# Multiagent coordination for Multiple Resource Job Scheduling

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

John Lawson
NASA Ames Research Center
lawson@email.arc.nasa.gov

## ABSTRACT

Efficient management of large-scale job processing systems is a challenging problem, particularly in the presence of multi-users and dynamically changing system conditions. In addition, many real world systems require the processing of multi-resource jobs where centralized coordination may be difficult. Most conventional algorithms, such as load balancing, are designed for centralized, single resource problems. Indeed, in such a case, load balancing is known to provide optimal solutions. However, load balancing is not well suited to the more general, distributed, multi-resource allocation problem across heterogeneous networks that is frequently encountered in real world applications. Approaches based on heuristics can be designed to handle multi-resource allocation, but such approaches do not necessarily attempt to optimize *directly* a system-wide objective function. In this paper, we investigate a multiagent coordination approach to distributed, multi-resource job scheduling across heterogeneous servers. In this approach, agents at servers make local decisions to optimize an agent specific objective. The agent objectives though, are derived so that they are aligned with the overall efficiency of the system. We demonstrate that such a system outperforms (sometimes dramatically) more crudely constructed multiagent systems as well as a multi-resource version of load balancing.

## Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence

## General Terms

Algorithms, Performance

## Keywords

Multiagent Coordination, Multiagent Learning, Job Scheduling

## 1. INTRODUCTION

With the ever increasing connectivity between servers, networked or grid computing is becoming a natural alternative to either dedicated homogeneous server grids or supercomputers for processing large numbers of jobs with varying priorities and resource requirements. However, managing a large, distributed data and job processing system capable of handling multiple resource requirements is a challenging problem, in that many difficulties need to be simultaneously addressed. In the presence of heterogeneous servers (e.g., processor speed, memory), jobs with multiple resource requirements (e.g., data access, memory) dynamic environments (e.g., job arrivals do not follow a static distribution) and the presence of disturbances in the system (e.g., failing servers or links) most algorithms designed for a single resource allocation algorithm either do not apply or fail to provide good solutions.

Indeed, though the single-resource case has been extensively, studied [33], the multi-resource job scheduling across a network of heterogeneous servers has received much less attention [27].[1] In addition, because of the natural distributed nature of such system, approaches based on centralized control are often inappropriate. Such methods, provide rigid, inefficient solutions, and in most cases have communication and synchronization requirements that offset any of the benefits of using a grid based system.

Load balancing is a centralized algorithm that has been successfully applied to single resource scheduling problems. In fact, for single resource optimization problems, there are theoretical results showing that load balancing does provide optimal solutions [33]. Generalizing load balancing to the multi-resource case, though, is far from straightforward. In its simplest form, multi-resource load balancing aims at ensuring that the level of activity on each server stays the same, i.e., the load on the system is balanced across all the servers. This approach to load balancing *assumes* that the load being distributed across the servers is a de-facto desirable solution, i.e. that it optimizes some pre-specified global objective. In the multi-resource case, this assumption is no longer valid, and one needs to determine which resource (or which combination of resources) needs to be "balanced". In fact, different extensions of load balancing to the multi-resource case leads to the optimization of different functions [27], and thus there are no guarantees that balancing a particular combination of the resources will lead to the optimization of the global objective. A further limiting feature of load balancing is that it requires centralized con-

[1]Throughout this paper, we refer to servers with different resource configurations as "heterogeneous" servers. We assume that there are no compatibility issues related to compilation of the jobs, and that any job can be executed at any server, assuming the server has the necessary resources. In some articles [27], this type of network is referred to as a "near-homogeneous" computational grid.

**Cite as:** Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

trol, and though heuristics exist to overcome limitation for the single-resource case, the performance of such algorithms suffers greatly in the multi-resource case [27].

