a single operating environment, however, decreases the flexibility of the system and makes it hard to integrate into different operating environments, *e.g.*, porting to new computing and networking hardware.

Cloud computing [6, 17] is an increasingly popular infrastructure paradigm where computing and networking resources are provided to a system or application as a service—typically for a "pay-as-you-go" usage fee. Provisioning services in cloud environments relieve enterprise operators of many tedious tasks associated with managing hardware and software resources used by systems and applications. Cloud computing also provides enterprise application developers and operators with additional flexibility by virtualizing resources, such as providing virtual machines that can differ from the actual hardware machines used.

Several pub/sub middleware platforms (such as the Java Message Service [16], and Web Services Brokered Notification [14]) can (1) leverage cloud environments, (2) support large-scale data-centric distributed systems, and (3) ease development and deployment of these systems. These pub/sub platforms, however, do not support fine-grained and robust QoS that are needed for enterprise DRE systems. Some large-scale distributed system platforms, such as the Global Information Grid [1] and Network-centric Enterprise Services [2], require rapid response, reliability, bandwidth guarantees, scalability, and fault-tolerance.

Conversely, conventional cloud environments are problematic for enterprise DRE systems since applications within these systems often cannot characterize the utilization of their specific resources (*e.g.*, CPU speeds and memory) accurately *a priori*. Consequently, applications in DRE systems may need to adjust to the available resources supplied by the cloud environment (*e.g.*, using compression algorithms optimized for given CPU power and memory) since the presence/absence of these resources affect timeliness and other QoS properties crucial to proper operation. If these adjustments take too long the mission that the DRE system supports could be jeopardized.

Configuring an enterprise DRE pub/sub system in a cloud environment is hard because the DRE system must understand how the computing and networking resources affect end-to-end QoS. For example, transport protocols provide different types of QoS (*e.g.*, reliability and latency) that must be configured in conjunction with the pub/sub middleware. To work properly, however, QoS-enabled pub/sub middleware must understand how these protocols behave with different cloud infrastructures. Likewise, the middleware must be configured with appropriate transport protocols to support the required end-to-end QoS. Manual or *ad hoc* configuration of the transport and middleware can be tedious, error-prone, and time consuming.

Solution approach $\rightarrow$ Supervised Machine Learning for Autonomous Configuration of DRE Pub/Sub Middleware in Cloud Computing Environments. This paper describes how we are (1) evaluating multiple QoS concerns (*i.e.*, reliability and latency) based on differences in computing and networking resources and (2) configuring QoS-enabled pub/sub middleware autonomically for cloud environments based on these evaluations. We have prototyped this approach in the *ADAptive Middleware And Network Transports* (ADAMANT) platform, which addresses the problem of configuring QoS-enabled DRE pub/sub middleware for cloud environments. Our approach provides the following contributions to research on autonomic configuration of DRE pub/sub middleware in cloud environments:

• **Supervised machine learning as a knowledge base to provide fast and predictable resource management in cloud environments**. *Artificial Neural Network* (ANN) tools determine in a timely manner the appropriate transport protocol for the QoS-enabled pub/sub middleware platform given the computing resources available in the cloud environment. ANN tools are trained on particular computing and networking configurations to provide the best QoS support for those configurations. Moreover, they provide predictable response times needed for DRE systems.

• **Configuration of DRE pub/sub middleware based on guidance from supervised machine learning**. Our ADAMANT middleware uses the *Adaptive Network Transports* (ANT) [10] to select the transport protocol(s) that best address multiple QoS concerns for given computing resources. ANT provides infrastructure for composing and configuring transport protocols using the scalable reliable multicast-based Ricochet transport protocol [3]. Supported protocols such as Ricochet enable trade-offs between latency and reliability to support middleware for enterprise DRE pub/sub systems in cloud environments.

We have implemented ADAMANT using multiple open-source pub/sub middleware implementations (*i.e.*, OpenDDS(`www.opendds.org`) and OpenSplice(`www.openslice.org`)) of the OMG Data Distribution Service (DDS) [18] specification. DDS defines a QoS-enabled DRE pub/sub middleware standard that enables applications to communicate by publishing information they have and subscribing to information they need in a timely manner. The OpenDDS and OpenSplice implementations of DDS provide pluggable protocol frameworks that can support standard transport protocols (such as TCP, UDP, and IP multicast), as well as custom transport protocols (such as Ricochet and reliable multicast).

Our prior work [10, 11] developed composite metrics to evaluate pub/sub middleware with various ANT-based transport protocols based on differences in application parameters (*e.g.*, number of data receivers and data sending rate). We also evaluated multiple approaches for adapting to application parameter changes in a dedicated (*i.e.*, non-cloud) operating environment without regard to changes in computing or networking resources. This paper extends our prior work by (1) evaluating pub/sub middleware in a cloud environment to take into account differences in computing and networking resources and (2) conducting empirical evaluations of an artificial neural network machine learning tool with respect to timeliness and configuration accuracy.

We validated ADAMANT by configuring Emulab (`www.emulab.net`) to emulate a cloud environment that allows test programs to request and configure several types of computing and networking resources on-demand. We then applied several composite metrics developed to ascertain how ADAMANT supports relevant QoS concerns for various Emulab-based cloud configurations. These metrics quantitatively measure multiple interrelated QoS concerns (*i.e.*, latency and reliability) to evaluate QoS mechanisms (such as transport protocols) used in QoS-enabled pub/sub DRE systems. Our supervised machine learning tools use the results of these composite metrics to determine the most appropriate transport protocol to apply in the Emulab cloud environment.

Paper organization. The remainder of this paper is organized as follows: Section 2 describes a representative search and rescue application to motivate the challenges that ADAMANT addresses; Section 3 examines the structure and functionality of ADAMANT and the supervised machine learning technique it uses to guide the configuration process; Section 4 analyzes the results of experiments conducted to validate ADAMANT in a cloud environment; Section 5 compares ADAMANT with related work; and Section 6 presents concluding remarks.

2 Motivating Example - Search and Rescue Operations in the Aftermath of a Regional Disaster

This section describes a representative enterprise DRE pub/sub application in a cloud computing environment to motivate the challenges that ADAMANT addresses.

2.1 Search and Rescue Operations for Disaster Recovery

To highlight the challenges of configuring enterprise DRE pub/sub systems for cloud environments in a timely manner, our work is motivated in the context of supporting search and rescue (SAR) operations that leverage cloud infrastructure. These operations help locate and extract survivors in a large metropolitan area after a regional disaster, such as a hurricane or tornado. SAR operations can use 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 data from various sensors and monitors to emergency vehicles that can be dispatched to areas where survivors are identified.

These datacenters can be mobile (*e.g.*, in truck trailers or large command-and-control aircraft if roads are damaged) and brought into the disaster area as needed. Moreover, these datacenters can be connected to cloud infrastructure via high-speed satellite links [12] since ground-based wired connectivity may not be available due to the disaster. In particular, our work focuses on configuring the QoS-enabled pub/sub middleware used by the temporary *ad hoc* datacenter for data dissemination.

