Survivability of a Distributed Multi-Agent Application - A Performance Control Perspective

Nathan Gnanasambandam, Seokcheon Lee, Soundar R.T. Kumara, Natarajan Gautam
Pennsylvania State University
University Park, PA 16802
{gsnathan, stonesky, skumara, ngautam}@psu.edu

Wilbur Peng, Vikram Manikonda
Intelligent Automation Inc.
Rockville, MD 20855
{wpeng, vikram}@i-a-i.com

Marshall Brinn
BBN Technologies
Cambridge, MA 02138
mbrinn@bbn.com

Mark Greaves
DARPA IXO
Arlington, VA 22203
mgreaves@darpa.mil

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. 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. When the problem scales, another larger queueing network could be composed using zones as building-blocks. This method advocates the systematic specification of the DMAS's attributes to aid real-time translation of the DMAS into a queueing network. We prototype our framework in Cougaar and verify our results.

1. Introduction

Distributed multi-agent systems (DMAS), through adaptivity, have enormous potential to act as the "brains" behind numerous emerging applications such as computational grids, e-commerce hubs, supply chains and sensor networks [13]. The fundamental hallmark of all these applications is dynamic and stressful environmental conditions, of one type or the other, in which the MAS as whole must survive albeit it suffers temporary or 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) [7].

We address a piece of the survivability problem by building an autonomous performance control framework for the DMAS. It is desirable that the adaptation framework be *generic* and *scalable* especially when building large-scale DMAS such as UltraLog [2]. For this, one can utilize a methodology similar to Jung and Tambe [19], composing the bigger society of smaller building blocks (i.e. agent communities). Although Jung and Tambe [19] successfully employ strategies for co-operativeness and distributed POMDP to analyze performance, 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 [27], Rana and Stout identify data-flows in the agent network and model scala-

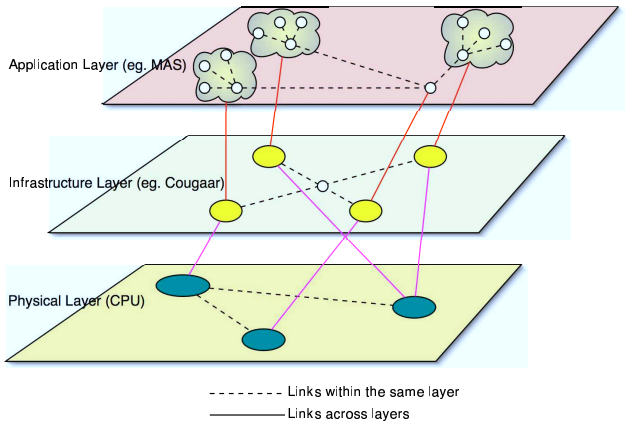


Figure 1. Operational Layers forming the MAS

bility 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. Tesauro et al. [34] propose a real-time MAS-based approach for data centers that is self-optimizing based on application-specific utility. While [19, 27] motivate the need to estimate performance of large DMAS using a building block approach, [34] 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 community. Queueing theory has been widely used in networks and operating systems [5]. 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 1 . 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 [35, 17], consider controlling attributes in the physical or infrastructural layers so that some properties (eg. robustness) 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 of share information with underlying layers in a timely fashion for control purposes. In this initial 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. While we intend to extend the approach to multi-layered control, it must be noted that it is not always possible for the application (or the application manager) to have access to all the underlying layers due to security reasons. In autonomic control of data centers, the application manager may have complete authority over parameters in the physical layer (servers, buffers, network), the infrastructure (middle-ware) and the applications. However, in DMAS scenarios, especially when dealing with mobile agents (as an application), trust between the layers is often partial forcing them to negotiate parameters through authorized channels. Hence, each layer must be capable of adapting with minimum cross-layer dependencies.

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. Secondly, we provide a methodology for engineering a self-optimizing DMAS to assure application-level survivability. While we see utility improvements by by adopting application-level adaptivity, we understand that further improvement may be gained by utilizing the adaptive capabilities of the underlying layers.

