Fluid models for evaluating threshold-based control policies for survivability of a distributed network

Vineet Aggarwal
ICF Consulting
9300 Lee Highway, Fairfax, VA 22031

Natarajan Gautam
Marcus Department of Industrial and Manufacturing Engineering
Penn State University
University Park, PA 16802

Abstract
The objective of this paper is to develop methods to improve the survivability of a distributed network of software agents. To achieve that, using thresholds for local information of the state of the system, each agent decides whether or not to accept an arriving task as well as whether or not to send a task to its CPU for processing. By adopting a control policy like this, we show using fluid-flow models that the survivability of the system is significantly better than the currently implemented case of accepting all arriving tasks and sending all tasks to the CPU.

Keywords
Survivability, distributed system, software agents, control policy, fluid models.

1. Introduction
Distributed systems of dynamic heterogeneous components without centralized control have emerged in a wide range of applications. The motivation for this research comes from a large scale distributed multi-agent system for military logistics called "Cognitive Agent Architecture" (COUGAAR; see http://www.cougaar.org). The COUGAAR system comprises of several software agents that work in a cooperative manner in order to accomplish tasks for military logistics planning operations. The set of software agents is known as a *society*. The overall objective is to build a survivable society of distributed autonomous agents whereby the system is robust as well as self-healing. The system must be able to dynamically adapt to the changes in its environment and reallocate essential processing, if necessary. It must be done in a distributed manner with no centralized control or human intervention. The key challenge is determining what individual strategies lead to the desired collective behavior.

We focus on a society where all the agents are distributed over different machines (or computers). At each machine, there is a single Central Processing Unit (CPU) that processes tasks submitted not only by the agent on the machine but also by other applications running on the machine. Based on the load (characterized by the pending tasks) and the resources (characterized by the CPU utilization) the agent can (1) accept or reject an incoming request for a task from the society, and (2) submit or refrain from submitting a task to the CPU. This is possible because each agent at the software level can use sensors to detect the load and resources and make decisions at the application level. This is significantly different from the vast literature that deals with task scheduling and management at the operating system (kernel) or hardware level.

As a motivating example, consider the Navy which has huge warships that are sent out on missions for long durations. These warships carry a large set of computing resources that support the missions apart from personnel and weapons that directly take part in warfare. One of the support functions could be logistics planning. These planning operations tend to be large scale and generally require a large chunk of the ship's computing resources for extended periods of time. Apart from planning, there are other regular activites that need computing resources for a short duration like communications. However, a ship being restricted by its size and other factors, cannot carry a dedicated set of computing resources for the planning operations and another set for other operations. It must share its available resources to serve both needs. This must be done efficiently and effectively keeping in mind the relative urgency for completion of different type of tasks. Change in planning objectives could result in change of work load characteristics. Therefore, the resource sharing mechanism must be robust to work effectively under the changing conditions. Any load balancing mechanism incorporated at the hardware level (like specially designed cluster computers) is not easily reconfigurable. They will work very effectively for a specific application but may be sub-optimal for another. An easily reconfigurable load balancing system is thus required to meet the needs of different

applications. We consider an approach where the load-balancing mechanism is implemented at the software level so that it can be easily reconfigured depending on the work load characteristics. Another notable difference is that with our strategy the agents act in a purely local fashion based on purely local information.

In this paper, a small piece of the large research program is addressed, in that we look at a single machine and study the impact of the control policies (such as accept or reject an incoming request for a task from the society, and submit or refrain from submitting a task to the CPU) on the machine's performance. We assume a workload that is bursty in nature – i.e. the tasks arrive at the node in short bursts. The bursty nature of the arrival process can be approximated better with a fluid model as opposed to a discrete model. The preference of replacing packet-level traffic in communication systems by fluid sources has been proposed by Anick et al [1], Elwalid and Mitra [3], Liu et al [4], etc. Fluid simulation of networks is gaining popularity as shown by Kesidis et al [5], Atkins and Chen [2], Kumaran and Mitra[7], etc.

The remainder of this paper is organized as follows. In Section 2, we talk about the scenario and the problem description in detail. Section 3 covers the fluid simulation model. Section 4 throws light on the results obtained from the simulation model. Finally, we state our concluding remarks in Section 5.