Multiagent learning methods are ideally suited to handle the challenges presented by such problems. In particular, agents based on reinforcement learning [5, 7, 24, 33, 34, 36] offer adaptive and flexible solutions that sidestep the potential mismatch between balancing a "load" across the network and optimizing the global objective function. Indeed, the agent based approach we propose aims to optimize the global objective without directly aiming to balance the load across the servers. It is entirely possible that good solutions to a multi-resource job scheduling across a heterogeneous grid problem reside in states where some servers are idle while others are operating at full capacity and have full queues. As long as that system behavior is considered good in terms of the global objective, no consideration should be made to "split", or balance the load.

Because of its direct aim at optimizing an objective function, agent-based methods address the limitations of load balancing. As such methods based on creating a currency [31], bio-inspired swarms [14, 44], mechanism design [9] and coordination [4] have been proposed, as have other innovative methods [11, 18, 21, 26, 19, 15] However, they introduce a new difficulty: how to ensure that the actions of multiple agents lead to a good global solution. To address this coordination issue, we need to ensure that the objective function of each agent is designed in a manner that promotes two properties. First, an action that improves the agent's objective function should also improve the global objective. Second an agent needs to clearly see the impact of its actions on its own objective function [1, 2, 3, 42]. An agent based solution to grid computing where the agent objective are set according to these two criteria offers the best compromise between a rigid centralized solution and a distributed solution where the interaction among the agents can have deleterious side effects on system behavior.

In this paper, we present an agent-based solution to the multi-resource optimization problem in heterogeneous network that outperforms both the multi-resource version of load balancing (by up to four times), and a "naive" multiagent system in which all the agents attempt to directly optimize the system objective. *The key contribution of this paper is in providing local objective functions for the agents (components of a server) in a manner that allows them to adapt locally, while ensuring that their achieving their local objectives improves global performance.* In Section 2 we present the system model and discuss the system dynamics of the multi-resource job scheduling across a heterogeneous grid. In In Section 3, we derive the agent based algorithm and present a multi-resource load balancing algorithm, along with a simple performance bound. In Section 5, we show simulation results where the multiagent system approach significantly outperforms multi-resource load balancing. Finally, in Section 6 we discuss these results and highlight future directions of research.

# 2. MULTI-RESOURCE OPTIMIZATION

With demand for computing resources increasing as both the number of users and the complexity of the applications increase, the ability of a system to efficiently schedule and process jobs is becoming increasingly important. As such, heterogeneous computational grids where jobs can enter the network from any point and be processed at any point are becoming increasingly popular. Below, we describe a model for such a computational grid and show how an agent-based approach can be implemented.

## 2.1 System Model

The computational grid model we use consists of a network of $N$ servers each with $K$ resources $(r_1, ...r_k)$. Each server has a specified capacity for each resource assigned to be an integer ranging from $[1, M]$. Thus, $M$ measures the heterogeneity of the resources. For example, the first resource $r_1$ can correspond to the processing speed of the server. In our configurations, on average, each server has 2-4 neighbors with which it has a direct connection.

Each job entering the system is also specified by $K$ resource requirements ranging from $[1, M]$. For example, the first job resource $r_1$ is an indication of the number of cycles the job requires to be processed. In this formulation, for each resource $r_i, i > 1$, the server resource capacity must be equal or greater than the job's requirement in order for a job to run on a particular server. Intuitively, this corresponds to the requirement that a server must have enough memory to accommodate a given job.

## 2.2 System Dynamics

In this model, each server has its own wait queue for jobs. For simplicity, we allow only one job to run on a server at a time; the other jobs remain in the queue until the processor becomes available. Jobs enter the local queues either externally (to the system) or are shipped from other servers. Jobs entering externally are sent to the back of the queue while jobs received from other queues go to the front. There are two reasons why shipped jobs go to the front: First, it provides a measure of "fairness" as those jobs already had to wait in the queue of the server in which they were originally placed. Second it provides efficiency, as it forces the system to deal with "difficult" jobs (either run them or ship them if they could not be run). This approach prevents these jobs from being endlessly shuffled. At each server, the first job in the queue is activated if the processor is available, and the resource requirements are met. If the processor is available, but the server does not have the resource capacity to run the job, the server remains idle until the problem job is sent to another server.