Before we consider the details of our framework, we classify the performance control approaches in literature in Section 2. We present the details for our Cougaar based test-bed system in Section 3. The architectural details of our framework is provided in Section 4. We provide an empirical evaluation in Section 5 and finally conclude with discussions and future work in Section 6.

2. Background and Motivation

2.1 Approaches in Literature

Because of the diversity of literature on control frameworks and performance evaluation, we examined a representative subset primarily on the basis of control objective, (component) interdependence and autonomy, generality, composability, real-time capability (off-line/on-line control) and layering in control architecture.

In some *AI based approaches* such as [32, 10], behavioral or rule based controllers are employed to make the system exhibit particular behavior based upon logical reasoning or learning. While performance is not the objective, layered learning is an interesting capability that may be helpful in a large scale MAS. Learning may be from a statistical sense as well where the parameters of a transfer function are learnt from empirical data to subsequently enforce feedback control [8]. Another architectural framework called MONAD [37], utilizes a hierarchical and distributed behavior-based control module, with immense flexibility through scripting for role and resource allocation, and co-ordination. While many these approaches favor the "sense-plan-act" or "sense and respond" paradigm and some partially support flexibility through scripting, some important unanswered questions are what happens when system size changes, can all axioms and behaviors be learnt a-priori and what is the performance impact of size (i.e. scalability)?

Control theoretic approaches in software performance optimization are becoming important [22, 29], with software becoming increasingly more complex, multi-layered and having real-time requirements. However, because of the dynamic system boundaries, size, varying measures of performance and non-linearity in DMAS it is very complex to design a strict control theoretic control process [21]. Some approaches such as [21, 34] take the heuristic path, with occasional analogs to control theory, with an emphasis on application or domain-specific utility. Kokar et al. [22] refer to this utility as *benefit function* and elaborate on various analogs between software systems and traditional control systems. From the perspective of autonomic control of computer systems, Bennani and Menasce [4] study the robustness of self-management techniques for servers under highly variable workloads. Although queueing theory has been used in this work, any notion of components being distributed or agent-based seems to be absent. Furthermore, exponential smoothing or regression based load-forecasting may not be sufficient to address situations caused by wartime dynamics, catastrophic failure and distributed computing. Nevertheless, in our approach we have a notion of controlling a distributed application's utility using queueing theory.

Numerous *market-based control mechanisms* are available in literature such as [24, 9, 12, 6]. In market-based control systems, agents emulate buyers and sellers in a market acting only with locally available information yet helping us realize global behaviour for the community of agents. While these methods are very effective and offer desirable properties such as decentralization, autonomy and control hierarchy, they have been used for resource allocation [24, 9] and resource control [6]. The Challenger [9] system seeks to minimize *mean flow time (job completion time - job origination time)*, the task is allocated to an agent providing least processing time. Load balancing is another application as applied by Ferguson et al. [12]. Resource allocation and load-balancing can be thought of as infrastructure level services, that agent frameworks such as Cougaar [1] provide and hence in our work we focus on application-level performance and the associated utility to the DMAS.

Using *finite state machines*, hybrid automata and their variants have been the foci of many research paths in agent control as in [11, 23]. The idea here is to utilize the states of the multi-agent system to represent, validate, evaluate, and choose plans that lead the system towards the goal. Often, the drawback here is that when the number of agents increase, the state-space approaches tend to become intractable.

Heuristics have widely been used in controlling multi-agent systems primarily in the following sense: searching and evaluating options based on domain knowledge and picking a course of action (maybe a compound action composed of a schedule of individual actions) eventually. The main idea in recent heuristics based control as exemplified by [36, 26, 31] is that schedules of actions are chosen based upon requirements such as costs, feasibilities for real-time contexts, complexity, quality etc. Opportunistic planning is an interesting idea as mentioned in Soto et al. [31] refers to the best-effort planning (maximum quality) considering available resources. These meta-heuristics offer very effective, special-purpose solutions to control agent behavior, however to be more flexible, we separate the performance evaluation and the domain-specific application utility computation.