2. Problem Description

We consider a scenario of a CPU as a resource that receives tasks from a software agent as well as other applications. Assume there is one buffer for the agent and n buffers for other applications. All buffers have infinite capacity. Tasks arrive at the buffers in bursts. The CPU has a processing capacity of c bytes/sec. The CPU may serve multiple buffers in parallel by context switching between jobs. The time quantum between context switches is very small compared to the processing time. Therefore, we model assuming that the CPU processing capacity is equally divided between the buffers being served. We denote the agent buffer by $j = 0$ and the other n buffers as $j = 1,..,n$. Tasks submitted by the agent or applications are either served by the CPU immediately or get added to the respective queue if there are pending tasks in that buffer. Let $\{X_j(t),\ t \geq 0\}$ denote the number of tasks waiting in buffer j at time t. At any given time, several buffers may be served by the CPU. Let $\{m(t),\ t \geq 0\}$ denote the number of buffers being served by the CPU including the agent at time t. Let $A(t)$ be a binary variable such that

$$A(t) = \begin{cases} 1 \text{ if the agent buffer is being served by the CPU at time } t \geq 0 \\ 0 \text{ if the agent buffer is } \textit{not} \text{ being served by the CPU at time } t \geq 0. \end{cases}$$

The objective of this research is to compare and contrast two specific policies under various system conditions. One policy is the *basic policy* that exercises no control where the agent accepts all incoming tasks and uses the CPU whenever it needs to. The other policy is a threshold-based adaptive control policy (that we call *proposed policy*) that observes the state of the system (i.e. workload & CPU utilization) and takes a control action of (a) whether or not to accept an arriving task, and (b) whether or not to use the CPU. Since all the applications share the CPU, it may be better to relinquish the CPU from time to time to improve the overall system performance.

2.1 Basic Policy

In the basic policy, the agent exercises no control over the system. The jobs arrive and queue up at the respective buffers. The agent accepts all tasks into the buffer as they arrive. None of the tasks are rejected by the agent. The CPU serves all non-empty buffers $\{X_j(t) > 0,\ t \geq 0,\ j = 0,1..,n\}$ including the agent buffer at time $t \geq 0$. The CPU processing capacity 'c' is shared equally by the buffers being served at any given time. Therefore, at time $t \geq 0$, each non-empty buffer gets an output capacity equal to $c/m(t)$.

2.2 Proposed Policy

In the COUGAAR multi-agent architecture, the agents are "flexible" in that they can process other agents' tasks as well. The key behind the proposed policy is that when one agent is on a machine that needs to use the CPU to run non-agent jobs, it can reject its tasks and send it to agents that are associated with less loaded CPUs. In the proposed policy, the agent is given control over the following decisions (a) whether or not to accept an arriving task, and (b) whether or not to use the CPU. This decision must be taken using the locally available information viz. the current work load in the agent buffer and the current workload due to other applications (non-agent traffic) at the node. The Non-agent workload can be defined as the number of non-agent buffers that are being served (or are non-empty) at time t. Let $N_j(t)$ represent whether non-agent buffer j ($j = 1,\ldots,n$) is empty at time t by

$$N_j(t) = \begin{cases} 1 \text{ if } \{ X_j(t) > 0,\ t \geq 0,\ 1 \leq j \leq n \} \\ 0 \text{ if } \{ X_j(t) = 0,\ t \geq 0,\ 1 \leq j \leq n \} \end{cases}$$

Let us denote *non-agent workload* by $U(t) = N_1(t) + \ldots + N_n(t)$. As per this policy, if the Non-agent workload reaches or exceeds a pre-defined load-limit L i.e. $U(t) \geq L$, the agent does not submit jobs to the CPU. The agent starts submitting the jobs again if $U(t) < L$. Another control point is the rejection of arriving tasks. Once the queue in the agent buffer reaches a threshold value B, the agent rejects further arriving jobs until the queue length drops below B

again. Let $C(t)$ represent the number of non-agent buffers that are being served when the agent buffer is also being served. We will call this as *CPU Usage*. Therefore, $C(t) = U(t)\ A(t)$. Let $R(t)$ represent the number of jobs being rejected at time t by the agent.

2.3 Performance Measures

We compare the performance of the two policies using the following measures:

- The average number of pending tasks for the agent buffer $E(X_0(\infty))$ *bytes/sec*
- The average CPU usage $E(C(\infty))$ (Average # of non-agent buffers that are non-empty per unit time when the agent is being served).
- The average rejection by the agent $E(R(\infty))$ *bytes/sec* in the proposed policy.

