Evaluating Supervised Machine Learning for Adapting Enterprise DRE Systems

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

***Abstract*— Several adaptation approaches, such as policy-based and reinforcement learning, have been devised to ensure end-to-end quality-of-service (QoS) for enterprise distributed systems in dynamic operating environments. Not all approaches are applicable for distributed real-time and embedded (DRE) systems, however, which have stringent accuracy, timeliness, and development complexity requirements. Supervised machine learning techniques, such as artificial neural networks (ANNs), are a promising approach to address time complexity concerns of adaptive enterprise DRE systems. Likewise, ANNs address the development complexity of adaptive DRE systems by ensuring that adaptations are appropriate for the operating environment. This paper empirically evaluates the accuracy and timeliness of the ANN machine learning technique for environments on which it has been trained. Our results show ANNs are highly accurate in determining correct adaptations and provide predictable time complexity, *e.g.*, with response times less than 6 μseconds.**

I. Introduction

Emerging trends and challenges. Enterprise distributed real-time and embedded (DRE) systems manage resources and data that are vital to the ongoing objectives of organizations or projects. Examples include shipboard computing environments, air traffic management systems, and recovery operations in the aftermath of regional or national disasters. These enterprise DRE systems often adjust the way they operate depending on their external environment. For example, search and rescue missions as part of disaster recovery operations can adjust the image resolution used to detect and track survivors depending on the resources available (*e.g.*, computing power, network bandwidth) [5].

Many enterprise DRE systems autonomically (1) monitor their environment and (2) modify their modes as the environment changes since manual adjustment is too slow and error prone. For example, a shift in network reliability can prompt quality-of-service (QoS)-enabled middleware, such as the OMG Data Distribution Service (DDS) (`www.omgwiki.org/dds`), to change mechanisms (such as the transport used to deliver data) since some transports provide better reliability than others in some environments. Likewise, cloud computing applications where elastically allocated resources (*e.g.*, CPU speeds and memory) cannot be characterized accurately *a priori* may need to adjust to available resources (such as using compression algorithms optimized for given CPU power and memory) at system startup. If adjustments take too long the mission(s) the system implements could be jeopardized.

One way to adapt enterprise DRE systems autonomically involves the use of *policy-based* approaches [1] that externalize and codify logic to determine the behavior of managed systems. Policy-based approaches provide deterministic response times to perform appropriate adjustments given changes in the environment and can be optimized to ensure efficient performance. The complexity of developing and maintaining policy-based approaches for enterprise DRE systems can be unacceptably high, however, since developers must determine which policies are applicable for certain environmental properties. Moreover, developers must manage how the policies interact to provide needed adjustments.

Machine learning techniques support algorithms that allow systems to adjust behavior based on empirical data, *e.g.*, inputs from the environment. These techniques can be used to support autonomic adaptation by learning appropriate adjustments to various operating environments. Unlike policy-based approaches, however, machine learning techniques automatically recognize complex sets of environment properties and make appropriate decisions accordingly.

Conventional machine learning techniques, such as decision trees and reinforcement learning, have been used to address autonomic adaptation for non-DRE systems [3]. These techniques are not well-suited for enterprise DRE systems, however, since they do not provide bounded times when determining adjustments [4]. Some techniques, such as reinforcement learning [6], explore the solution space until an appropriate solution is found, regardless of the elapsed time. Other techniques, such as decision trees, have time complexities that are dependent upon the specific data and cannot be determined *a priori*. Moreover, decision trees may contain branches that are much longer than others, which can make the determination of appropriate adaptations unpredictable—an undesirable quality in DRE systems.

Solution approach → Overfitted machine learning to guide QoS-enabled middleware adaptation. Machine learning uses guidance from past known environments to handle new and unknown environments. This generality sacrifices some accuracy, however, that would otherwise be provided for known environments. Machine learning techniques that are specialized for the environments they have seen—and on which they have been trained—are said to be *overfitted* [2], which reduces development complexity and makes the accuracy comparable to policy-based approaches.

This paper describes an overfitted machine learning approach we tailored to reduce the complexity of developing autonomically adaptive enterprise DRE systems. In particular, we are tuning an *artificial neural network* (ANN) [7] (which is a technique modeled on the interaction of neurons in the human brain) to retain as much information about specific environment configurations and adjustments as possible (*e.g.*, greatly increasing the number of connections between input environment characteristics and output adjustments typically

used in an ANN). Our *ADAptive Middleware And Network Transports* (ADAMANT) work presented in this paper integrates the ANN machine learning technique with the DDS QoS-enabled middleware to ensure accurate, timely, and predictable adaptation to operating environment changes, such as an increase in the data sending rate or number of data receivers.