Given that we have a model for performance estimation (whose parameters and state-space are known), dynamic programming (DP) and its adaptive version - reinforcement learning (RL), and model predictive control (MPC) have been used to find the control policy [3, 33, 20, 28, 25]. Since the complexity of finding the optimal policy grows exponentially with the state space [3] and convergence has to be ensured in RL [33, 20], we take an MPC-like approach in our work for finding quick solutions in real-time. We discuss this further in Section 4.

2.2 Related Work

In large scale MAS applications, performance estimation and modeling itself can be a formidable task as illustrated by [16] in the UltraLog [2] context. UltraLog [2], built on Cougaar [1], uses for heuristic control a host of architectural features such as operating modes, conditions, and plays and play-books as described in [21]. Helsinger et al. [15] incorporate the aforementioned features into their closed-loop heuristic framework that balances the different dimensions of system survivability through targeted

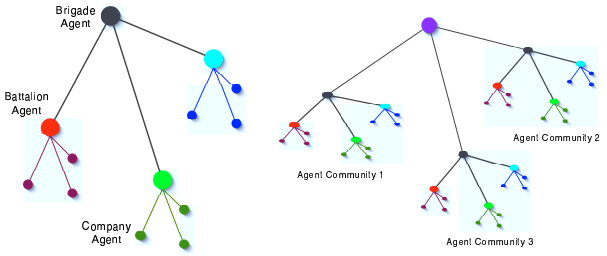
defense mechanisms, trade-offs and layered control actions. The importance of high-level, system specifications (interchangeably called *TechSpecs*, specification database, component database) has been emphasized in many places such as [18, 21, 14]. These specifications contain component-wise, static input/output behavior, operating requirements and control actions of agents along with domain measures of performance and computation methodologies [14]. Also, queueing network based methodologies for offline and design-time performance evaluation have been applied and validated in [14, 30]. Building on these ideas, we build a real-time framework with queueing based performance prediction capabilities.

2.3 Problem Statement

Being the top-most layer as an application, the survivability of a DMAS depends on its ability to leverage its knowledge of the domain, the system's overall utility and available control-knobs. The utility of the application is the combined benefit along several conflicting (eg. completeness and timeliness [7, 2]) and/or independent (eg. confidentiality and correctness [7, 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*) that maximize the utility from performance. Hence, the research objective of this work is to design and develop a generic, real-time, self-controlling framework for DMAS, that utilizes a queueing network model for performance evaluation and a learned utility model to select an appropriate control alternative.

2.4 Solution Methodology

This research concentrates on adjusting 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 on 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 flows), 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.



(a) MAS building block: Community

(b) Agent society formed by composing communities

Figure 2. MAS Community and Society

3. Overview of Application (CPE) Scenario

The Continuous Planning and Execution **(**CPE**)** Society is a command and control (C2) MAS built on *Cougaar* (DARPA Agent Framework [1]) that serves as the test-bed for performance control. Designed as a building block for larger scale MAS, the primary CPE prototype consists of three tiers (*Brigade*, *Battalion*, *Company*) as shown in Figure 2a. While the discussion is mainly with respect to the structure of CPE, the system can be grown by combining many CPE communities to form large agent societies as shown in Figure 2b.

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 to be executed continuously by the application with maximum achievable performance in the presence of stresses that include temporary and catastrophic failure. Stresses associated with wartime situations cause the resource allocation (CPU, memory, bandwidth) and offered load (due to increased planning requirements) to fluctuate immensely.