An accurate comparison between the two policies can only be done when the combined effect of the changes in the above three measures is studied. For this purpose, we assign appropriate weights to each of the above factors and add its effects. Let w_x, w_u and w_r represent the weights for average number of tasks pending in the agent buffer, average CPU usage and average rejection respectively. The term w_x can be thought of as the average cost of holding tasks (or average inventory cost) in the agent buffer per unit time, w_u can be thought as the cost that the agent must pay to use the CPU per unit time depending on the already existing non-agent workload on the CPU ($C(t)$) and w_r can be thought of as the cost per byte/unit time of task rejected. Therefore, the long run average cost per unit time of the given system can be computed as

$$Average\ Cost = w_x\, E(X_0(\infty)) + w_u\, E(C(\infty)) + w_r\, E(R(\infty))$$

Our objective thus becomes Minimize (*Average Cost*) since we would ideally like to reduce the average number of pending tasks in the system, the CPU usage cost and at the same time keep rejection rate to a minimum. Although we assume given, these weights are assigned by a central authority to ensure survivability in a cooperative manner.

3. Simulation

One of the key characteristics of the system we are considering is that both the agent as well as the non-agent tasks arrive in a bursty manner i.e. at each buffer, a bunch of tasks arrive in quick succession and then there no arrivals for some duration of time. Therefore instead of developing a detailed packet-level simulator, we take advantage of the on-off fluid-like traffic behavior and build a higher-level fluid simulator. Thereby we compare the performance of the two policies in section 2, using the fluid simulation model. This section briefly describes the model. Further details can be found in [8]. The simulation model for the "basic policy" is not explained separately since the basic policy can be considered to be a special case of the proposed policy with $B = \infty$ and $L = \infty$.

We model the scenario as an $n + 1$ buffer fluid-flow system as shown in Fig. 1. All buffers are of infinite capacity. Fluid enters buffer j according to an alternating on-off process such that, for an exponentially distributed time (with mean $1/\alpha_j$ sec) fluid enters continuously at rate r_j bytes/sec and then no fluid enters for another exponentially distributed time (with mean $1/\beta_j$ sec). The long-run average arrival rate of traffic from source j is $r_j\beta_j/(\alpha_j+\beta_j)$. Without loss of generality assume that $r_j > c$ (for $j = 0, 1,..,n$). The condition for system stability is that the average task arrival rate should be less than the CPU processing capacity c bytes/sec [6]. Therefore, $\sum_{j=0}^{n} \frac{r_j \beta_j}{\alpha_j + \beta_j} < c$