The dynamics of our simulations thus proceed as follows. At each time step $\tau$, a random number of new jobs are added to the wait queue of each server. In particular, each server has a probability of receiving a new job at each time. If a given processor is idle, and the first job in the queue meets the resource requirements, that job is activated. If not, the server remains idle. In addition, for each $\tau$, the server makes a decision about the first job in the queue, deciding whether to keep the job or send it to a neighboring server. These decisions are made based on the agents' probability vectors which in turn are set using a basic learning algorithm (discussed in more detail in Section 3.3).

Thus, there are two main sources of inefficiency in the system. The first are the bottlenecks created by jobs whose requirements exceed the capacity of their server. When such a job get to the front of the queue, the server remains idle until the job is shipped to a neighbor. The second source of inefficiency arises from mismatches between a processor's speed and a job's cycle requirement.

## 3. MULTIAGENT ARCHITECTURE

There are many possible ways to map the multi-job scheduling problem onto a multiagent system, including simply assigning an agent to each server and letting those agents' actions be determining where to send a particular job. Instead, in this work, we explore the mapping where there are multiple agents at each server. This results in a system with more agents with a relatively easier learning problem, rather than fewer agents with a more difficult learning problem. In fact, this choice shifts the burden from a pure learning problem where the details of the agents' algorithms are the key to the coordination problem to *how the agents interact* with one another. In particular, to each agent, we assign a vector $\vec{p}$ whose components give the probability of routing a job to its various neighbors. In this scenario, the agents are given the task of setting their own probability vector. The design question consists of determining what objective function each agent should attempt to optimize so that they set the probability vectors that also optimize the overall job processing efficiency of the full system.

The resource specifications of a job determined which agent at the server is responsible for the shipping decision. In this work, we focus on the job partitioning where for jobs with $K$ resources, $2^K$ agents can be assigned per server where agent 1 deals with jobs such that $r_1 \in [1, M/2], ..., r_k \in [1, M/2]$, agent 2 deals with jobs $r_1 \in [M/2+1, M], r_2 \in [1, M/2]$ $, \cdots, r_k \in [1, M/2]$, etc. This approach can be directly applied in systems with a small number of resources (e.g., three for processing speed, memory requirement and disk access), and this is the method we use in this paper. If the number of resources to manage becomes large, then resources can be clustered together as appropriate. This can be achieved either by direct design or by having agents form teams based on the correlations of those resources.

For the dynamics governing the system evolution, we will distinguish between two time scales : $\tau$ gives the time steps at which the system operates (e.g., jobs enter the system, move between queues, and are processed) whereas $t$ gives the time steps at which the agents operate (e.g., observe their objectives, change their actions). This distinction is important because it is the only way by which an agent can get a "signal" from the system that reflects the impact of its decision, i.e, the system has to settle down before an objective can be matched to an action. Therefore, an agent $i$ changes its probability vector at each time $t$. Within a "single agent time step" $t$ though, many jobs enter the system, are executed, routed etc. each of which occurs at time interval $\tau$ ($t >> \tau$).

### 3.1 State Space and Global Objective

Let us define the state of each agent $i$ at time $t$ as by

$$z_{i,t} = \{(j_0, w_0, I_0^{i,t}, e_0^{i,t}), \cdots, (j_k, w_k, I_k^{i,t}, e_k^{i,t}), \cdots\} \quad (1)$$

where $j_k$ identifies job $k$, $w_k$ is the weight of that job which gives the importance of that job in the system, $I_k^{i,t}$ is the "job indicator" function and is equal to 1 if job $k$ was handled (received, shipped or executed) by agent $i$ at time step $t$, and 0 otherwise, and $e_k^{i,t}$ determines whether job $k$ was executed at agent $i$ at time step $t$.