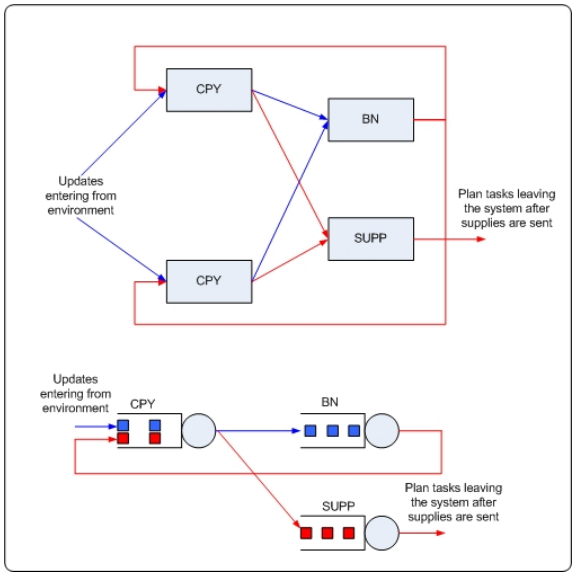


Figure 3. Traffic flow within CPE

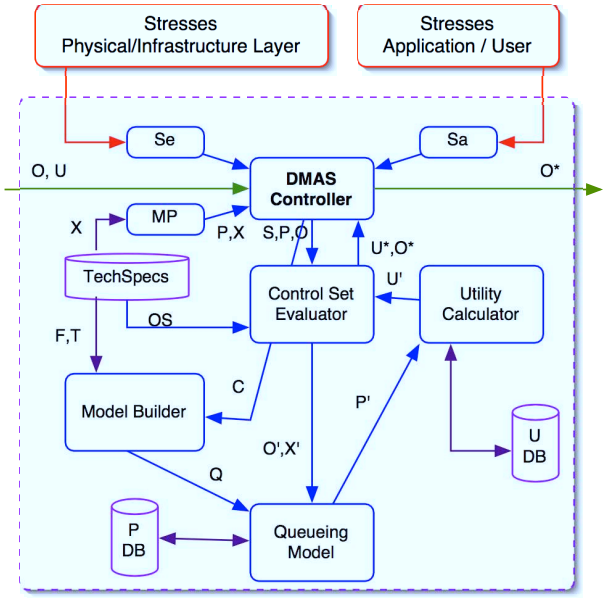


Figure 4. Architecture Overview

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 application (similar to [14]) are available to be utilized by the control framework.

Although, functionally CPE and UltraLog are unique, the same flavor of activities are reflected in both. Both of them share the same Cougaar infrastructure; execute planning in a dynamic, distributed settings with similar QoS requirements; and are both one application with physically distributed components interconnected by task flows (as shown in Figure 3 in the case of CPE), wherein the individual utilities of the components contribute to the global survivability.

4. Architecture of the Performance Control Framework

The distributed performance control framework that accomplishes application-level survivability while operating amidst infrastructure/physical layer and environmental stresses is represented in Figure 4. This representation consists of activities, modules, knowledge repositories and information flow through a distributed collection of agents. The features for adaptivity are solely at the application level without considering infrastructure or physical level adaptivity such as dynamically allocating processor share or adjusting the buffer sizes.

Architecture Overview

When the application 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, utilizations and their statistics. They are collected in a distribution fashion through the measurement points MP. 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 Evaluator* knows the current 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 operating mode 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 functionality and role offered by each component of the framework in greater detail.

4.1 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 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 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 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. 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 in the absence of packets reaching on time or not reaching at all, previously stored packets and their corresponding information can be utilized for control purposes.

4.2 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 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 [38]) 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.

4.3 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 [34]. While Tesauro et al. [34] compute the resource level utility functions (based on the application manager's knowledge of the system performance model) 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 (some more differences are pointed out in Section 4.4). By introducing a level of indirection, we may get some desirable properties (discussed in Section 6) 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 4.4. 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 [34].

When *plan*, *update* and *control* tasks (as mentioned in Section 3) 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.

The primary performance prediction tool that we use are

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

called Queueing Network Models (QNM) [5]. 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 *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 [5]. 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 [5, 38]. In the context of agent networks, Jackson and BCMP queueing networks to estimate the performance in [14]. By extending this work we support several templates of queueing models (such as BCMP, Whitt's QNA [38], Jackson, M/G/1, a simulation) that can be utilized for performance prediction.

4.4 Self-Controlling Capability

In contrast to [34], we deal with optimization of the domain utility of a *single* 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 the measurement points (MP) a measurement of the actual performance P and a vector of other statistics (relating to X). 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 specified by the user or another superior agent. 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 from within the set OS. In words, we find the vector of *opmodes* (O^*) that maximizes domain utility at current S and opmodes O. This computation is performed in the *Utility Calculator* module using a learned utility model based on UDB.

