A Framework for Performance Control of Distributed Autonomous Agents

Nathan Gnanasambandam, Seokcheon Lee, Soundar R.T. Kumara and Natarajan Gautam
310 Leonhard Building, The Pennsylvania State University, University Park, PA, 16802, USA

Abstract

We propose an autonomous and scalable queueing theory-based methodology to control the performance of a hierarchical network of distributed agents. Multi-agent systems (MAS) such as supply chains functioning in highly dynamic environments need to achieve maximum overall utility during operation. Hence, the objective of the control framework is to identify the trade-offs between quality and performance and adaptively choose the operational settings to posture the MAS for better utility. By formulating the MAS as an open queueing network with multiple classes of traffic we evaluate the performance and subsequently the utility, from which we identify the control alternative for a localized, multi-tier zone.

Keywords: Queueing Network, Multi-Agent Systems, Performance control.

1 Introduction

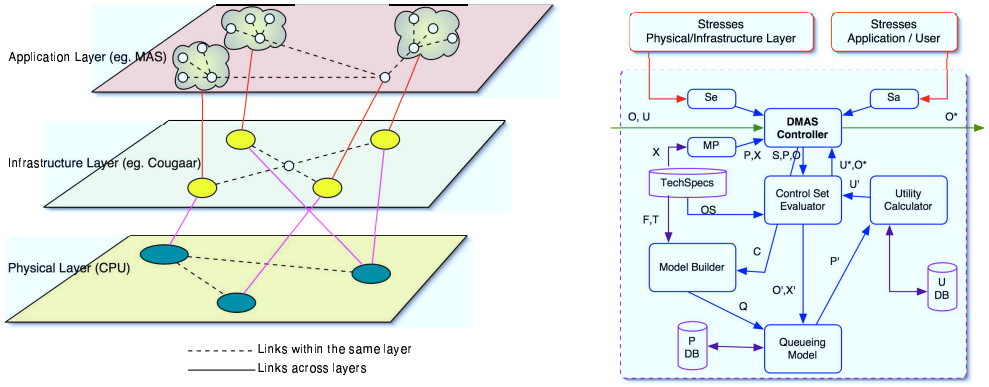
With the growing view of agent-oriented software sytems [1] and the increased deployment of distributed multi-agent systems (DMAS) for numerous emerging applications such as computational grids, e-commerce hubs, supply chains and sensor networks, we are faced with large-scale distributed agents whose performance needs to be estimated and controlled. Often times, these DMAS operate in dynamic and stressful environmental conditions, of one type or the other, in which the MAS as whole must survive. While 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. In this paper, we address a piece of this problem by building an autonomous performance control framework for MAS.

While building large multi-agent societies (such as UltraLog [2]), it is desirable that the associated adaptation framework be *generic* and *scalable*. One way to do this is to utilize a methodology similar to Jung and Tambe [3], where the bigger society is composed of smaller building blocks, in this case, corresponding to communities of agents. Although, strategies for co-operativeness and distributed POMDP to have been utilized to analyze performance in [3], an increase in the number of variables in each agent can quickly render POMDP ineffective even in reasonable sized agent communities due to the state-space explosion problem. In [4], Rana and Stout identify data-flows in the agent network and model scalability with Petri nets, but their focus is on identifying synchronization points, deadlocks and dependency constraints with coarse support for performance metrics relating to delays and processing times for the flows. In a recent architecture for autonomic computing, Tesauro et al. [5] build a real-time MAS-based framework that is self-optimizing based on application-specific utility. While [3, 4] motivate the need to estimate performance of large DMAS using a building block approach, [5] justifies the need to use domain specific utility whose basis should be the network's service-level attributes such as delays, utilization and response times.

We believe that by using queueing theory we can analyze data-flows within the agent community 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 agent community. Queueing theory has been widely used in networks and operating systems [6]. However, the authors have not seen the application of queueing to MAS modeling and analysis. Since, agents lend themselves to being conveniently represented as a network of queues, we concentrate on engineering a queueing theory based adaptation (control) framework to enhance the *application-level performance*.