Now, the state of the full system, $z_t$ at time $t$, is given by:

$$z_t = \{(j_0, w_0, 1, e_0^t), \cdots, (j_k, w_k, 1, e_k^t), \cdots\} \quad (2)$$

where $e_k^t$ determines whether job $k$ was executed at time step $t$. Note that the job indicator function $I_k^t$ is always set at 1 for the full system, since by definition, if the job is in the system, it must have been handled by at least one agent. Nevertheless, we keep the notation, both for ensuring consistency between the state vector of an agent and that of the full system, and because its presence in the global objective will facilitate the derivation of the agents' objectives.

Based on this, the global objective at time $t$ is given by:

$$G(z_t) = \frac{\sum_k w_k.e_k^t}{\sum_k w_k} \quad (3)$$

Intuitively, $G$ gives the weighted ratio of all the jobs that were processed at time step t to all jobs that entered the system at that time step (recall that "time step $t$ is a window of time, not a single time step from the point of view of the jobs".)

### 3.2 Agent Objectives

In this work we investigated three different types of agent objectives. Each was used exactly in the same manner with the same learning algorithms. Hence, the only difference in system performance is based on the objective the agents were trying to optimize.

- The first agent objective was the global objective given in Equation 3. This objective allowed each agent to directly attempt to optimize the full system objective directly. By definition, this objective guarantees that if all agents succeed in optimizing their own objectives, the system objective will also be optimized. However, because in large systems each agents objective will depend on the actions of other agents, in practice this objective function only provides good solutions for very small systems [2, 13, 40].

- The second agent objective was the difference objective discussed which aims to isolate the impact of an agent on the system [3, 2, 39, 40, 42]. This is achieved by computing the difference between the system objective and the system objective that would result if agent $i$ were removed from the system. An agent can be "removed" from the system by setting $I_k^{i,t}$ to 0 for all jobs $k$ for which it was set to 1 at time step $t$. The difference objective ($D$) for agent $i$ is given by:

$$\begin{aligned} D_i &= G(z) - G(z_{-i}) \\ &= \frac{\sum_k w_k.e_k^t}{\sum_k w_k^t} - \frac{\sum_k w_k.e_k^t.\bar{I}_k^{i,t}}{\sum_k w_k^t} \\ &= \frac{\sum_k w_k.e_k^t.I_k^{i,t}}{\sum_k w_k^t} \end{aligned} \quad (4)$$

  where $\bar{I}_k^{i,t}$ is the complement of $I_k^{i,t}$ and equals 1 when $I_k^{i,t}$ equals 0 and 0 when $I_k^{i,t}$ equals 1. Intuitively, $D_i$ represents the weighted fraction of jobs that were handled by agent $i$ to the jobs that entered the system.

- The third agent objective was the "Selfish" objective, where the agents were only concerned with processing jobs that were assigned to them. The selfish objective ($S$) for agent $i$ is given by:

$$\begin{aligned} S_i &= G(z_i) \\ &= \frac{\sum_k w_k.e_k^t.I_k^{i,t}}{\sum_k w_k^t.I_k^{i,t}} \end{aligned} \quad (5)$$

Intuitively, $S$ gives the ratio of the jobs processed by the system at time step $t$, to the total jobs that passed through that agent, hence the indicator function in the denominator.

Notice that both $D$ and $S$ are specifically tuned to the performance of a particular agent, their form is significantly different. $D$ attempts to measure the impact of agent $i$ on the system, whereas $S$ attempts to measure the efficiency of agent $i$ directly, without attempting to measures its effect on the full system. Systems using both $D$ and $S$ are highly sensitive to the actions of the agent, and $D$ is much more aligned with the system objective than $S$ is. Similarly, though both $G$ and $D$ are aligned with the system objective (tautologically for $G$), an agent using $D$ will have an easier time seeing the impact of their actions on their objective functions. Note that regardless of which objective function the agents use, the system performance is always measured by the global objective given in Eq 3.