In addition to differences pointed out thus far, here are some more differences between this work and [34]:

- Tesauro et al. [34] assume that the application manager in Unity has a model of system performance, which we do not assume. Although they allude to a *modeler* module, they do not explain the details of their performance model. We use a queueing network model that is constructed in real-time to estimate the performance for any set of *opmodes* O' by taking the current *opmodes* O and scaling them appropriately based on observed histories (X) to X' in the *Control Set Evaluator*.

- Because of the interactions involved and complexity of performance modeling [19, 27], 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 [34], they assume operating system support for being able to tune parameters such as buffer sizes and operating system settings which may not be true in many MAS-based situations

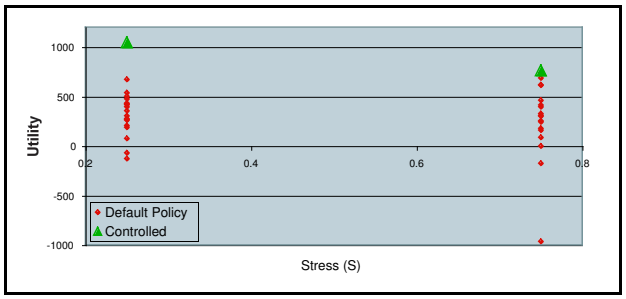


Figure 5. Results Overview

because of mobility, security and real-time constraints. Besides, 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 most importantly, their work reflects a two level hierarchy where the resource manager mediates several application environments to obtain maximum utility to the data center. But our work is from the perspective of a single, self-healing, multi-level application that is trying to be survivable by maximizing its own utility.

Inspite of these differences, it is interesting to see that the self-controlling capability can be achieved, with or without explicit layering, in real-world applications.

5. Empirical Evaluation on CPE Test-bed

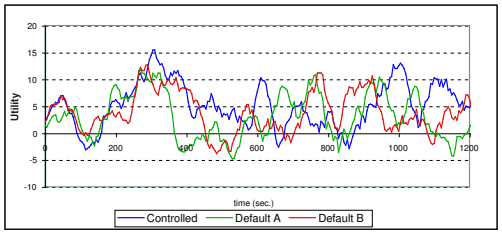
The aforementioned framework was implemented within CPE which we use as a test-bed for our experimentation. The main goal of this experimentation was to examine if application-level adaptivity led to any utility gains in the long run. The superior agents in CPE continuously plan maneuvers for their subordinates which get executed by the lower rung nodes. We subjected the entire distributed community to random stresses by simulating enemy clusters of varying sizes and arrival rates. These stresses translated into the need to perform the distributed "sense-plan-respond" more frequently causing increased load and traffic in the network of agents. The stresses were created by a *world-agent* whose main purpose was to simulate warlike dynamics within our test-bed.

The CPE prototype consists of 14 agents spread across a physical layer of 6 CPUs. We utilized the prototype CPE framework to run 36 experiments at two stress levels ($S = 0.25$ and $S = 0.75$). There were three layers of hierarchy as shown in Figure 2a 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. To keep our queueing models simple, we assumed that the external arrival was Poisson and the service times were exponentially distributed. In order to cater to general arrival rates, the framework contains a QNA-based and a simulation-based model. Using this assumption a BCMP or M/G/1 queueing model could be selected by the framework for real-time performance estimation. The baseline for comparison was the *do nothing policy* (default) where we let the Cougaar infrastructure manage conditions of high load. Although our framework did better than any set of opmodes as shown in Figure 5 for the two stress modes, we show instantaneous and cumulative utility for two opmode sets ($Default\ A$ and $Default\ B$) in particular in Figure 6. We noticed that in the long run the framework did enhance the utility of the application as compared to the default policy.