Inherently, the DMAS can be visualized as a multi-layered system as is depicted in Figure 1a. The top-most layer is where the application resides, usually conforming to some organization such as mesh, tree etc. The infrastructure layer not only abstracts away many of the complexities of the underlying resources (such as CPU, bandwidth), but more importantly provides services (such as Message Transport) and aiding agent-agent services (such as naming, directory etc.). The bottom most layer is where the actual computational resources, memory and bandwidth reside. Most studies in the literature do not make this distinction and as such control is not executed in a layered fashion. Some studies such as [7, 8], consider controlling attributes in the physical or infrastructural layers so that some properties (eg. survivability) could result and/or the facilities provided by these layers are taken advantage of. Often, this requires rewiring the physical layer, availability of a infrastructure level service or the ability of the application to share information with underlying layers in a timely fashion for control purposes. In this work, we consider control only due to application-level trade-offs such as quality of service versus performance and assume that infrastructure level services (such as load-balancing, priority scheduling) or physical level capabilities (such as rewiring) are not possible. This does not exclude the possibility that in future we can combine all approaches to achieve a multi-layered control.

Our contribution in this work is to combine queueing analysis and application-level control to engineer a generic framework that is capable of self-optimizing its domain-specific utility.



(a) Operational Layers

(b) Framework Architecture

Figure 1: MAS framework

1.1 Problem Statement

Typically, the top-most layer in the computing infrastructure (here the DMAS-based application) possesses maximum transparency to system's overall utility, control-knobs and domain knowledge. The utility of the application is the combined benefit along several conflicting (eg. completeness and timeliness [9, 2]) and/or independent (eg. confidentiality and correctness [9, 2]) dimensions, which the application tries to maximize in a best-effort sense through trade-offs. Understandably, in a distributed multi-agent setting, mechanisms to measure, monitor and control this multi-criteria utility function become hard and inefficient, especially under conditions of scale-up. Given that the application does not change its high-level goals, task-structure or functionality in real-time, it is beneficial to have a framework that assists in the choice of operational modes (or *opmodes*) in a distributed way. Hence, the research objective of this work is to design and develop a generic, real-time framework for DMAS, that utilizes a queueing network model for performance evaluation and a learned utility model to select an appropriate control alternative.

1.2 Solution Methodology

The focus of this research is to adjust the application-level parameters or *opmodes* within the distributed agents to make an autonomous choice of operational parameters for agents in a reasonable-sized domain (called an agent community). The choice of *opmodes* is based on the perceived application-level utility of the combined system (i.e. the whole community) that current environmental conditions allow. We assume that the application's utility depends of the choice of *opmodes* at the agents constituting the community because the *opmodes* directly affect the performance. A queueing network model is utilized to predict the impact of DMAS control settings and environmental conditions on steady-state performance (in terms of end-to-end delays in tasks), which in turn is used to estimate the application-level utility. After evaluating and ranking several alternatives from among the feasible set of operational settings on the basis of utility, the best choice is picked.

2 Architecture of the Performance Control Framework

We implement the performance control framework for the Continuous Planning and Execution **(**CPE**)** Society which is a command and control MAS built on *Cougaar* (DARPA Agent Platform [10]). While we describe the functionality of the components of the framework (Figure 1b) in this section, we highlight the autonomic capabilities that are built into the system.