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 facilitated by cloud infrastructure where the data are disseminated, received by fusion applications, and processed to detect survivors. Once survivors are detected, the SAR system will develop a three dimensional view and highly accurate position information so that rescue operations can commence.

A key requirement of data fusion applications within the datacenter is the tight timing bounds on correlated event streams such as the infrared scans coming from UAVs and video coming from cameras mounted atop traffic lights. The event streams need to match up closely so the survivor detection application can produce accurate results. If an infrared data stream is out of sync with a video data stream, the survivor detection

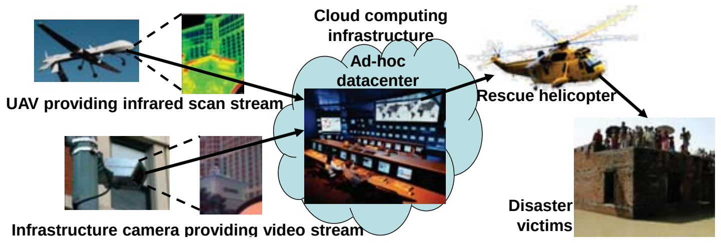

Cloud computing infrastructure
Ad-hoc datacenter
UAV providing infrared scan stream
Rescue helicopter
Infrastructure camera providing video stream
Disaster victims

Fig. 1. Search and Rescue Motivating Example

application can generate a false negative and fail to initiate needed rescue operations. Likewise, without timely data coordination the survivor detection software can generate a false positive thereby expending scarce resources such as rescue workers, rescue vehicles, and data center coordinators unnecessarily. The timeliness and reliability properties of the data are affected by the underlying hardware infrastructure, *e.g.*, faster processors and networks can decrease latency and allow more error correcting data to be transmitted to improve reliability.

SAR operations in the aftermath of a disaster can be impeded by the lack of computing and networking resources needed for an *ad hoc* datacenter. The same disaster that caused missing or stranded people also can diminish or completely eliminate local computing resources. Cloud infrastructure located off-site can provide the needed resources to carry out the SAR operations. Applications using cloud resources can be preempted to support emergency systems such as SAR operations during national crises much as emergency vehicles preempt normal traffic and commandeer the use of traffic lights and roadways. The resources that the cloud provides, however, are not known *a priori*. Thus, the effective QoS for the SAR operations are dependent on the computing resources provided.

2.2 Key Challenges in Supporting Search and Rescue Operations in Cloud Computing Environments

Meeting the requirements of SAR operations outlined in Section 2.1 is hard due to the inherent complexity of configuring enterprise DRE pub/sub middleware based on the computing resources the cloud provides. These resources are not known *a priori* and yet the QoS of the system is affected by the specific resources provided. The remainder of this section describes four challenges that ADAMANT addresses to support the communication requirements of the SAR operations presented above.

Challenge 1: Configuring for data timeliness and reliability. SAR operations must receive sufficient data reliability and timeliness so that multiple data streams can be fused appropriately. For instance, the SAR operation example described above shows how data streams (such as infrared scan and video streams) can be exploited by multiple applications simultaneously in a datacenter. The top half of Figure 2 shows how security monitoring and structural damage applications can use video stream data to detect looting and unsafe buildings, respectively. The bottom half of Figure 2 shows how fire detection applications and power grid assessment applications can use infrared scans to detect fires and working HVAC systems, respectively.

Likewise, the SAR systems must be configured to best use the computing and networking resources from the cloud to address data timeliness and reliability. These systems must therefore (1) use transport protocols that provide both reliability and timeliness and (2) know how these protocols behave in different computing and networking environments. Sections 3.1 and 4.1 describe how ADAMANT addresses this challenge by utilizing composite QoS metrics to measure both timeliness and reliability and incorporating transport protocols that configure the datacenter's pub/sub middleware to balance reliability and low latency.

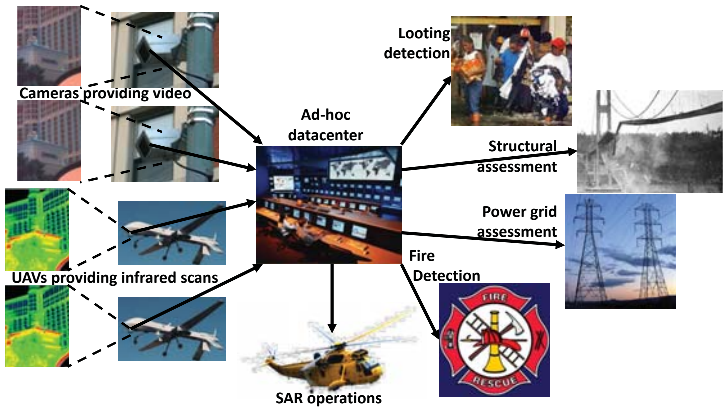


Fig. 2. Uses of Infrared Scans & Video Streams during Disaster Recovery

Challenge 2: Timely configuration. Due to timeliness concerns of DRE systems such as SAR systems, the *ad hoc* datacenter used for SAR operations must be configured in a timely manner based on the computing and networking resources provided by the cloud. If the datacenter cannot be configured quickly, invaluable time will be lost leading to survivors not being saved and critical infrastructure (such as dams and power plants) not being safeguarded from further damage. During a regional or national emergency any wasted time can mean the difference between life and death for survivors and the salvaging or destruction of key regional utilities.

Moreover, applications and systems used during one disaster can be leveraged for other disasters. Available computing and networking resources differ from one set of disaster recovery operations to another. Depending on the available cloud resources, therefore, the configuration times of *ad hoc* datacenters for SAR operations, for example, must be bounded and fast to ensure appropriate responsiveness. Determining appropriate configurations must also provide predictable response to ensure rapid and dependable response times across different computing and networking resources. Sections 3.2 and 4.4 describe how ADAMANT addresses this challenge by utilizing an artificial neural network machine learning tool to autonomically configure the datacenter's pub/sub middleware quickly and predictably.

Challenge 3: Accuracy of configurations. Since data timeliness and reliability is related to the computing resources available and the configuration of the datacenter supporting the SAR operations in a cloud as noted in Challenge 1, configuring the datacenter must be done in an accurate manner. If the datacenter is incorrectly configured then the timeliness and reliability of the data (*e.g.*, the UAV scans and camera video used to detect survivors) will not be optimal for the given computing resources. For critical operations during disasters, such as rescuing survivors, the supporting SAR system must utilize the available resources to their fullest extent. Sections 3.2 and 4.4 describe how ADAMANT addresses this challenge by using the artificial neural network machine learning tool to configure the datacenter's pub/sub middleware accurately.

Challenge 4: Reducing development complexity. Regional and local disasters occur in many places and at many different times. The functionality of applications used during one disaster may also be needed for other disasters. A system that is developed for one particular disaster in a particular operating environment, however, might not work well for a different disaster in a different operating environment. SAR operations could unexpectedly fail at a time when they are needed most due to differences in computing and networking resources available. Systems therefore must be developed and configured readily between the different operating environments presented by cloud computing to leverage the systems across a wide range of disaster scenarios. Section 3.2 describes how ADAMANT addresses this challenge by using an artificial neural network machine learning tool to manage mapping the computing and network resources and application parameters (*e.g.*, data sending rate, number of data receivers) to the appropriate transport protocol to use.