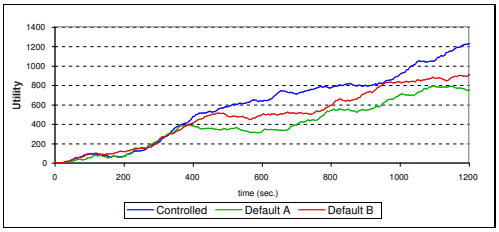
At both stress levels, the controlled scenario performed better that the default as shown in Figure 6. We did observe oscillations in the instantaneous utility and we attribute this to the impreciseness of the prediction of stresses. Stresses vary relatively fast in the order of seconds while the control granularity was of the order of minutes. Since this is a military engagement situation following no stress patterns, it is hard to cope with in the higher stress case. In contrast to MAS applications dealing with data centers where load can be attributed to time-of-day and other seasonal effects, it is not possible to get accurate load predictions for MAS applications simulating wartime loads. We think that this could be the reason why our utility falls in the latter case. In subsequent work, we intend to enhance Cougaar capability to be supportive of application-layer by forcing it to guarantee some end-to-end delay requirements.

6. Conclusions and Future Work

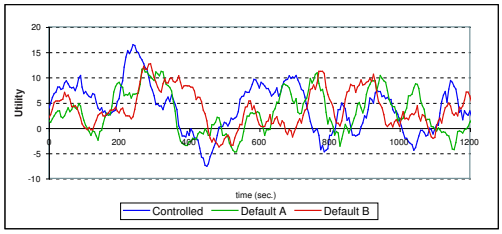
In this paper, we were able to successfully control a real-time DMAS to achieve overall better utility in the long run, thus making the application survivable. Utility improvements were made through application-level trade-offs between quality of service and performance. We utilized a queueing network based framework for performance analysis and subsequently used a learned utility model for computing the overall benefit to the DMAS (i.e. community). While Tesauro et al. [34] employ a resource arbiter to maximize the combined utility of several application environments in a data center scenario, we focus on using queueing theory to maximize the utility from performance of a single distributed application given that is has been allocated some resources. We think that the approaches are complementary, with this study providing empirical evidence to support the observation by Jennings and Wooldridge in [18]



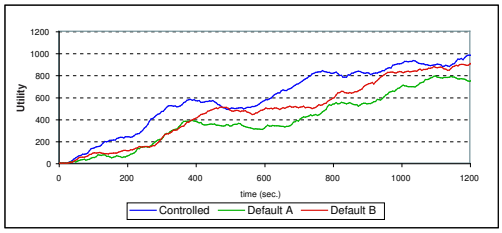
(a) Instantaneous Utility (stress 0.25)



(b) Cumulative Utility (stress 0.25)



(c) Instantaneous Utility (stress 0.75)



(d) Cumulative Utility (stress 0.75)

Figure 6. Sample Results

that agents can be used to optimize distributed application environments, including themselves, through flexible high-level (i.e. application-level) interactions.

Furthermore, this work has resulted in a general architectural lesson. We believe that any distributed application would have flows of traffic and would require service level attributes such as response times, utilization or delays of components to be optimized. The paradigm that we have chosen can capture such quantities and help evaluate choices that may lead to better application utility. This concept of breaking the application into flows and allowing a real-time model-based predictor to steer the system into regions of higher utility is pretty generic in nature.

We have kept the building-blocks small and the number of common interactions (between models) minimal since this may assist in making the framework more *scalable.* Distributed *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.

Acknowledgements

This work was performed under the DARPA UltraLog Grant#: MDA 972-01-1-0038. The authors wish to acknowledge DARPA for their generous support.

References

[1] Cougaar open source site. *http://www.cougaar.org*. DARPA.

[2] Ultralog program site. *http://dtsn.darpa.mil/ixo/*. DARPA.

[3] A. G. Barto, S. J. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[4] M. N. Bennani and D. A. Menasce. Assessing the robustness of self-managing computer systems under highly variable workloads. *International Conference on Autonomic Computing*, 2004.

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

[6] J. Bredin, D. Kotz, and D. Rus. Market-based resource control for mobile agents. *Autonomous Agents*, 1998.

[7] M. Brinn and M. Greaves. Leveraging agent properties to assure survivability of distributed multi-agent sys-

tems. *Proceedings of the Second Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.

[8] T. Chao, F. Shan, and S. X. Yang. Modeling and design monitor using layered control architecture. *Autonomous Agents and Multi-Agent Systems*, 2002.