2.1 Overview of Application (CPE) Scenario

In our set-up, the primary building block consists of three tiers in the application layer. CPE embodies a complete military logistics scenario with agents emulating roles such as suppliers, consumers and controllers all functioning in a dynamic and hostile (destructive) external environment. Embedded in the hierarchical structure of CPE are both command and control, and superior-subordinate relationships. The subordinates compile sensor *updates* and furnish them to superiors. This enables the superiors to perform the designated function of creating *plans* (for maneuvering and supply) as well as *control* directives

for downstream subordinates. Upon receipt of plans, the subordinates execute them. The supply agents replenish consumed resources periodically. This high level system definition is the functionality of CPE that it seeks to perform repeatedly with maximum utility while residing in the application layer.As part of the application-level adaptivity features, a set of *opmodes* are built into the system. *Opmodes* allow individual tasks (such as *plans, updates, control)* to be executed at different *qualities* or to be processed at different rates. We assume that *TechSpecs* for the CPE scenario are available to be utilized by the control framework. The framework that accomplishes the aforementioned goal of CPE in a distributed fashion while performing at a maximum possible level of utility is represented in Figure 1b.

2.2 Self-Monitoring Capability

Any system that wants to control itself should possess a clear specification of the scope of the variables it has to monitor. The *TechSpecs* is a distributed structure that supports this purpose by housing all variables, X, that have to be monitored in different portions of the community (or sub-system). The data/statistics collected in a distributed way, is then aggregated to assist in control alternatives by the top-level controller that each community will possess.

The attributes that need to be tracked are formulated in the form of *measurement points* (MP). The measurement points are "soft" storage containers residing inside the agents and contain information on what, where and how frequently they should be measured. Each agent can look up its own *TechSpecs* and from time-to-time forward that 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. We have measurement points for time-periods, time-stamps, operating-modes, control and generic vector-based measurements. These measurement points can be chained for tracking information for a flow such that information is tagged-on at every point the flow traverses. For the sake of reliability, the information that is contained in these agents is replicated at several points, so that when packets do not reach on time or not reach at all, previously stored packets can be utilized for control purposes.

2.3 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 with. The type of model that is utilized at any given moment is based on accuracy, computation time and history of effectiveness. For example, a simulation based queueing model may be very accurate but cannot complete evaluating enough alternatives in limited time, in which case an analytical model (such as BCMP, QNA [11]) is preferred.

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. If at a given time, we know that there are n agents interconnected in a hierarchical fashion then the role of this unit is to represent that information in the required template format (Q). The current number of agents is known to the controller by tracking the measurement points. For example, if there is no response from an agent for a sufficient period of time, then for the purpose of modeling, the controller may assume the agent to be non-existent. In this way dynamic configurations can be handled. On the other hand, *TechSpecs* do mandate connections according to superior-subordinate relationships thereby maintaining the flow structure at all times. Once the modeling is complete, the MAS has to capability to analyze its current performance using the selected type of model. The MAS does have the flexibility, to choose another model template for a different iteration.

2.4 Self-Evaluating Capability

The evaluation capability, the first step in control, 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. Our notion of performance evaluation is similar to [5]. While Tesauro et al. [5] compute the resource level utility functions (based on the application manager's knowledge of system performance) that can be combined to obtain a globally optimal allocation of resources, we predict the performance of the MAS as a function of its operating modes in real-time (within *Queueing Model*) and then use it to calculate its global utility. By introducing a level of indirection, we may get some desirable properties (explained in Section 4.2) because we separate an application's domain-specific utility computation from performance prediction (or analysis). This theoretically enables us to predict the performance of *any* application whose *TechSpecs* are clearly defined and then compute the application-specific utility. In both cases, control alternatives are picked based on best-utility. We discuss the notion of control alternatives in Section 2.5. Also, our performance metrics (and hence utility) are based on service level attributes such as end-to-end delay and latency, which is a desirable attribute of autonomic systems [5].

When *plan*, *update* and *control* tasks (as mentioned in Section 2.1) flow in this 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. This is because the configuration (number of agents allocated on a node), resource availability (load due to other contending software) and environmental conditions at each agent is different. In addition, the tasks themselves can be of varying qualities or fidelities that affects the time taken to process that task. Under these conditions, performance is estimated on the basis of the end-to-end delay involved in a "sense-plan-respond" cycle.