## 3.3 Agent Learning

As discussed above, the agent learning took place at a higher time scale than the system operated. For a given time step $t$, each agent followed a fixed policy (e. g., the probability vectors were fixed). During that time step $t$, the system operated at $\tau$ intervals (for these experiments $t = 400\tau$). At the end of time step $t$, the objective functions were calculated and recorded in the agents' training sets. In order to be able to compare the performance individual probability vectors, we cleared the system (i.e. the queues) after each $t$. During the initial phase, $0 \leq t \leq 100$, the probability vectors were set at random. After this "data collection" phase, $t \geq 100$, the agents utilized a basic learning algorithm to set their probability vectors as described below.

The learning algorithm proceeded by first generating a set of candidate probability vectors with a Gaussian distribution about the current probability vector. Expected objective function values were estimated by performing a weighted average over objective function values from the agents' training set. The objective values were weighted by both how long ago the value was recorded (data aging) and the distance between the candidate and the previous probability vector.

$$V(\vec{p}_T) = \frac{\sum_t F_t \; e^{-\alpha_T(T-t)} \; e^{-\alpha_P||\vec{p}_T - \vec{p}_t||}}{\sum_t \; e^{-\alpha_T(T-t)} \; e^{-\alpha_P||\vec{p}_T - \vec{p}_t||}}. \qquad (6)$$

Here, T is the current learning period, $t$ is the period which resulted in objective value $F_t$, $\vec{p}_T$ is the current probability vector, $\vec{p}_t$ is the vector that resulted in objective value $F_t$, and $\alpha_P$ and $\alpha_T$ are system parameters. Depending on the agent objective chosen, $F_t$ is given by $G$, $D_i$, or $S_i$ as discussed above. The new probability vector is then chosen by sampling a Boltzmann probability distribution over the estimated values $V(\vec{p}_T)$. This process allows for good exploration of the probability space, while ensuring that the most recent probability vectors have more relevance in that they are more likely to provide solutions tuned to the current conditions.

# 4. MULTI-RESOURCE LOAD BALANCING ALGORITHM

In addition to the agent-based methods introduced in this article, we also investigated the feasibility of a distributed, deterministic, multi-resource load balancing algorithm. For each server, we calculated a load for each of the $k$ resources, $l_k = \Sigma_n^N (s_k^n / c_k)$ where $s_k^n$ is the need of resource $k$ of job $n$ and $c_k$ is the the capacity of resource $k$ of the server. Thus, the resource load has been normalized to the resource capacity of the server. We assign a load to a particular server $i$ as the average of its individual resource loads $L_i = Avg(l_k)$. We, then, calculate the system load as the average over the servers $L_{avg} = Avg(L_i)$.

The load balancing algorithm proceeds as follows. At each time step $\tau$, each server calculates its own load and compares it with the global load $L_{avg}$. If the server's load is greater than the global, modulo some tolerance, the server looks to get rid of its highest load job. Each server has access to global information about the loads on all the other servers. Using this information, the server determines which of the other servers has the lowest load. It then ships its high load job to the low load server via the one of its neighbors that lies on the shortest path between the sending and the receiving servers.

# 5. EXPERIMENTAL RESULTS

We ran extensive simulations that tested the performance of the algorithms in a variety of settings. All the results reported here were on networks of $N = 50$ servers having $K = 2$ and $K = 4$ resources. The 50 servers had 4 or 16 agents respectively, making for 200 to 800 total agents in the system. The servers were connected into a network having a ring configuration with random connections added in the spirit of "small world" networks [32, 41]. In general, each server had 2-4 neighbors with which it had a direct connection.

We examined the performance for different number of resources $K$, job arrival probabilities $r$ and different resource ranges $M$. We tabulated the performance for the multiagent approach with learning agents, a load balancing algorithm generalized for the multi-resource case, and a random shipping algorithm RAND. In the RAND algorithm, the proportion vectors for shipping/holding the first job in the queue was set randomly. This is the situation when the agents are in the training phase of their learning algorithm. For scenarios involving learning agents, we performed experiments using agent objectives based on global ($G$), selfish ($S$), and the difference ($D$) objectives.