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

To motivate the need for overfitting machine learning techniques, this section describes the challenges associated with search and rescue (SAR) operations. SAR operations are part of disaster recovery enterprise DRE systems which manage relief efforts in the aftermath of a disaster, such as a hurricane, earthquake, or tornado. SAR operations help locate and extract survivors in a large metropolitan area after a regional catastrophe. SAR operations 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 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.

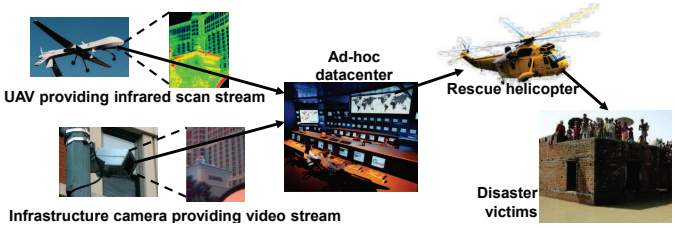

Fig. 1. **Search and Rescue Motivating Example**

III. Key Challenges of Enterprise DRE Systems

Below we summarize key challenges that arise when developing autonomic enterprise DRE systems, such as the SAR motivating example in Section II.

A. Challenge 1: Reduction of Development Complexity

Developing autonomic behavior can incur high complexity due to the number and type of relevant environmental conditions. For example, the number of data receivers can affect the optimal transport protocols and parameter settings used since some protocols provide adequate QoS for a small number of receivers whereas other protocols provide adequate QoS for a larger number of receivers. Codifying this knowledge requires developers to manually (1) determine the appropriate protocol for a given environment and (2) map this determination accurately into implementation artifacts (*e.g.*, source code). The manual management of mapping between environment and protocol is tedious and error-prone, which increases development complexity.

B. Challenge 2: Timely Adaptation to Dynamic Environments

Due to the dynamic environment inherent in enterprise DRE systems, application operations (such as image compression to reduce network traffic or disseminating data with both timeliness and reliability properties) must adjust in a bounded timely—ideally constant time—manner as the environment changes. Operations that cannot adjust quickly and in a bounded amount of time will fail to perform adequately when resources change, *e.g.*, if resources are lost or withdrawn—or demand for information increases—operations must be configured to accommodate these changes with appropriate responsiveness to maintain a minimum level of service. If resources increase or demand decreases, operations should adjust as quickly as possible to provide higher fidelity or more expansive coverage. Manual modification is often too slow and error-prone to maintain QoS.

C. Challenge 3: Accurate Adaptation to Dynamic Environments

Application operations in enterprise DRE systems must be able to adjust to changes in the environment accurately. As changes in enterprise DRE systems occur (*e.g.*, increases in networking capability, requests for data from additional senders and receivers, etc.), the system should take advantage of additional resources or provide access to additional data producers and consumers while maintaining or increasing QoS. For a given environment configuration, the enterprise DRE system must accurately implement adjustments that are appropriate to the operating environment.

IV. Solution Approach - Overfitting Machine Learning Techniques

Our solution approach overfits machine learning techniques to increase accuracy in determining appropriate adjustments, such as adjustments to transport protocols to support QoS in dynamic environments. This approach enables enterprise DRE systems to autonomically adjust to their environments. Moreover, we leverage techniques that provide the time complexity assurance needed for enterprise DRE systems.

Our overfitting approach tunes an ANN to retain a high degree of information about specific environment configurations and adjustments, *e.g.*, increasing the number of *hidden nodes* used in an ANN. Hidden nodes are the computational components that provide connections between the relevant properties of the operating environment (*e.g.*, CPU speed, network reliability) with the adjustments needed for those environments. As the ANN learns, it strengthens or weakens the connections between inputs, hidden nodes, and outputs to provide appropriate adjustments. Increasing the number of hidden nodes increases the level of detail that the ANN maintains. Our approach resolves the challenges presented in Section II as follows:

• Overfitted machine learning techniques address Challenge 1 in Section III-A by decreasing the development complexity involved with codifying adjustments for multiple configurations of operating environments. Policy-based approaches for autonomic adaptation place the complexity burden on application developers, who must manually maintain operating environment configurations, appropriate adjustments needed, and the mapping between the configurations and the adjustments. Moreover, developers must accurately codify this mapping in their implementations. Overfitted machine learning techniques relieve developers of this burden since they manage the complexity via training to react appropriately, such as the appropriate transport protocol and parameter settings as an operating environment changes.

• Machine learning techniques that utilize a static number of equations for learning address Challenge 2 in Section III-B by providing predictable time complexities for determining appropriate adjustments. In particular, we apply overfitted ANNs to QoS-enabled middleware to support enterprise DRE systems by incorporating the appropriate transport protocol adjustments according to feedback provided while the technique is trained. When an ANN is used in an enterprise DRE system, the time to determine an appropriate adjustment is bounded by the constant number of equations involved.