3 Overview of ADAMANT

This section presents an overview of the *ADAptive Middleware And Network Transports* (ADAMANT) platform, which is QoS-enabled pub/sub middleware that integrates and enhances the *Adaptive Network Transports* (ANT) framework to support multiple transport protocols and the *Artificial Neural Network* (ANN) machine learning technology to select appropriate transport protocols in a timely and reliable manner. ADAMANT extends our prior work [10,11] by empirically evaluating (1) the QoS delivered by DDS pub/sub middleware with respect to differences in computing and networking resources provided by cloud environments and (2) the accuracy and timeliness of ANN-based machine learning tools in determining appropriate middleware configurations.

Figure 3 shows how ADAMANT works in a cloud environment (*e.g.*, the *ad-hoc* SAR datacenter) to deploy cloud resources. Since ADAMANT configures itself based on the resources in a cloud, it must determine those resources autonomically when the cloud environment makes them available. ADAMANT queries the environment for hardware and networking resources using OS utilities.

For example, on Linux ADAMANT accesses the `/proc/cpuinfo` file to gather CPU information and executes the `ethtool` program to query network characteristics. ADAMANT combines this hardware information with other relevant application properties (*e.g.*, number of receivers and data sending rate) and sends it as input to the ANN, which determines the appropriate protocol in a timely manner and passes this information to ANT. ANT then configures the DDS middleware to use the appropriate transport protocol. The remainder of this section describes the structure and functionality of ADAMANT.

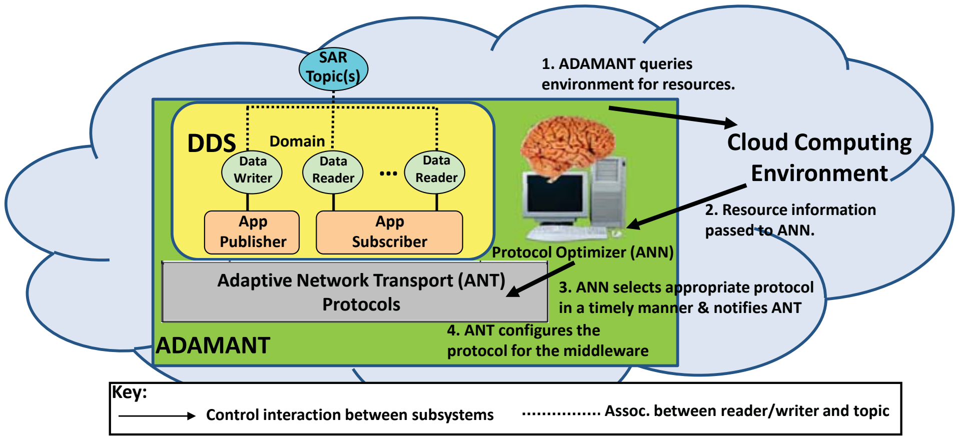

SAR Topic(s)
1. ADAMANT queries environment for resources.
Cloud Computing Environment
DDS
Domain
Data Writer
Data Reader
...
Data Reader
App Publisher
App Subscriber
2. Resource information passed to ANN.
Protocol Optimizer (ANN)
Adaptive Network Transport (ANT) Protocols
3. ANN selects appropriate protocol in a timely manner & notifies ANT
4. ANT configures the protocol for the middleware
ADAMANT
Key:
Control interaction between subsystems
Assoc. between reader/writer and topic

Fig. 3. ADAMANT Architecture and Control Flow

3.1 Adaptive Network Transports (ANT) Framework

The ANT framework supports various transport protocol properties, including multicast, packet tracking, NAK-based reliability, ACK-based reliability, flow control, group membership, and membership fault detection. These properties can be configured at startup to achieve greater flexibility and support configuration adaptation.

The ANT framework originally was derived from the Ricochet [3] transport protocol, which uses a bi-modal multicast protocol and a novel type of forward error correction (FEC) called lateral error correction (LEC) to provide QoS and scalability properties. Ricochet supports (1) time-critical multicast for high data rates with strong probabilistic delivery guarantees and (2) low-latency error detection along with low-latency error recovery. We included ANT's Ricochet protocol and ANT's NAKcast protocol, which is a NAK-based multicast protocol supporting a timeout parameter for when to send NAKs to the sender, with the evaluations done in this paper. These protocols have been selected due to their support for balancing reliability and low latency [10].

The Ricochet protocol has two tunable parameters. The *R* parameter determines the number of packets a receiver should receive before it sends out a repair packet to other receivers. The *C* parameter determines the number of receivers that will be sent a repair packet from any single receiver. These two parameters affect the timeliness, reliability, and jitter of the data received as shown in Section 4.3. ANT helps address Challenge 1 in Section 2.2 by supporting transport protocols that balance reliability and low latency.

3.2 Artificial Neural Network Tools to Determine Middleware Configurations

Several machine learning approaches can be used to configure middleware autonomically in a cloud computing environment. We selected ANN technology [11] due to its (1) fast and predictable performance, (2) accuracy for environments known *a priori* (*i.e.*, used for ANN training) *and* unknown until runtime (*i.e.*, not used for ANN training), and (3) low accidental development complexity. In particular, we chose the *Fast Artificial Neural Network* (FANN)(`leenissen.dk/fann`) implementation due to its configurability, documentation, ease of use, and open-source code. Section 4.4

shows the accuracy and timeliness of a neural network trained and tested using the data collected from the experiments described in Section 4.3. In particular, neural networks provide 100% accuracy for environments known *a priori*, high accuracy for environments unknown until runtime, and the low latency, constant time-complexity required for DRE systems such as SAR operations.

The use of an ANN helps address Challenges 2 and 3 in Section 2.2 by providing accurate, fast, and predictable guidance for determining an appropriate ADAMANT configuration for a given cloud computing environment. An ANN also helps address Challenge 4 in Section 2.2 by autonomically managing the mappings from the computing and network resources available and the application parameters (*e.g.*, data sending rate, number of data receivers) to the appropriate transport protocols. An ANN thus reduces the development complexity for configuring the pub/sub middleware appropriately as compared to manual adaptation approaches (*e.g.*, implementing switch statements), which are tedious and error-prone [13].

4 Experimental Results

The section presents the results of experiments we conducted to empirically evaluate (1) the effect of computing and networking resources on the QoS provided by ADAMANT as measured by the composite QoS metrics defined in Section 4.1 and (2) the timeliness and accuracy of an ANN in determining an appropriate ADAMANT configuration given a particular cloud computing environment. The experiments include ADAMANT with multiple aspects of the operating environment varied, (*e.g.*, CPU speed, network bandwidth, DDS implementation, percent data loss in the network) along with multiple aspects of the application being varied as would be expected with SAR operations (*e.g.*, number of receivers, sending rate of the data).

