An Autonomous Performance Control Framework for Distributed Multi-Agent Systems: A Queueing Theory Based Approach

Nathan Gnanasambandam
gsnathan@psu.edu

Seokcheon Lee
stonesky@psu.edu

Soundar R.T. Kumara
skumara@psu.edu

Pennsylvania State University
310 Leonhard Building
University Park, PA 16802

ABSTRACT

Distributed Multi-Agent Systems (DMAS) such as supply chains functioning in highly dynamic environments need to achieve maximum overall utility during operation. The utility from maintaining performance is an important component of their survivability. This utility is often met by identifying trade-offs between quality of service and performance. To adaptively choose the operational settings for better utility, we propose an autonomous and scalable queueing theory based methodology to control the performance of a hierarchical network of distributed agents.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: *Design studies, modeling techniques, performance attributes*

General Terms

Performance

Keywords

Multi-Agent Systems, Survivability, Queueing Models

1. INTRODUCTION

With the emerging popularity of distributed multi-agent systems as application platforms, it is necessary that they survive dynamic and stressful environmental conditions, even partial permanent damage. While the survival notion necessitates adaptivity to diverse conditions along the dimensions of performance, security and robustness, delivering the correct proportion of these quantities can be quite a challenge. From a performance standpoint, a survivable system can deliver excellent Quality of Service (QoS) even when stressed. A DMAS could be considered survivable if it can maintain at least x% of system capabilities and y% of system performance in the face of z% of infrastructure loss and wartime loads (x, y, z are user-defined) [1].

We address a piece of the survivability problem by building an autonomous performance control framework for the DMAS drawing on the idea of composing the bigger society of smaller building blocks (i.e. agent communities) [3]. Identifying data-flows in the agent network (similar to [4]) and utilizing the network's service-level attributes such as delays, utilization and response times as a basis for its utility (like in [5]) we build a self-optimizing framework for DMAS. We believe that by using queueing theory we can analyze data-flows within the agent community as a network of queues with greater granularity in terms of processing delays and network latencies and also capitalize on using a building block approach by restricting the model to the community. We contribute by engineering a queueing theory based adaptation (control) framework to enhance the performance of the application layer, which inherently can be visualized as residing over the infrastructure (logical layer or middle-ware) and the physical layer (resources such as CPU, bandwidth).

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

2. FRAMEWORK ARCHITECTURE

Building on the ideas of high-level system specifications (or *TechSpecs*) and utilizing queueing network models (QNMs) for performance estimation as in [2] we build a real-time framework for application-level survivability. This framework is represented in Figure 1 and consists of activities, modules, knowledge repositories and information flow through a distributed collection of agents.

2.1 Architecture Overview

When the DMAS is stressed by an amount S by the underlying layers (due to under-allocation of resources) and the environment (due to increased workloads during wartime conditions), the *DMAS Controller* has to examine all its performance-related variables from set X and the current overall performance P in order to adapt. The variables that need to be maintained are specified in the *TechSpecs* and may include delays, time-stamps, utilization and their statistics. They are collected in a distributed fashion through the measurement points MP which are "soft" storage containers residing inside the agents and contain information on what, where and how frequently they should be measured. The *DMAS Controller* knows the set of flows F that traverse the network and the set of packet types T from the *TechSpecs*. With $\{F, T, X, C\}$, where C is a suggestion from the *DMAS Controller*, the *Model Builder* can select a suitable queueing model template Q. The *Control Set*

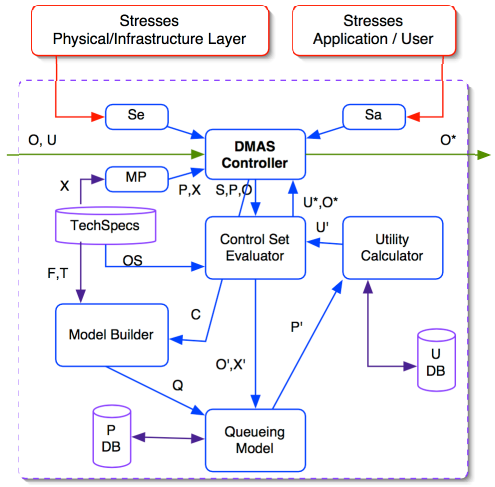

Figure 1: Architecture Overview

Evaluator knows the current operating mode (*opmode*) set O as well as the set of possible *opmodes*, OS from *TechSpecs*. To evaluate the performance due to a candidate opmode set $O^{'}$, the *Control Set Evaluator* uses the *Queueing Model* with a scaled set of operating conditions $X^{'}$. Once the performance $P^{'}$ is estimated by the *Queueing Model* it can be cached in the performance database PDB and then sent to the *Utility Calculator*. The *Utility Calculator* computes the domain utility due to $(O^{'}, P^{'})$ and caches it in the utility database, UDB. Subsequently, the optimal opmode set O^{*} is identified and sent to the *DMAS Controller*. The functional units of the architecture are distributed but for each community that forms part of a MAS society, O^{*} will be calculated by a single agent. We now examine the capabilities of the framework.