• Overfitting the machine learning technique addresses Challenge 3 in Section III-C by increasing the technique's accuracy. Our approach increases the accuracy of determining appropriate adjustments for specific operating environments by increasing the number of hidden nodes that connect the operating environment properties, such as CPU speed and network bandwidth, with the appropriate adjustments, such as transport protocols to support QoS. Specifically, overfitting an ANN provides accuracy equal to policy-based approaches.

V. Experimental Results

The section presents the results of experiments we conducted using an ANN to determine development complexity, timeliness, and accuracy in selecting an appropriate ADAMANT configuration given a particular operating environment. The experimental input data used to train the ANN include ADAMANT with multiple properties of the operating environment varied (*e.g.*, CPU speed, network bandwidth, DDS implementation, percent data loss in the network), along with multiple properties of the application being varied (*e.g.*, number of receivers, sending rate of the data), as would be expected with SAR operations.

We collected 394 inputs from previous experiments where an input consists of data values that determine a particular operating environment (*e.g.*, CPU speed, network bandwidth, number of data receivers, sending rate). We also provided the expected output to the ANN, *i.e.*, the transport protocol that provided the best QoS with respect to data reliability, average latency, and *jitter* (*i.e.*, standard deviation of the latency of network packets). An example of one of the 394 inputs is the following: 3 data receivers, 1% network loss, 25Hz data sending rate, 3GHz CPU, 1Gb network, using the OpenSplice DDS implementation, and specifying reliability and average latency as the QoS properties of interest. Based on our experiments, the corresponding output would be the NAK-based multicast protocol with a 1 ms retransmission timeout.

A. Evaluating the Development Complexity of Policy-based Approaches

Policy-based approaches support a straightforward way to determine optimal transport protocols for a given operating environment. These approaches can direct the system to alter its behavior after certain operating conditions are checked and met. Figure 2 shows an example where the application checks for the following environment properties applicable to the experimental data we collected relevant to SAR operations:

```
if (network_loss_percent == 1 && num_receivers < 5
    && sending_rate <= 25Hz && CPU_speed == 3GHz
    && RAM == 2GB && net_bw == 1Gb
    && DDS_impl == OpenSplice
    && metric == reliability_and_avg_latency) {
  transport_framework->use (transport1);
} else if (network_loss_percent == 3
          && num_receivers >= 5 && num_receivers < 10
          && sending_rate > 50Hz && CPU_speed == 850MHz
          && RAM == 500MB && net_bw == 100Mb
          && DDS_impl == OpenDDS
          && metric == reliability_and_jitter) {
  transport_framework->use (transport2);
} else if ...
```

Fig. 2. **Policy-based Example**

(1) percentage loss in the network (*i.e.*, network_loss_percent), (2) number of data receivers (*i.e.*, num_receivers), (3) the rate of publishing data (*i.e.*, sending_rate), (4) the CPU processing speed (*i.e.*, CPU_speed), (5) the random-access memory available (*i.e.*, RAM), (6) the network bandwidth provided (*i.e.*, net_bw), (7) the DDS implementation used (*i.e.*, DDS_impl), and (8) the QoS properties of interest (*i.e.*, metric).

Policy-based approaches can be optimized since the bounded number of (1) conditions that are checked and (2) the behaviors used to direct the system are explicitly identified. For example, a switch statement or nested if statements in a programming language can be used to implement policy-based approaches, as shown in Figure 2. In general, policy-based approaches can provide bounded times in searching for an adaptation solution and therefore address the boundedness evaluation criterion of Challenge 2 in Section III-B for adaptation approaches (*e.g.*, switch statements can be optimized for predictable performance). Policy-based approaches also are highly accurate for known solutions since developers can codify the exact behavior needed for a known environment, thereby addressing Challenge 3 in Section III-C.

Accidental complexity increases, however, when the conditions and responses for policy-based approaches are managed manually. Of the 8 properties shown in Figure 2, 6 properties can take an infinite range of potential values which cause an infinite number of combinations to be checked. Moreover, if the policies need to be modified the chance of introducing an error increases with the number of properties considered along with the number of ranges of values for each property. Policy-based approaches thus do not address the accidental development complexity criterion for adaptation approaches, whereas ANNs manage this complexity automatically as part of its learning.

B. Evaluating the Accuracy of ANNs

Our first step to using an ANN was to train it on the 394 inputs described in Section V. We used the *Fast Artificial Neural Network* (FANN) library (`leenissen.dk/fann`) as our ANN implementation due to its configurability, documentation, ease of use, and open-source availability. FANN offers extensive configurability for the neural network including the number of hidden nodes that connect the inputs with the output.