4.1 Composite QoS Metrics for Reliability and Timeliness

Our prior work [10, 11] on QoS-enabled pub/sub middleware performance for non-cloud environments indicated that some transport protocols provide better reliability (as measured by the number of network packets received divided by the number sent) and latency for certain environments while other protocols are better for other environments. We therefore developed several *composite QoS metrics* to evaluate multiple QoS aspects simultaneously, thereby providing a uniform and objective evaluation of ADAMANT in cloud computing environments. Our composite QoS metrics focus on reliability and average latency, including the QoS aspects of (1) *jitter* (*i.e.*, standard deviation of the latency of network packets), (2) *burstiness* (*i.e.*, the standard deviation of average bandwidth usage per second of time), and (3) network bandwidth usage.

Two of the composite QoS metrics we defined are *ReLate2* and *ReLate2Jit*. ReLate2 is the product of the average data packet latency and the percent loss + 1 (to account for 0% loss) which implies an order of magnitude increase for 9% loss. This adjustment is relevant for multimedia data in our SAR example based on previous research, *e.g.*, if average packet latency is 1,000 μs and the percent loss is 0 (*i.e.*, no packets lost) then the ReLate2 value is 1,000. Having 9% and 19% loss with the same average latency produces the ReLate2 values of 10,000 and 20,000 respectively. ReLate2Jit is a product of the ReLate2 value and the jitter of the data packets to quantify reliability, average latency, and jitter.

We apply these metrics below to QoS-enabled DDS pub/sub middleware using various transport protocols supported by ANT to train the ANN. The ANN is trained with an understanding of how integration of middleware with each protocol affects the QoS properties of reliability and latency given the variability of computing and networking resources of a cloud environment.

4.2 Experimental Setup

We conducted our experiments using the Emulab network testbed, which provides on-demand computing platforms and network resources that can be easily configured with the desired OS, network topology, and network traffic shaping. We used Emulab due to its (1) support for multiple types of computing platforms, (2) numbers of computing platforms, and (3) support for multiple network bandwidths. The flexibility of Emulab presents a representative testbed to train and test ADAMANT's configurability support for cloud computing environments.

As described in Section 2, we are concerned with the distribution of data for SAR datacenters, where network packets are typically dropped at end hosts [4]. The ADAMANT software for the receiving data readers supports programmatically dropping random data packets. We modified ADAMANT to drop packets based on the loss percentage specified for the experiment.

Our experiments were configured with the following traffic generation models using version 1.2.1 of OpenDDS and version 3.4.2 of OpenSplice. One DDS data writer sent out data, a variable number of DDS data readers received the data. The data writer and each data reader ran on its own computing platform and the data writer sent 12 bytes of data 20,000 times at a specified sending rate. To account for experiment variations we ran 5 experiments for each configuration, *e.g.*, 3 receiving data writers, 50 Hz sending rate, 2% end host packet loss, pc3000 computing platform, and 1Gb network bandwidth.

We configured ADAMANT with Ricochet and NAKcast to determine how well it performs using these protocols. We modified NAKcast's timeout value as well as Ricochet's *R* and *C* parameters as described in Section 3.1. Table 1 outlines the points of variability provided by the cloud computing environment. We include the DDS implementation in this table since some cloud computing environments provide hardware and software resources. We include network loss in the table since the network characteristics in cloud computing can be specified in an end-user license agreement, which identifies the services that the cloud computing environment will provide and that consumers accept. The middleware for the SAR operations can then be configured appropriately using this information.

Table 1. Environment Variables

Point of Variability	Values
Machine type	pc850, pc3000
Network bandwidth	1Gb, 100Mb, 10Mb
DDS Implementation	OpenDDS, OpenSplice
Percent end-host network loss	1 to 5 %

Table 2. Application Variables

Point of Variability	Values
Number of receiving data readers	3 - 15
Frequency of sending data	10 Hz, 25 Hz, 50 Hz, 100 Hz

Table 2 outlines the points of variability due to the SAR operations. In particular, we varied the number of data receivers since only a few SAR applications might be interested in one data stream (*e.g.*, for a localized area with fine-grained searching) while many applications might be interested in a different data stream (*e.g.*, for a broader area with coarse-grained searching). Likewise, the sending rate might be high for SAR operations that need high-resolution imaging for detailed searching while a lower sending rate is sufficient for SAR operations where lower resolution imaging is sufficient for more generalized searching.

For computing resources we used Emulab's pc850 and pc3000 hardware platforms. The pc850 platform includes an 850 MHz 32-bit Pentium III processor with 256 MB of RAM. The pc3000 platform includes a 3 GHz 64-bit Xeon processor with 2 GB of RAM. We used the Fedora Core 6 operating system with real-time extensions on these hardware platforms to collect high resolution timings. The nodes were all configured in a LAN configuration indicative of a datacenter.

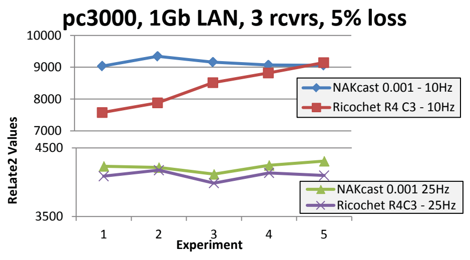
4.3 Evaluating How Cloud Computing Resources Affect QoS

Below we analyze the results from experiments involving different cloud computing environments. We show experimental data where the selection of ADAMANT's transport protocol to support QoS differs based on the cloud computing environment. Information in this section addresses Challenge 1 in Section 2.2 by characterizing the performance of the transport protocols for various cloud computing environments.

Figures 4 and 5 show the results of experiments where we held constant the number of receivers (3), the percent loss (5%), and the DDS middleware (OpenSplice). We varied the computing platform and the network bandwidth using the pc850 and pc3000 platforms, and 100Mb and 1Gb LANs, respectively. We ran the experiments using NAKcast with a NAK timeout setting of 50ms, 25ms, 10ms, and 1ms, and Ricochet with R=4, C=3 and R=8, C=3. We only include NAKcast with a timeout of 1ms and Ricochet R=4 C=3 since these were the only protocols that produced the best (*i.e.*, lowest) ReLate2 values for these operating environments. Likewise, we ran the ADAMANT experiments with sending rates of 10Hz, 25Hz, 50Hz, and 100Hz but only show results for 10Hz and 25Hz since these highlight different protocols that produce the lowest ReLate2 value.

Figure 4 shows two cases where the Ricochet protocol with R = 4 and C = 3 produces the best (*i.e.*, lowest) ReLate2 values for sending rates of both 10Hz and 25Hz when using the pc3000 computing platform and the 1Gb network. Conversely, Figure 5 shows how the NAKcast protocol with a NAK timeout set to 1 ms produces the best (*i.e.*, lowest) ReLate2 values for the same sending rates of 10Hz and 25Hz when using the pc850 computing platform and the 100Mb network. These figures show that by changing only the CPU speed, amount of RAM, and network bandwidth, different protocols produce a better ReLate2 value and therefore better support the QoS properties of reliability and average latency. The SAR datacenter pub/sub middleware should therefore be configured differently depending on the computing and networking resources that a cloud computing environment provides. No single protocol performs best in all cases based on the computing and networking resources.