2.1.1 *Self-Monitoring Capability*

TechSpecs acts as a distributed structure that contains meta-data about all variables, X, that have to be monitored in different portions of the community. The data/statistics collected in a distributed way, is then aggregated to assist in control alternatives by the top-level controller that each community possesses. Each agent can look up its own *TechSpecs* and from time-to-time forward a measurement to its superior. The superior can analyse this information (eg. calculate statistics such as delay, delay-jitter) and/or add to this information and forward it again.

2.1.2 *Self-Modeling Capability*

One of the key features of this framework is that it has the capability to choose a type of model for representing itself for the purpose of performance evaluation. The system is equipped with several queueing model templates that it can utilize to analyze the system configuration with. The inputs to the *Model Builder* are the flows that traverse the network (F), the types of packets (T) and the current configuration of the network. Given we know that there are n agents interconnected in a hierarchical fashion, this unit represents the information in the required template format (Q) which is subsequently used to analyze the current performance.

2.1.3 *Self-Evaluating Capability*

The evaluation capability allows the MAS to examine its own performance under a given set of plausible conditions. This prediction of performance is used for the elimination of control alternatives that may lead to instabilities. Given that a variety of tasks traverse the heterogeneous network of agents in predefined routes (called *flows*), the processing and wait times of tasks at various points in the network are not alike because of dissimilar configurations, resource availabilities and/or environmental stresses. Under these conditions, performance is evaluated in terms of end-to-end delays for the "sense-plan-respond" cycles.

2.1.4 *Self-Controlling Capability*

Since tasks can be processed at various pre-defined qualities, *opmodes* allow for trading-off quality of service (task quality) for performance (end-to-end response time). The available resources fluctuate depending on stresses $S = S_e + S_a$, where S_e are the stresses from the environment (i.e. multiple contending applications) and S_a are the application stresses (i.e. increased tasks). Using current measured performance P and the measured stress S the *DMAS Controller* relates the overall utility (U) as $U(P,S) = \sum w_n x_n$ where x_n is the actual utility component and w_n is the associated weight specified by the user. To adjust P to get the best achievable utility under S, the following is done. Since P depends on O, which is a vector of opmodes collected from the community, we can use the QNM to find O^{*} and hence P^{*} that maximizes $U(P,S)$ for a given S from within the set OS. In words, we find the vector of *opmodes* (O^{*}) that maximizes domain utility at current S. The utility computation is performed in the *Utility Calculator* module using a learned utility model based on UDB.

3. CONCLUSIONS

We combined queueing analysis and application-level control to engineer a generic framework that is capable of self-optimizing its domain-specific utility to assure application-level survivability. While application-level adaptivity yields improvement in utility further gains are possible by leveraging underlying layers.

4. ADDITIONAL AUTHORS

Additional Authors: Natarajan Gautam (Pennsylvania State University, email: `ngautam@psu.edu`), Wilbur Peng and Vikram Manikonda (IAI Inc., email: `wpeng,vikram@i-a-i.com`), Marshall Brinn (BBN Technologies, email: `mbrinn@bbn.com`) and Mark Greaves (DARPA IXO, email: `mgreaves@darpa.mil`).

5. REFERENCES

[1] M. Brinn and M. Greaves. Leveraging agent properties to assure survivability of distributed multi-agent systems. *Proceedings of the Second Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.

[2] N. Gnanasambandam, S. Lee, N. Gautam, S. R. T. Kumara, W. Peng, V. Manikonda, M. Brinn, and M. Greaves. Reliable mas performance prediction using queueing models. *IEEE Multi-agent Security and Survivabilty Symposium*, 2004.

[3] H. Jung and M. Tambe. Performance models for large scale multi-agent systems: Using distributed pomdp building blocks. *Proceedings of the Second Joint Conference on Autonomous Agents and Multi-Agent Systems*, July 2003.

[4] O. F. Rana and K. Stout. What is scalabilty in multi-agent systems? *Proceedings of the Fourth International Conference on Autonomous Agents*, 2000.

[5] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. *Autonomous Agents and Multi-Agent Systems*, 2004.