Table 1: Notation

Symbol	Description
N	Total # of nodes in the community
λ_{ij}	Average arrival rate of class j at node i
$1/\mu_{ijk}$	Average processing time of class j at node i at quality k
M	Total number of classes
T_i	Routing probability matrix for class i
W_{ijk}	Steady state waiting time for class j at node i at quality k
Q_{ij}	Set of qualities at which a class j task can be processed at node i

The primary performance prediction tool that we use are called Queueing Network Models (QNM) [6]. The QNM is the representation of the agent community in the queueing domain. As the first step of performance estimation, the agent community needs to be translated into a queueing network model. Table 1 provides the notations used is this section. Inputs and outputs at a node are regarded as tasks. The rate at which tasks of class j are received at node i is captured by the arrival rate (λ_{ij}). Actions by agents consume time, so they get abstracted as processing rates (μ_{ij}). Further, each task can be processed at a quality $k \in Q_{ij}$, that causes the processing rates to be represented as μ_{ijk}. Statistics of processing times are maintained at each agent in the Performance Database (*PDB*) to arrive at a linear regression model between quality k and μ_{ijk}. Flows get associated with classes of traffic denoted by the index j. If a connection exists between two nodes, this is converted to a transition probability p_{ij}, where i is the source and j is the target node. Typically, we consider flows originating from the environment, getting processed and exiting the network making the agent network an open queueing network [6]. Since we may typically have multiple flows through a single node, we consider multi-class queueing networks where the flows are associated with a class. Performance metrics such as delays for the "sense-plan-respond" cycle is captured in terms of average waiting times, W_{ijk}. As mentioned earlier, *TechSpecs* is a convenient place where information such as flows and Q_{ij} can be embedded.

The choice of QNM depends on the number of classes, arrival distribution and processing discipline as well as a suggestion C by the *DMAS controller* that makes this choice based upon history of prior effectiveness. Some analytical approaches to estimate performance can be found in [6, 11]. In the context of agent networks, Jackson and BCMP queueing networks have been applied to estimate the performance in [12]. By extending this work we provide several templates of queueing models (such as BCMP, Whitt's QNA [11], Jackson, M/G/1, a simulation) that can be utilized for performance prediction.

2.5 Self-Controlling Capability

In contrast to [5], we deal with optimization of the domain utility of a MAS that is distributed, rather than allocating resources in an optimal fashion to *multiple* applications that have a good idea of their utility function (through policies). As mentioned before *opmodes* allow for trading-off quality of service (task quality and response time) and performance. We are assuming there is a maximum ceiling R on the amount of resources, and 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, changes in the infrastructural or physical layers) and S_a are the application stresses (i.e. increased tasks). The $DMAS\ controller$ receives from MP (measurement points) a measurement of the actual performance P and a vector of other statistics (X) about task processing times. Also at the top-level the overall utility (U) is $U(P,\ S) = \sum w_n x_n$ is known where x_n is the actual utility component and w_n is the associated weight. We cannot change S, but we can adjust P to get better utility. 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. In words, we find the vector of *opmodes* (O^*) that maximizes domain utility at current S and update O. This computation is performed in the *Utility Calculator* module using a utility model that is learned and stored in the Utility Database (UDB). This formulation although independently found matches the self-optimization notion in [5]. But some differences exist as follows. Tesauro et al. [5] assume that the system's knowledge includes a performance model, which we do not assume. We use a queueing network model to estimate the performance in real-time for any set of *opmodes* O' by taking the current set of *opmodes* O and scaling them appropriately based on observed histories (X) to X' in the *Control Set Evaluator*. Also, we deal with a single MAS with an overall utility function for the entire distributed functionality (within the community). Because of the interactions involved and complexity of performance modeling[3, 4], it may be time-consuming to utilize inferencing and learning mechanisms in real-time. This is why we use an analytical queueing network to get the performance estimate quickly. Another difference is that in [5], they assume operating system support which may not be true in many MAS-based situations because of mobility, security and real-time constraints. Furthermore, in addition to the estimation of performance, the queueing model may have the capability to eliminate instabilities from a queueing sense, which is not apparent in the other approach. But inspite of these differences, it is interesting to see that self-controlling capability can be achieved, with or without explicit layering, in a couple of real-world applications.