We decompose the ReLate2 metric into its constituent parts of reliability and average packet latency to gain a better understanding of how changes in hardware can



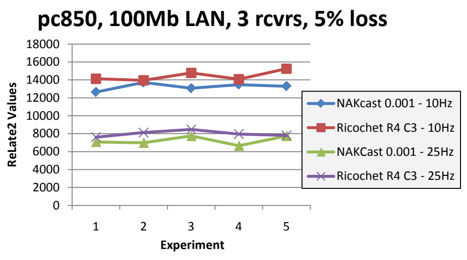
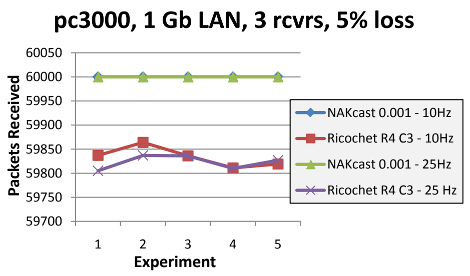


Fig. 4. ReLate2: pc3000, 1Gb LAN, 3 receivers, 5% loss, 10 & 25Hz

Fig. 5. ReLate2: pc850, 100Mb LAN, 3 receivers, 5% loss, 10 & 25Hz

affect the QoS properties relevant to the ReLate2 metric. Figures 6 and 7 show the reliability of the NAKcast 0.001 and Ricochet R4 C3 protocols. The reliability of the protocols is relatively unaffected by differences in hardware and network resources as would be expected. The percent network loss is held constant for these experiments and the differences in hardware are not expected to affect how many packets are delivered reliably.

Figures 8 and 9, show that differences in computing speed and networking bandwidth have an effect on the average latency of packet arrival. In particular, there is a wider gap in the average latency times between the NAKcast and the Ricochet protocol when faster computing and networking resources are used.



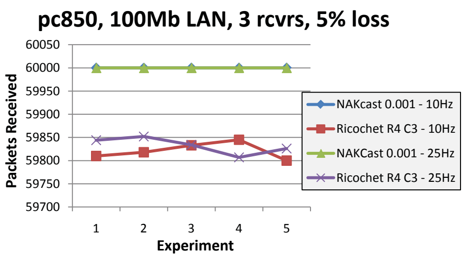


Fig. 6. Reliability: pc3000, 1Gb LAN, 3 receivers, 5% loss, 10 & 25Hz

Fig. 7. Reliability: pc850, 100Mb LAN, 3 receivers, 5% loss, 10 & 25Hz

Since protocol reliability in these experiments is virtually constant, the difference in NAKcast performing better in one environment and Ricochet performing better in another stems from differences in average latency. With faster hardware and networks, Ricochet's average latency can overcome its lower reliability to perform better when reliability and average latency are both considered. Note that the graphs for the individual QoS property of average latency consistently show Ricochet performing better, while the graphs consistently show NAKcast performing better for reliability. Only when the QoS properties are combined in the ReLate2 metric is there a distinction between the appropriate protocol based on the hardware resources.

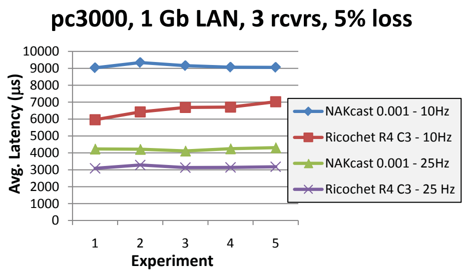


Fig. 8. Latency: pc3000, 1Gb LAN, 3 receivers, 5% loss, 10 & 25Hz

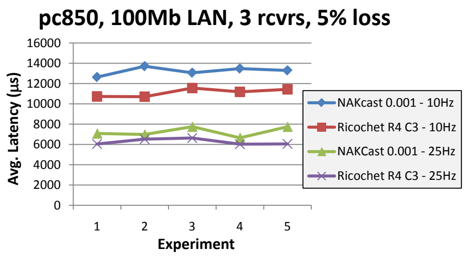


Fig. 9. Latency: pc850, 100Mb LAN, 3 receivers, 5% loss, 10 & 25Hz

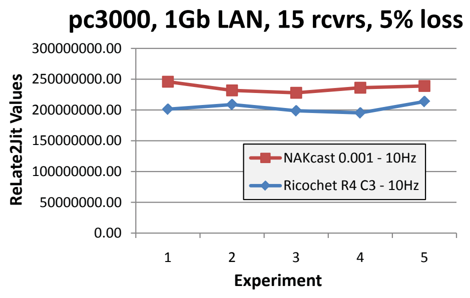


Fig. 10. ReLate2Jit: pc3000, 1Gb LAN, 15 receivers, 5% loss, 10Hz

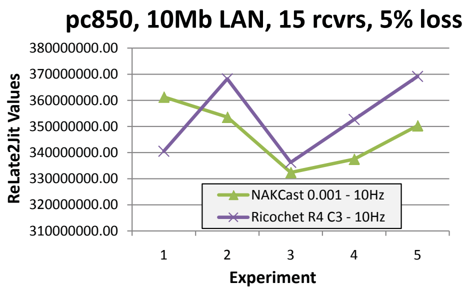


Fig. 11. ReLate2Jit: pc850, 100Mb LAN, 15 receivers, 5% loss, 10Hz

Figures 10 and 11 show that the differences in hardware resources affect the protocol to choose based on the ReLate2Jit metric which measures reliability, average packet latency, and the standard deviation of packet latency (*i.e.*, jitter). The number of receivers is 15, the network percent loss is 5%, and the DDS middleware is OpenSplice. We again varied the computing platform and the network bandwidth using the pc850 and pc3000 platforms and 100Mb and 1Gb LANs, respectively. The figures only include data for NAKcast with a 1 ms timeout and Ricochet R=4 C=3 both with a 10Hz sending rate since, with this rate, the environment has triggered the selection of different protocols based on the ReLate2Jit values.

Figure 10 shows Ricochet R=4 C=3 to consistently have the best (*i.e.*, lowest) ReLate2Jit values when using pc3000 computers and a 1Gb network. Figure 11 shows NAKcast with a timeout of 1 ms as most of the time (4 out of 5 experiment runs) having the better ReLate2Jit value. We decompose the ReLate2Jit values to have a better understanding of the differences.

Figures 12 and 13 show the average latency broken out from the ReLate2Jit values above. These figures show that Ricochet R=4 C=3 consistently has the lowest average latencies regardless of the computing and network resources. Likewise, Figures 14 and 15 show that Ricochet R=4 C=3 consistently has lower jitter values across the different hardware. Figures 16 and 17 again show that NAKcast provides high reliability, while Ricochet provides less reliability.