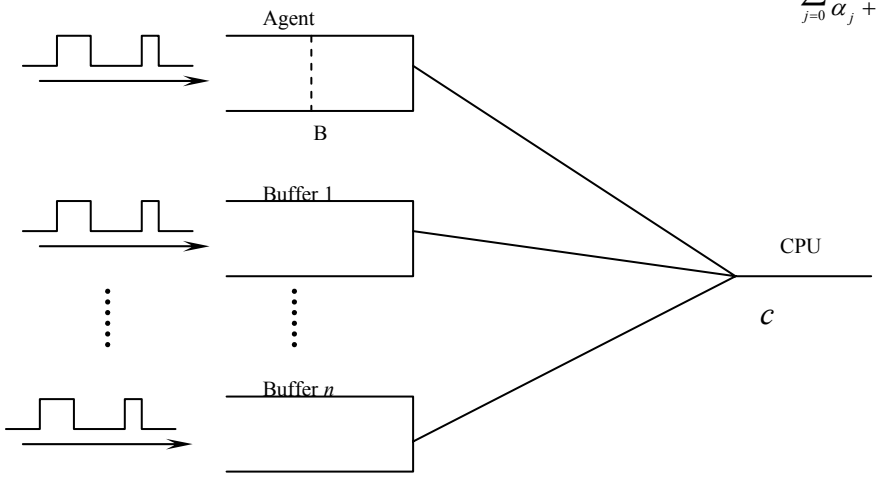

Fig. 1: One Agent and n non-agent Application

The various variables present in the simulation model can be divided into the following three categories:
Policy Variables: These are: (a) the load limit L and (b) the threshold value B for the agent buffer at which the agent will reject incoming tasks.
System Variables: These are: (a) ratio of agent load to non-agent load r_a/r_q where r_a = average arrival rate of fluid in agent buffer = $r_0\beta_0/(\alpha_0+\beta_0)$ and r_q = average workload due to non-agent fluid = $\sum_{j=1}^{n} \frac{r_j \beta_j}{\alpha_j + \beta_j}$; (b) $(\frac{r_a + r_q}{c})$, ratio of total load to output capacity; (c) burstiness of agent source β_0/α_0.
Response Variables: These are: (a) the average number of pending tasks for the agent buffer E$(X_0(\infty))$ *bytes/sec*; (b) the average CPU usage E$(C(\infty))$; (c) the average rejection by the agent E$(R(\infty))$ *bytes/sec* in the proposed policy.

4. Results

In this section, we compare the performance measures described in Section 2.3 with respect to the two policies (basic and proposed) under various system conditions and thereby present numerical results. Our approach is to set some base values (see Table 1) for the various variables described above, and then perturb them to study their effect.

Simulation Time	T	200000	sec
Agent Alpha	α_0	2	MHz
Agent Beta	β_0	1	MHz
Agent Burstiness	β_0/α_0	0.5	
Non-Agent Alpha	α_j	60	MHz
Non-agent Beta	βj	3	MHz
Peak Arrival Rate for Agent	r_0	6	bytes/sec
Peak Arrival Rate for Non-agent	r_j	10.5	bytes/sec
Avg. Agent Arrival Rate	r_a	2	bytes/sec
Avg. Non-agent Arrival Rate	rj	0.5	bytes/sec
Avg. Individual Non-agent Arrival Rate	r_q	2	bytes/sec
Ratio of Agent Load to Other Load	r_a/r_q	1	
Output Rate	c	4.211	bytes/sec
Ratio of Total Load to Output	r_t/c	0.95	
Buffer Threshold Value of Agent	B	20	bytes
Load Limit	L	2	

Table 1: Base Values of System Variables

4.1 Comparison of the two policies for different values of the policy variables B and L

We vary the value of L from 5 to 1 and B from 30 to 0 and keep every other system variable as per the base values given in Table 1. Under the proposed policy, for the same threshold value B, the average agent queue length increases only marginally as the load limit decreases. But as the threshold value B increases from 0 to 30, the average agent Q length increases considerably for all values of load limit L, as seen in fig. 2(a). The average agent Q length under the basic policy is 6.632. In fact, for any threshold limit $B < \infty$, the average agent Q length will always be less than that of the basic system. With increase in threshold value $B > 0$, the avg. CPU usage increases (fig 2(b)). This is expected as the agent buffer can submit jobs for longer durations because of increased storage. During this time, more non-agent queues have a chance of being non-empty and thus the average CPU usage $C(t)$ increases. The exception to this rule is the case where $B = 0$. Since no jobs are stored the agent can only submit the jobs that arrive when $U(t) < L$. When $U(t) \geq L$, all incoming jobs are rejected. Therefore, $C(t)$ will always take values either equal to or less than $L-1$. Average CPU usage under the basic system is only 1.402. The rejection rate is primarily a function of the threshold value B. The lower the threshold value, the higher will be the rejection rate (fig. 2(c)). For a given threshold value, the rejection rate does not vary greatly with change in value of load limit L. We show in fig 2(d) by comparing costs for the system under proposed policy at different values of B and L. We assume that the value of cost weights are w_x = 0.4 \$/*bytes/sec*, w_u = 0.4 \$/*bytes/sec* and w_r = 0.2 \$/*bytes/sec*. The cost for the basic system with the same cost weights is \$ 2.9328 /sec. As in fig. 2(d), there is a substantial savings under the proposed policy as compared to the basic policy. As the value of B decreases towards zero, the total cost of the system decreases

considerably. As the Load Limit is decreased from 5 to 1, there is a further reduction in total cost. The total cost was lower than that of the basic cost under all combinations of *B* and *L*.