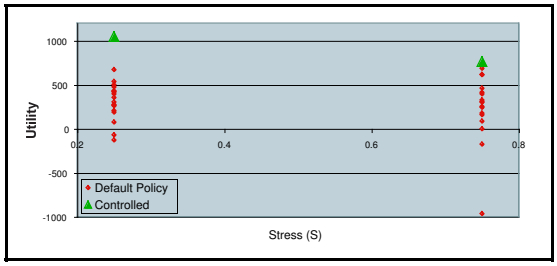
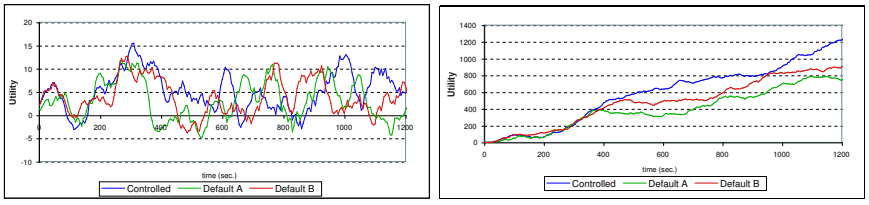
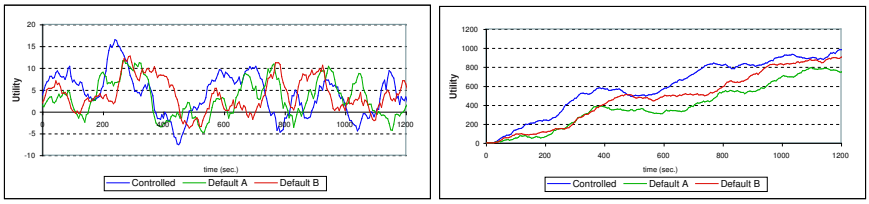


Figure 2: Results overview



(a) Instantaneous Utility (stress 0.25)

(b) Cumulative Utility (stress 0.25)



(c) Instantaneous Utility (stress 0.75)

(d) Cumulative Utility (stress 0.75)

Figure 3: Sample results

3 Empirical Evaluation on CPE Test-bed

We utilized the prototype CPE framework to run 36 experiments at two stress levels ($S = 0.25$ and 0.75). The scenario consisted of 14 agents, besides a world agent that created random scenarios in military logistics for the agents to react to. There were three layers of hierarchy with a three-way branching at each level and one supply node. The community's utility function was based on the achievement of real goals in military engagements such as terminating or damaging the enemy and reducing the penalty involved in consuming resources such as fuel or sustaining damage. We also assumed for the model selection process that the external arrival was Poisson and the service times were exponentially distributed. In order to cater to general arrival rates, our framework contains a QNA- and simulation-based model. Using this assumption a BCMP or M/G/1 queueing model could be utilized. We used the Cougaar based default control without additional support from our framework as the baseline (denoted as *Default A* and *Default B*) and found that controlling the agent community using our framework (denoted as *controlled*) was beneficial in the long run. The overview of the results is provided in Figure 2.

At both stress levels, the controlled scenario performed better that the default as shown in Figure 3. We did observe oscillations in the instantaneous utility and we attribute this to the impreciseness of the prediction of stresses. Stresses vary relatively fast in the order of seconds while the control granularity was of the order of minutes. Since this is a military engagement situation following no pre-determined stress patterns, it is hard to cope with in the higher stress case. We think that this could be the reason why our utility falls in the latter case.