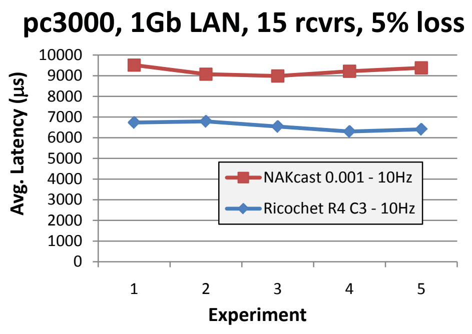


Fig. 12. Latency: pc3000, 1Gb LAN, 15 receivers, 5% loss, 10Hz

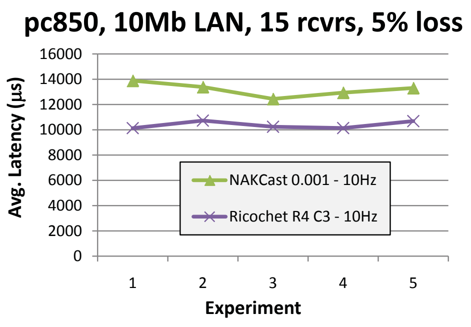


Fig. 13. Latency: pc850, 100Mb LAN, 15 receivers, 5% loss, 10Hz

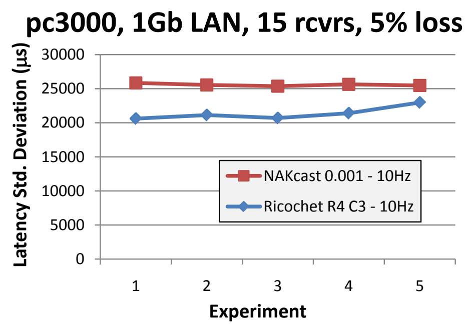


Fig. 14. Jitter: pc3000, 1Gb LAN, 15 receivers, 5% loss, 10Hz

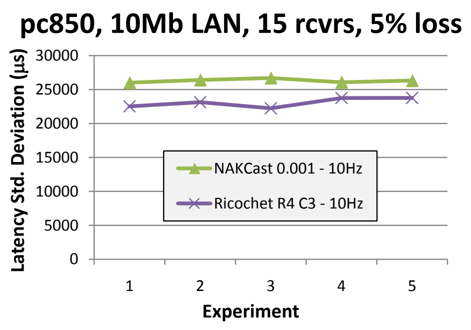


Fig. 15. Jitter: pc850, 100Mb LAN, 15 receivers, 5% loss, 10Hz

All figures for individual QoS properties (*i.e.*, Figures 12 through 17) related to the ReLate2Jit measurements in Figures 10 and 11 show fairly consistent results across differing hardware. When these QoS properties are combined into a single, objective value,

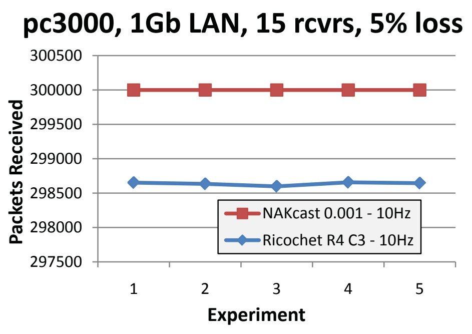


Fig. 16. Reliability: pc3000, 1Gb LAN, 15 receivers, 5% loss, 10Hz

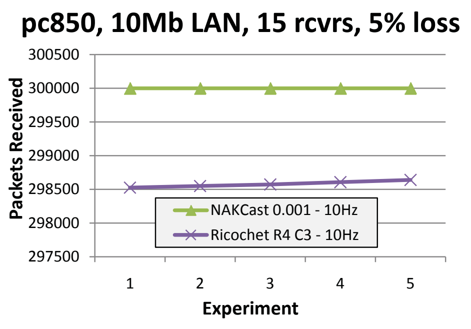


Fig. 17. Reliability: pc850, 100Mb LAN, 15 receivers, 5% loss, 10Hz

however, we are better able to distinguish one protocol from another thus highlighting the advantages to using composite metrics.

4.4 Determining Appropriate Protocol with Artificial Neural Networks

Evaluating the Accuracy of Artificial Neural Networks The first step to using an ANN is to train it on a set of data. We provided the ANN with 394 inputs where an input consists of data values outlined in Tables 1 and 2 plus the composite metric of interest (*i.e.*, ReLate2 or ReLate2Jit). We also provided the expected output, *i.e.*, the transport protocol that provided the best composite QoS value for ReLate2 or ReLate2Jit.

An example of one of the 394 inputs is the following: 3 data receivers, 1% network loss, 25Hz sending rate, pc3000 computers, 1Gb network, OpenSplice DDS implementation, and ReLate2Jit as the metric of interest. Based on our experiments, the corresponding output would be the NAKcast protocol with a NAK timeout of 1 ms. All the 394 inputs are taken from experiments that we ran as outlined in Section 4.

FANN offers extensive configurability for the neural network, including the number of *hidden nodes* that connect inputs with outputs. We ran training experiments with the ANN using different numbers of hidden nodes to determine the most accurate ANN. For a given number of hidden nodes we trained the ANN five times. The weights of the ANN determine how strong connections are between nodes. The weights are initialized randomly and the initial values effect how well the ANN learns.

Figures 18 and 19 show the ANN accuracies for environment configurations that were known *a priori* and environments that were unknown until runtime respectively. The ANN was configured with different numbers of hidden nodes and a stopping error

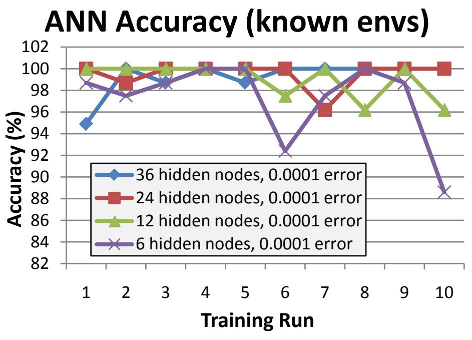


Fig. 18. ANN Accuracy for environments known *a priori*

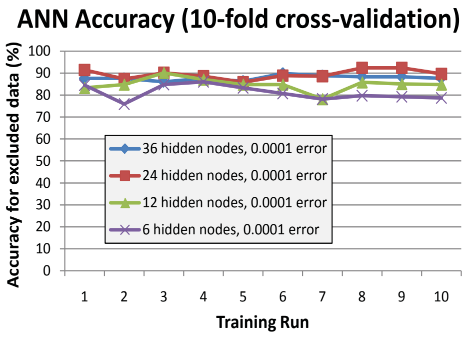


Fig. 19. ANN Accuracy for environments unknown until runtime

of 0.0001 (*i.e.*, an indication to the ANN that it should keep iterating over the data until the error between what the ANN generates and the correct response is 0.0001). Additional experiments were conducted with higher stopping errors (*e.g.*, 0.01), but lower stopping errors consistently produced more accurate classifications as expected.