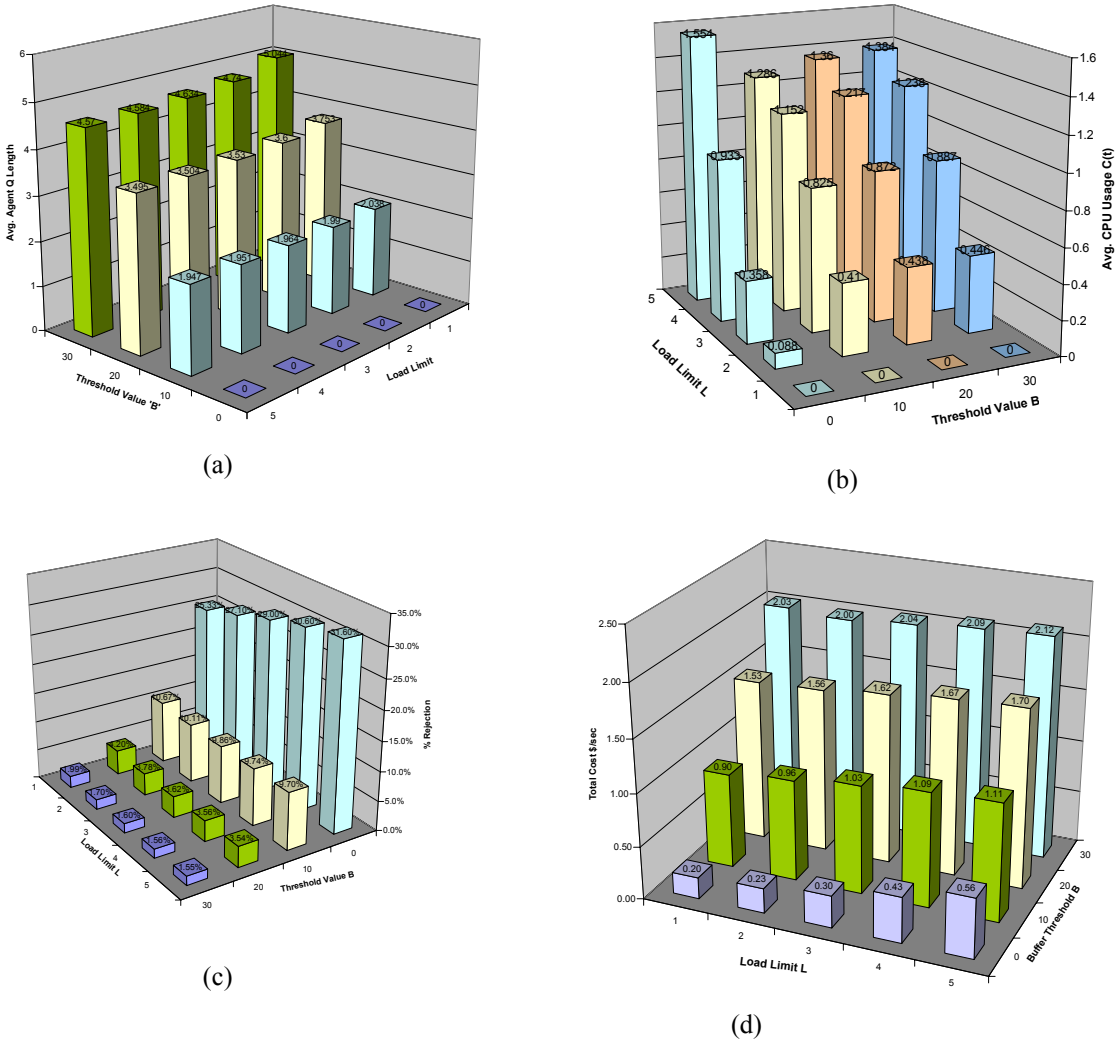

Fig. 2: Performance versus varying *B* and *L*

4.2 Comparison between the two systems at different values of Load Factor

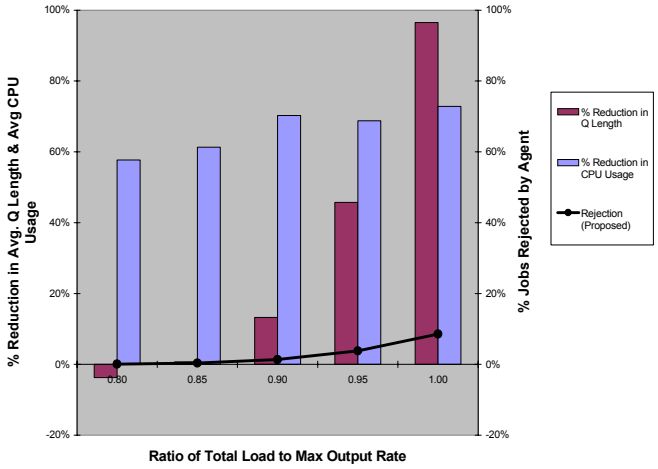

Fig. 3: Impact of Proposed Policy due to Changing Total Load

The load factor $(r_a+r_q)/c$ is varied using the values of 0.8, 0.85, 0.90, 0.95 and 1 by modifying *c* suitably. The behavior of the two systems is studied and a comparison is drawn. As expected, the average agent Q length increases with increasing load factor under the basic policy but it will grow at a slower rate under the proposed policy due to the maximum limit of *B*. Thus, we see a greater reduction in average agent Q length at high load factors. As the load factor increases, the average CPU usage cost is greater. Also, the rate of rejection increases with increase in load factor. Thus, this policy would behave better for systems with high load factor if the rejection cost is lower or even comparable to the inventory holding cost and CPU usage cost.

5. Conclusions

We simulated a system where a CPU has to be shared by an agent to complete its tasks as well as other non-agent local applications that need the processing capacity of the CPU. This system was modeled with *1* agent buffer and *n* non-agent buffers that receive tasks from the respective applications. Two different policies were studied (a) *Basic Policy* in which the agent exercises no control and submits tasks to the CPU as they arrive and (b) *Proposed Policy* in which the agent can make decisions based on the information it has. Using a threshold policy, the agent decides (i) whether or not to accept an arriving task, and (ii) whether or not to use the CPU. The system behaviour under the two policies were studied and contrasted using a fluid-simulation model. We showed that under various system conditions (many of which are detailed in [8]), the *proposed* policy far outperforms the *basic* policy thereby indicating that by appropriately controlling the agent, the survivability of the architecture can be improved.