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 10 different times. The weights of the ANN determine how strong connections are between nodes. The weights are randomly initialized and these initial values have an effect on how well and how quickly the ANN learns.

Figure 3 shows the accuracies for the ANN configured with 6, 12, 24, and 36 hidden nodes over 10 training runs. Figure 3

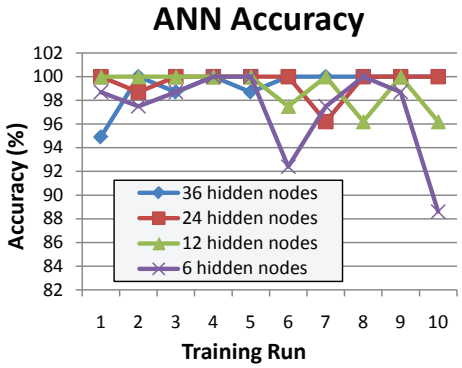

Fig. 3. **Accuracy of ANN with 6, 12, 24, & 36 hidden nodes**

also shows the effect of random initial weights on the accuracy of the ANN since the accuracy can vary across training runs. Accuracy was determined by querying the ANN with the data on which it was trained.

A 100% accurate classification was generated at least once with all hidden node configurations. The ANN with 24 hidden nodes provided the best accuracy across all the training runs even compared to using 36 hidden nodes—100% accuracy all but 2 times out of 10. We are therefore using the ANN configured with 24 hidden nodes to ensure accuracy for known environments and to minimize the difference between the expected and actual outputs for a single input configuration.

C. Evaluating the Timeliness of ANNs

As described in Challenge 2 in Section III-B, the datacenter for the SAR operations requires timely configuration adjustments. This section provides timing information for the ANN when queried for an optimal transport protocol. We used a 3 GHz CPU with 2GB of RAM running the Fedora Core 6 operating system with real-time extensions. Timeliness was determined by querying the ANN with all 394 inputs on which it was trained. A high resolution timestamp was taken before and after each call made to the ANN.

Figures 4 and 5 show the average response times and standard deviation of the response times, respectively, for 10 separate experiments where for each experiment we query the

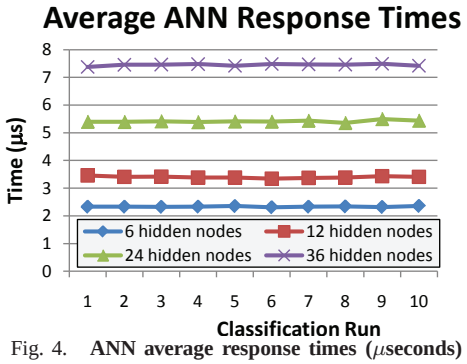

Fig. 4. **ANN average response times (μseconds)**

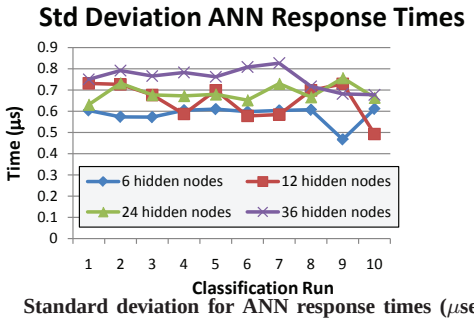

Fig. 5. **Standard deviation for ANN response times (μseconds)**

ANN for each of the 394 inputs. The figures show that the ANN provides timely and consistent responses. As expected, the response times using more hidden nodes are slower than response times with fewer hidden nodes. The increase in latency is less than linear, however (*e.g.*, response times using 12 hidden nodes are less than twice that using 6 hidden nodes).

VI. Concluding Remarks

The empirical results in this paper show how overfitted ANNs help address the development complexity, timeliness, and accuracy of adaptive enterprise DRE systems. For example, we used the FANN library to accurately determine which protocol to use to support the desired QoS in a given operating environment. We are also researching more generalized machine learning to handle adaptation in unknown environments, as described at `www.dre.vanderbilt.edu/~jhoffert/ADAMANT`.

References

[1] A. Choudhary. Policy based management in the global information grid. *International Journal of Internet Protocol Technology*, 3(1):72–80, 2008.

[2] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, 1995.

[3] A. Hess *et al.* *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, chapter Automatic Adaptation and Analysis of SIP Headers Using Decision Trees. Springer Berlin / Heidelberg, 2008.

[4] J. Hoffert *et al.* 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, April 2010.

[5] N. Shankaran *et al.* Hierarchical control of multiple resources in distributed real-time and embedded systems. *Real-Time Systems*, April 2007.

[6] X. Bu *et al.* A reinforcement learning approach to online web systems auto-configuration. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, pages 2–11, Washington, DC, USA, 2009. IEEE Computer Society.

[7] Dan W. Patterson. *Artificial Neural Networks: Theory and Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.