Accuracy for environments known *a priori* was determined by querying the ANN with the data on which it was trained. Since we know the answer we gave to the ANN when it was trained we check to make sure the answer matches the ANN's response. Over the 10 training runs shown in Figure 18 the highest number of 100% accurate classifications was generated using 24 hidden nodes (*i.e.*, 8).

Accuracy for environments unknown until runtime is determined by splitting out the 394 environment configurations into mutually exclusive training and testing data sets. This approach is referred to as *n*-fold cross-validation where *n* is the number of mutually exclusive training and testing data sets [15]. The value of *n* also determines the amount of data excluded from training and used only for testing.

We used 10-fold cross-validation which indicates 10 sets of training and testing data where for each fold the training and testing data are mutually exclusive and the training data excludes 1/10 of the total data. As shown in Figure 19 the ANN with 24 hidden nodes and a stopping error of 0.0001 produced the highest average accuracy of 89.49%. We conducted our timings tests using this ANN since it provided the highest number of 100% accurate classifications for environments known *a priori* and the highest accuracy for environments unknown until runtime.

Evaluating the Timeliness of Artificial Neural Networks As described in Challenge 2 in Section 2.2, the datacenter for the SAR operations needs to have timely configuration adjustments. We now provide timing information based on the ANN's responsiveness when queried for an optimal transport protocol. Timeliness was determined by querying the ANN with all 394 inputs on which it was trained. A high resolution timestamp was taken right before and right after each call made to the ANN.

Figures 20 and 21 show the average response times and standard deviation of the response times, respectively, for 5 separate experiments where for each experiment we query the ANN for each of the 394 inputs. The figures show that the ANN provides

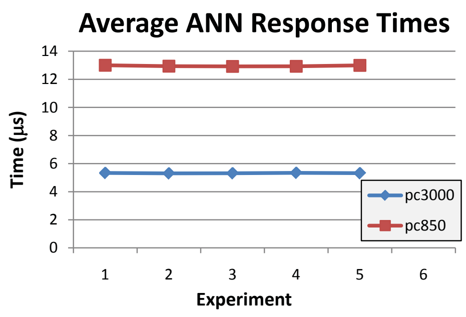


Fig. 20. ANN average response times

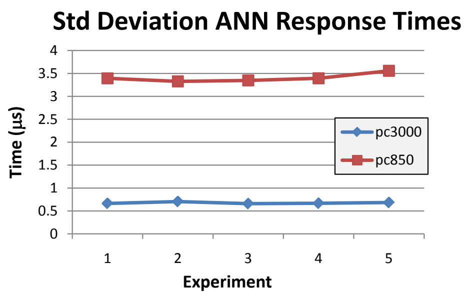


Fig. 21. Standard deviation for ANN response times

timely and consistent responses. As expected, the response times on the pc850 platform are slower than for the pc3000.

Inspection of the ANN source code confirmed experimental results that the ANN provides fast and predictable responses for both environments known *a priori* and unknown until runtime. When queried for a response with a given set of input values, the ANN loops through all connections between input nodes, hidden nodes, and output nodes. The number of nodes and number of connections between them were determined previously when the ANN was trained. With a high level of accuracy, predictability, and minimal development complexity, ANNs provide a suitable technique for determining ADAMANT configurations.

5 Related Work

This section compares our work on ADAMANT with related R&D efforts.

Support for adaptive middleware. Ostermann *et al.* [19] present the ASKALON middleware for cloud environments that is based on middleware for grid workflow application development but enhanced to leverage clouds. ASKALON provides an infrastructure that allows the execution of workflows on conventional grid resources but that can adapt on-demand to supplement these resources with additional cloud resources as needed. In contrast to ADAMANT, however, ASKALON does not address the adaptive configurability needs of enterprise DRE systems in elastic clouds.

Gridkit [8] is a middleware framework that supports reconfigurability of applications dependent upon the condition of the environment and the functionality of registered components. Gridkit focuses on grid applications which are highly heterogeneous in nature. In contrast to ADAMANT, however, Gridkit does not address timely adaptation, nor does it focus on discovering and leveraging the elastic provisioning of cloud resources.

David and Ledoux have developed SAFRAN [7] to enable applications to become context-aware themselves so that they can adapt to their contexts. SAFRAN provides reactive adaptation policy infrastructure for components using an aspect-oriented approach. The SAFRAN component framework, however, provides only development support for maintaining specified QoS. The adaptive policies and component implementation are the responsibility of the application developer. Moreover, SAFRAN does not address timely configuration of components across the elastic resources of cloud computing. In contrast, ADAMANT provides a middleware implementation that adapts to the cloud resources presented to it.

Machine learning in support of autonomic adaptation. Vienne and Sourrouille [23] 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. In contrast to ADAMANT, however, DCBL focuses only on a single computer, rather than scalable DRE pub/sub systems. Moreover, reinforcement learning used by DCBL can have non-constant and even unbounded time complexities unlike ADAMANT which provides fast and predictable decision making.

RAC [5] uses reinforcement learning for the configuration of Web services. RAC autonomically configures services via performance parameter settings to change the services' workload and also to change the virtual machine configurations. Due to RAC's use of reinforcement learning, its determination of an appropriate response is unbounded due to online exploration of the solution space and modification of decisions while the system is running. In contrast, ADAMANT uses ANN machine learning to provide fast, predictable complexity decision making.

Tock *et al* [21] utilize machine learning for data dissemination in their work on Multicast Mapping (MCM). MCM hierarchically clusters data flows so that multiple topics map to a single session and multiple sessions are mapped to a single reliable multicast group. MCM manages the scarce availability of multicast addresses in large-scale systems and uses machine learning for adaptation as user interest and message rates change during the day. MCM is designed only to address the scarce resource of IP multicast addresses in large-scale systems, however, rather than timely adaptation based on available resources as done with ADAMANT.

Infrastructure for autonomic computing. Grace *et al.* [9] 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 provide an adaptation implementation as ADAMANT does.

Valetto *et al.* [22] have 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 provide an implementation unlike ADAMANT.

6 Concluding Remarks

Developers of systems which use DRE pub/sub middleware face several configuration challenges for cloud computing environments. To address these challenges, this paper presented the structure, functionality, and performance of *ADAptive Middleware And Network Transports* (ADAMANT). ADAMANT is pub/sub middleware that uses supervised machine learning to autonomously configure cloud environments with transport protocols that enhance the predictability of enterprise DRE systems.

The results in this paper empirically showed how computing hardware environments affect QoS for these systems and how ADAMANT configures the system based on the computing resources provided at startup in a fast and accurate manner while reducing development complexity over manual adaptation approaches. We selected ANNs to determine appropriate configurations since they provide (1) the highest level of accuracy possible for known environments, (2) better than random or default guidance for environments not known until runtime, and (3) the timing complexity required for DRE systems. The following is a summary of lessons learned from our experience evaluating ADAMANT's configuration performance in various cloud environments:

• **Computing resources affect which QoS mechanism provides the best support.** Differences in CPU speed and network bandwidth affect the choice of the most appropriate QoS mechanism. For certain computing environments, one transport protocol provided the best QoS; for other environments a different transport protocol was best. We leveraged this information to select the appropriate protocol for given computing resources. We are investigating other machine learning techniques that provide timeliness and high accuracy to compare with ANNs.