One of the main requirements objectives of the COUGAAR architecture was that the system must be able to dynamically adapt to the changes in its environment and reallocate essential processing if necessary. It must be done in a distributed manner with no centralized control. The proposed policy is a step in that direction. Under this policy, the agent takes decisions dynamically based on the state of the system. The decisions are taken with minimal and local information thus avoiding excessive information processing overheads. During periods of high non-agent workload, the agent holds back its tasks, waits for the CPU workload to reduce, and at the same time defers new arriving tasks to other agents with low workload through rejection. Thus, it improves the CPU utilization by spreading workload across the different resources managed by agent society and also over different periods of time.

Acknowledgements

This work was partially supported by DARPA (Grant: MDA 972-01-1-0038).

References

[1] D. Anick, D. Mitra and M.M. Sodhi. *Stochastic Theory of a Data Handling System with Multiple Sources*. Bell System Tech. Journal, 61, 1871-1894, 1982.

[2] D. Atkins and H.Chen. *Perfromance Evaluation of Schedulig Control of Queueing Networks: Fluid model Heuristic*. Queueing Systems, 1995

[3] A.I. Elwalid and D. Mitra. *Analysis and Design of Rate-Based Congestion Control of High Speed Networks, part I: Stochastic Fluid Models, Access Regulation*. Queueing Systems, Theory and Applications, Vol.9, 29-64, 1991.

[4] B. Liu, Y. Guo, J. Kurose, Towsley, W. Gong. *Fluid Simulation of Large Scale Networks: Issues and Tradeoffs*. In Proc. of PDPTA'99, Vol IV, pages 2136-2142, June 1999.

[5] G. Kesidis, A. Singh, D. Cheung, and Kwok, *Feasibility of Fluid Event-Drive Simulation for ATM Networks*. In Proceedings of IEEE Globecomm, volume 3, pages 2013-2017 , Nov. 1996.

[6] V. Kulkarni, and T. Rolski, *Fluid Model Driven by an Ornstein-Uhlenbeck P rocess*. Prob. Eng. Inf. Sci. 8, 403 - 417, 1994

[7] K. Kumaran and D. Mitra. *Performance and Fluid Simulation of Novel Shared Buffer Management System*. In Proceedings of IEEE INFOCOM 98, March 1998.

[8] V. Aggarwal. *Efficient Allocation of Distributed Resources and Loads in a Multi-Agent System*, M.S. Thesis, Industrial Engineering, Penn State University, 2002.