The experiments can be grouped into three categories:[2]

- Low Difficulty: There are two resources, each having two types and the jobs enter the system at a slow pace. All three key parameters have "low" settings: ($K = 2$, $r = .2$, $M = 2$).

- Medium Difficulty: One of the parameters is set to a "high" setting. For example, there are four resources, **or** there are eight settings for each resource, **or** jobs have a high arrival rate. This covers the following pa-

[2]The case where all three parameters are set to "high" results in an impossible problem in that the jobs entering the system have a high probability of not running on most of the machines. Because jobs are coming into the system at a high rate, the system never has a chance to move those jobs before they "clog" the system, leading to a situation where jobs are entering the system faster than the system can process them. This situation leads to poor performance by all the algorithms as the problem is in essence unsolvable.

rameter combinations: ($K = 4$, $r = .2$, $M = 2$) ; ($K = 2$, $r = .8$, $M = 2$) ; ($K = 2$, $r = .2$, $M = 8$) .

- High Difficulty: Two of the parameters are set to "high" values. For example, there may be both four resources **and** eight types for each, or four resources **and** a high rate of job arrival). This covers the following parameter combinations: ($K = 4$, $r = .2$, $M = 2$) ; ($K = 2$, $r = .8$, $M = 2$) ; ($K = 2$, $r = .2$, $M = 8$) .

In the following subsections, we present the results of all the algorithms for the three cases outlined above. The results show the algorithm performance at the end of the runs (t=400) and are averaged over 50 different randomly generated network configurations. The best performance in each case is noted in bold when the differences in the mean ($\frac{\sigma}{\sqrt{N}}$ for N runs with standard deviation $\sigma$) are statistically significant.

## 5.1 Low Difficulty Parameters

**Table 1: System Processing Efficiency for Easy Setting**

| K | r | M | Algorithm | Global Objective | $\sigma$ |
|---|---|---|---|---|---|
| | | | RAND | 0.9318 | - |
| 2 | .2 | 2 | S | 0.976 | 0.0039 |
| | | | G | 0.947 | 0.0052 |
| | | | D | 0.979 | 0.0023 |
| | | | LB | **0.997** | 0.00083 |

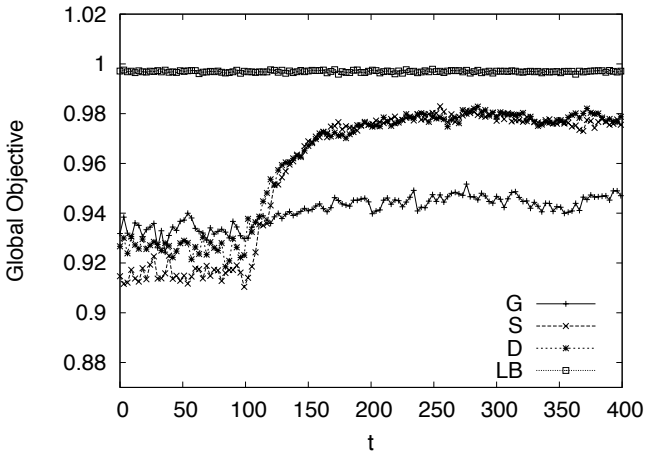


**Figure 1: Simulations results for 50 servers with 4 agents each with parameter values: (K=2,r=0.2,M=2). Each $t$ represents a "run" of 400 $\tau$ time steps with each agent having a fixed probability vector $\vec{p}$ during the run. At the end of each run, objectives are calculated, the queues cleared, and the agents reset/modify their $\vec{p}$ based on their learning algorithms. Results are averages over 50 different systems configurations, and error bars (differences in the mean) are less than .01 in all figures.**

Table 5.1 shows the results (and the standard deviations $\sigma$) for the setting where the system is not overloaded and where there are only two resource types. Load balancing performs best in this setting. This is an interesting, though expected result. All the algorithms perform well in this case and there are two reasons for the success of load balancing. First, this is a situation that is closest to the single resource resource allocation problem where load balancing excels. Second, because the problem is easy, there is no need for the agent based algorithms to explore alternative solutions. However, because of the nature of such algorithms, they occasionally try a suboptimal solution to determine whether a better alternative exists. This "exploration" is a desirable trait. In this case however, because the "greedy" solution is good, any exploration causes a minor drop in performance. Figure 1 shows the convergence characteristics of the agent based algorithms, demonstrating the quick learning capability of agent using $S$ and $D$ objectives.