4 Conclusions and Future Work

4.1 Conclusions

In this paper, we were able to successfully control a real-time MAS to achieve overall better utility in the long run using application-level trade-offs between quality of service and performance. We utilized a queueing network based framework for performance analysis and subsequently used a learned utility model for computing the overall benefit to the MAS (i.e. community). While Tesauro et al. [5] have found a similar construction to improve utility in multiple applications, we concentrated on optimizing the utility of a single distributed application using queueing theory. We think that the approaches are complementary, with this study providing empirical evidence to support the observation in [1] that agents can be used to optimize distributed application environments, including themselves, through flexible high-level (i.e. application-level) interactions.

4.2 Discussion and Future Work

We believe that keeping the building-blocks small and the number of interactions (between performance and utility models) minimal may assist in making the framework more *flexible* and *scalable.* For example, if system size increases, we can consider a superior agent or *human user* to be at the next higher level controlling the weights in the utility function without affecting the performance model. The larger system with supervisory control would then be analyzed using another higher-level QNM or a network of networks. *TechSpecs* has assisted this effort to a large extent, re-emphasizing the well-founded *separation principle* (separating knowledge/policy and mechanism) in the computing field. While we think that the aforementioned architectural principles have been well-utilized, we hope to broaden the layered control approach to encompass the infrastructural-level control into the framework. Another avenue for improvement is to design self-protecting mechanisms within our framework so that the security aspect of the framework is reinforced.

Acknowledgments

The work described here was performed under the DARPA UltraLog Grant#: MDA972-1-1-0038. The authors wish to acknowledge DARPA for their generous support.

References

[1] Jennings, N. R. and Wooldridge, M., 2000, “Agent-Oriented Software Engineering”, Handbook of Agent Technology, AAAI/MIT Press.

[2] UltraLog Program Site. www.ultralog.net. DARPA.

[3] Jung, H., and Tambe, M., 2003, “Performance Models for Large Scale Multi-Agent Systems”, Proceedings of the Seocnd Joint Conference on Autonomous Agents and Multi-Agnet Systems.

[4] Rana, O. F., and Stout, O., “What is Scalability in Mult-Agent Systems”, 2000, Proceedings of the Fourth International Conference on Autonomous Agents.

[5] Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Whalley, I., Kephart, J. O., and White, S. R., 2004, “A Multi-Agent Systems Approach to Autonomic Computing”, Autonomous Agents and Multi-Agent Systems.

[6] Bolch, G., de Meer, H., Greiner, S., and Trivedi, K. S., 1998, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. John Wiley and Sons.

[7] Thadakamalla, H. P., Raghavan, U. N., Kumara, S. R. T., and Albert, R., 2004, “Survivability of Multi-Agent Systems - A Topological Perspective”, IEEE Intelligent Systems: Dependable Agent Systems, vol. 19, no. 5, pp. 24-31, Sep/Oct 2004.

[8] Hong, Y., and Kumara, S. R. T., 2004, “Coordinating Control Decisions of Software Agents for Adaptation to Dynamic Environments”, Working Paper, Marcus Department of Industrial and Manufacturing Engineering, Pennsylvania State Univerity, University Park, PA.

[9] Brinn, M., and Greaves, M., 2003, “Leveraging Agent Properties to Assure Survivability of Distributed Multi-Agent Systems”, in the Proceedings of the Second Joint Conference on Autonomous Agents and Multi-Agent Systems.

[10] Cougaar Open Source Site. www.cougaar.org. DARPA.

[11] Whitt, W., 1983, “The Queueing Network Analyzer”, The Bell System Technical Journal, vol. 62, no. 9, pp. 2779-2815.

[12] Gnanasambandam, N., Lee, S., Gautam, N., Kumara, S. R. T., Peng, W., Manikonda, V., Brinn, M., and Greaves, M., 2004, “Reliable MAS Performance Prediction Using Queueing Models”, IEEE Multi-Agent Security and Survivability Symposium.