[9] A. Chavaz, A. Moukas, and P. Maes. Challenger: A multi-agent systems for distributed resource allocation. *Agents*, 1997.

[10] L. Chen, K. Bechkoum, and G. Clapworthy. A logical approach to high-level agent control. *Agents*, 2001.

[11] A. E. Fallah-Seghrouchni, I. Degirmenciyan-Cartault, and F. Marc. Modelling, control and validation of multi-agent plans in dynamic context. *Autonomous Agents and Multi-Agent Systems*, 2004.

[12] D. Ferguson, Y. Yemini, and C. Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. *Proceedings of the International Conference on Distributed Systems*, 1988.

[13] I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. *Autonomous Agents and Multi-Agent Systems*, 2004.

[14] 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.

[15] A. Helsinger, K. Kleinmann, and M. Brinn. A framework to control emergent survivability of multi agent systems. *Autonomous Agents and Multi-Agent Systems*, 2004.

[16] A. Helsinger, R. Lazarus, W. Wright, and J. Zinky. Tools and techniques for performance measurement of large distributed multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2003.

[17] Y. Hong and S. R. T. Kumara. Coordinating control decisions of software agents for adaptation to dynamic environments. *The 37th CIRP International Seminar on Manufacturing Systems*, 2004.

[18] N. R. Jennings and M. Wooldridge. *Handbook of Agent Technology*, chapter Agent-Oriented Software Engineering. AAAI/MIT Press, 2000.

[19] 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.

[20] L. P. Kaelbling, M. L. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[21] K. Kleinmann, R. Lazarus, and R. Tomlinson. An infrastructure for adaptive control of multi-agent systems. *IEEE Conference on Knowledge-Intensive Multi-Agent Systems*, 2003.

[22] M. M. Kokar, K. Baclawski, and Y. A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, pages 37–45, May/June 1999.

[23] K. C. Lee, W. H. Mansfield, and A. P. Sheth. A framework for controlling cooperative agents. *IEEE Computer*, 1993.

[24] T. W. Malone, R. Fikes, K.R.Grant, and M.T.Howard. *Enterprise: A Market-like Task Scheduler for Distributed Computing Environments*. Elsevier, Holland, 1988.

[25] M. Morari and J. H. Lee. Model predictive control: past, present and future. *Computers and Chemical Engineering*, 23(4):667–682, 1999.

[26] A. Raja, V. Lesser, and T. Wagner. Toward robust agent control in open environments. *Agents*, 2000.

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

[28] J. B. Rawlings. Tutorial overview of model predictive control. *IEEE Control Systems*, 20(3):38–52, 2000.

[29] R. Sanz and K.-E. Arzen. Trends in software and control. *IEEE Control Systems Magazine*, June 2003.

[30] F. Sheikh, J. Rolia, P. Garg, S. Frolund, and A. Shepard. Performance evaluation of a large scale distributed application design. *World Congress on Systems Simulation*, 1997.

[31] I. Soto, M. Garijo, C. A. Iglesias, and M. Ramos. An agent architecture to fulfill real-time requirement. *Agents*, 2000.

[32] P. Stone and M. Veloso. Using decision tree confidence factors for multi-agent control. *Autonomous Agents*, 1998.

[33] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems*, 12(2):19–22, 1992.

[34] 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.

[35] H. P. Thadakamalla, U. N. Raghavan, S. R. T. Kumara, and R. Albert. Survivability of multi-agent supply networks: A topological perspective. *IEEE Intelligent Systems: Dependable Agent Systems*, 19(5):24–31, September/October 2004.

[36] R. Vincent, B. Horling, V. Lesser, and T. Wagner. Implementing soft real-time agent control. *Agents*, 2001.

[37] T. Vu, J. Go, G. Kaminka, M. Velosa, and B. Browning. Monad: A flexible architecture for multi-agent control. *Autonomous Agents and Multi-Agent Systems*, 2003.

[38] W. Whitt. The queueing network analyzer. *The Bell System Technical Journal*, 62(9):2779–2815, 1983.