• **Fast, predictable configuration for DRE pub/sub systems can support dynamic autonomic adaptation.** ADAMANT can accurately and quickly configure a

DRE pub/sub system at startup in cloud environments. Some systems, however, run in operating environments that change during system operation. The ADAMANT results have motivated future work on autonomic adaptation of middleware and transport protocols to support QoS in turbulent environments. Fast, predictable configuration can be used to adapt transport protocols to support QoS while the system is monitoring the environment. When the system detects environmental changes (*e.g.*, increase in number of receivers or increase in sending rate), supervised machine learning can provide guidance to support QoS for the new configuration.

• **Composite QoS metrics should be decomposed to better understand behavior of the system.** A change in the values from composite QoS metrics can be caused by changes in any of the individual QoS concerns or any combination of the concerns. The composite QoS metrics provide a higher level of abstraction for evaluating QoS and, as with any abstraction, details which might be important can be obfuscated. The composite QoS metrics we use are fairly easy to decompose as shown by Figures 4–9 in Section 4.3, although the more QoS properties that are composed the more decomposition is needed, which is hard, tedious, and time-consuming.

• **Exploring a configuration space for trade-offs requires a disciplined approach with analysis to guide the exploration.** Depending on the number of dimensions involved in the search space there can be many configurations to explore. In this paper we had multiple variables, *e.g.*, CPU speed, RAM, network bandwidth, update rate, % packet loss, number of data readers, NAKcast's timeout value, and Ricochet's *R* value. Since the number of potential experiments was large, we found it helpful to make coarse-grained adjustments for initial experiments. We would then analyze the results to guide areas of refinement to find trade-offs between transport protocols.

All ADAMANT source code and documentation and its supporting ANN tools is available as open-source at `www.dre.vanderbilt.edu/~jhoffert/ADAMANT`.

References

1. Global Information Grid. The National Security Agency, www.nsa.gov/ia/industry/gig.cfm?MenuID=10.3.2.2
2. Net-Centric Enterprise Services. Defense Information Systems Agency, http://www.disa.mil/nces/
3. Balakrishnan, M., Birman, K., Phanishayee, A., Pleisch, S.: Ricochet: Lateral Error Correction for Time-Critical Multicast. In: NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation. Boston, MA (2007)
4. Balakrishnan, M., Pleisch, S., Birman, K.: Slingshot: Time-Critical Multicast for Clustered Applications. In: The IEEE Conference on Network Computing and Applications (2005)
5. Bu, X., Rao, J., Xu, C.Z.: A Reinforcement Learning Approach to Online Web Systems Auto-configuration. In: The 29th IEEE International Conference on Distributed Computing Systems. pp. 2–11. IEEE Computer Society, Washington, DC, USA (2009)
6. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud Computing and Emerging IT platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Future Generation Computer Systems 25(6), 599 – 616 (2009)
7. David, P.C., Ledoux, T.: Software Composition, chap. An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components, pp. 82–97. Springer LNCS, Berlin / Heidelberg (2006)

8. Grace, P., Coulson, G., Blair, G.S., Porter, B.: Deep Middleware for the Divergent Grid. In: Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware. pp. 334–353. Springer-Verlag New York, Inc., New York, NY, USA (2005)
9. Grace, P., Coulson, G., Blair, G.S., Porter, B.: A Distributed Architecture Meta-model for Self-managed Middleware. In: Proceedings of the 5th Workshop on Adaptive and Reflective Middleware (ARM '06). p. 3. ACM, New York, NY, USA (2006)
10. Hoffert, J., Gokhale, A., Schmidt, D.: Evaluating Transport Protocols for Real-time Event Stream Processing Middleware and Applications. In: Proceedings of the 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA '09). Vilamoura, Algarve-Portugal (Nov 2009)
11. Hoffert, J., Schmidt, D.C., Gokhale, A.: Adapting and Evaluating Distributed Real-time and Embedded Systems in Dynamic Environments. In: Proceedings of the 1st International Workshop on Data Dissemination for Large scale Complex Critical Infrastructures (DD4LCCI 2010). Valencia, Spain (Apr 2010)
12. Ibnkahla, M., Rahman, Q., Sulyman, A., Al-Asady, H., Yuan, J., Safwat, A.: High-speed Satellite Mobile Communications: Technologies and Challenges. Proceedings of the IEEE 92(2), 312 – 339 (Feb 2004)
13. Kavimandan, A., Narayanan, A., Gokhale, A., Karsai, G.: Evaluating the Correctness and Effectiveness of a Middleware QoS Configuration Process in Distributed Real-time and Embedded Systems. In: Proceedings of the 11^{th} IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2008). pp. 100–107. Orlando, FL, USA (May 2008)
14. Lin, Q., Neo, H.K., Zhang, L., Huang, G., Gay, R.: Grid-based Large-scale Web3D Collaborative Virtual Environment. In: Web3D '07: Proceedings of the Twelfth International Conference on 3D Web Technology. pp. 123–132. ACM, New York, NY, USA (2007)
15. Liu, Y.: Create Stable Neural Networks by Cross-Validation. In: IJCNN '06: Proceedings of the International Joint Conference on Neural Networks. pp. 3925–3928 (2006)
16. Menth, M., Henjes, R.: Analysis of the Message Waiting Time for the FioranoMQ JMS Server. Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on pp. 1–1 (2006)
17. Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds. In: Proceedings of EuroSys 2010. pp. 237–250. Paris, France (Apr 2010)
18. Object Management Group: Data Distribution Service for Real-time Systems Specification, 1.2 edn. (Jan 2007)
19. Ostermann, S., Prodan, R., Fahringer, T.: Extending Grids with Cloud Resource Management for Scientific Computing. In: 10th IEEE/ACM International Conference on Grid Computing, 2009. pp. 42 –49 (13-15 2009)
20. Shankaran, N., Koutsoukos, X., Lu, C., Schmidt, D.C., Xue, Y.: Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems. Real-Time Systems 1(3), 237–282 (April 2008)
21. Tock, Y., Naaman, N., Harpaz, A., Gershinsky, G.: Hierarchical Clustering of Message Flows in a Multicast Data Dissemination System. In: Proceedings of Parallel and Distributed Computing and Systems (PDCS 2005) (Nov 2005)
22. Valetto, G., Goix, L.W., Delaire, G.: Towards Service Awareness and Autonomic Features in a SIP-Enabled Network. In: Autonomic Communication. pp. 202–213. Springer-Verlag, Berlin, Heidelberg (2006)
23. Vienne, P., Sourrouille, J.L.: A Middleware for Autonomic QoS Management Based on Learning. In: Proceedings of the 5th International Workshop on Software Engineering and Middleware. pp. 1–8. ACM, New York, NY, USA (2005)