## 5.2 Medium Difficulty Parameters

Table 5.2 shows the results (and the standard deviations $\sigma$) for moderately difficult settings. All three cases have one form of difficulty (too high an arrival rate, too many resources or too many types of jobs). In this setting, the agent based algorithms significantly outperform load balancing. The performance of load balancing degrades markedly for high $K$, and especially for high $M$. In fact, even setting the probability vectors at random (RAND) outperforms load balancing for $M = 8$.

This can be understood by the fact that the agent based approaches make decisions about only the first job in the queue. But it is this first job that can create serious bottlenecks in the system; if the first job needs more resources than the server can provide, the job cannot run and remains in queue, blocking other jobs from being processed as well. Load balancing, on the other hand, is attempting only to equalize the load across on the entire queue and does nothing to deal with such potential bottlenecks. For large $M$, the potential for bottlenecks increases markedly. Random probability vectors have the advantage over load balancing that they operate directly on the location where a bottleneck can occur. In cases for which there are many resources but low arrival rates ($r = 0.2, M = 8$) this provides an advantage for the random probability algorithm, whereas for cases with few resources buy high arrival rates, it does not.

More interestingly, in this setting, both $S$ and $D$ outperform $G$, showing the need for providing local and agent specific objectives. This result is explained by the need of the agents to extract the signal from the noise in order to learn the right actions. The close dependence of these agent specific objectives to the actions of the agents allows these algorithms to learn in settings where agents using $G$ do not. Figure 2 shows the convergence characteristics of the agent based algorithms, demonstrating that the rapid learning capability of $S$ and $D$. In this setting, both $S$ and $D$ outperform load balancing shortly after their learners are turned on.

## 5.3 High Difficulty Parameters

Table 5.3 shows the results (and the standard deviations $\sigma$) for the difficult problem settings. All three cases have two

**Table 2: System Processing Efficiency for Moderate Settings**

| K | r | M | Algorithm | Global Objective | $\sigma$ |
|---|---|---|---|---|---|
| | | | RAND | 0.644 | - |
| | | | G | 0.670 | 0.012 |
| 2 | .2 | 8 | S | **0.793** | 0.011 |
| | | | D | **0.793** | 0.012 |
| | | | LB | 0.225 | 0.013 |
| | | | RAND | 0.626 | - |
| | | | G | 0.629 | 0.0095 |
| 2 | .8 | 2 | S | 0.654 | 0.010 |
| | | | D | **0.691** | 0.0088 |
| | | | LB | 0.645 | 0.014 |
| | | | RAND | 0.530 | - |
| | | | G | 0.549 | 0.012 |
| 4 | .2 | 2 | S | **0.749** | 0.011 |
| | | | D | 0.687 | 0.0098 |
| | | | LB | 0.474 | 0.020 |

forms of difficulty (e.g., high arrival rates *and* too many job types). In this setting, not only do agent based algorithm significantly outperform load balancing, but the $D$ agent objective function starts to outperform the other objective functions. These results also show the importance of setting the agents' objectives to be functions that are both aligned with the system objective and impacted by the agents' actions. The team game ($G$) objective has poor learning properties for the individual agents since it includes information from the full system. The selfish ($S$) objective is not aligned with the global objective, and therefore leads to the agent learning the wrong actions. The difference objective consistently outperforms $G$ and $S$ for the difficult parameter settings, because it depends more closely on the action of the agents and is aligned with the global objective.

Note that for $K = 2$, $r = .8$, and $M = 8$, the differences in the mean are significant since the results are based on 50 runs and in this case $\frac{\sigma}{\sqrt{N}} = 0.0018$, well below the difference between the performance of $S$ and $D$. Figure 3 provide the convergence results for that setting, showing that because they depend on the actions of the agents more closely $S$ and $D$ outperform $G$ and that because it is aligned with the system objective, $D$ outperforms $S$.

# 6. DISCUSSION

In this work we investigated how agent based algorithms can learn to effectively solve a multi-resource optimization problem involving networks of heterogeneous servers. Conventional approaches to this problems (e.g., as load balancing) work well when there is instantaneous, centralized control. For all but very few applications, this is an unreasonable assumption on the system's capabilities. Practical, heuristics based approaches on the other hand provide good solutions for the resource problems, but often break down in the more general, multi-resource optimization case.

The agent based solution we propose is based on assigning agents to each server whose actions are to determine whether a job requiring specific resources should be processed at that server or shipped to another server, and if so, to which other server. After a job is shipped, a new agent (residing at the new server) becomes responsible for that job. These deci-

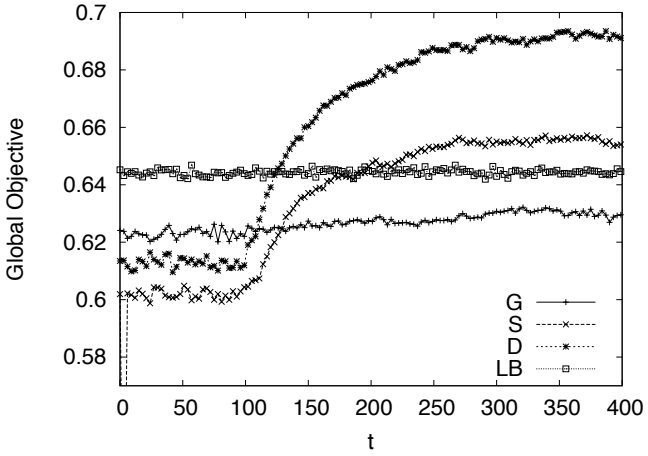


**Figure 2: Simulations results for 50 servers with 4 agents each with parameter values (r=0.8,M=2). For this medium difficulty problem, agents using D and G as objectives outperform load balancing.**

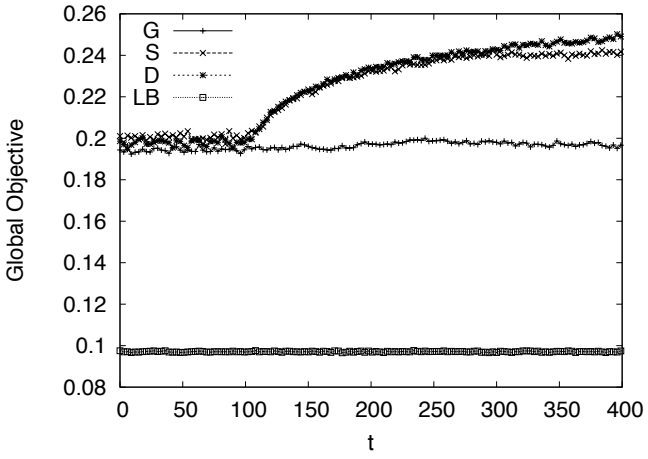


**Figure 3: Simulations results for 50 servers with 4 agents each with parameter values (r=0.8,M=8). For this hard problem, load balancing performs very poorly, and agents using tailored objectives $S$ and $D$ perform significantly better than agents using $G$.**

**Table 3: System Processing Efficiency for Difficult Setting**

| K | r | M | Algorithm | Global Objective | $\sigma$ |
|---|---|---|---|---|---|
| | | | RAND | 0.194 | - |
| | | | G | 0.198 | 0.0077 |
| 2 | .8 | 8 | S | 0.241 | 0.0093 |
| | | | D | **0.249** | 0.013 |
| | | | LB | 0.0974 | 0.0045 |
| | | | RAND | 0.130 | - |
| | | | G | 0.137 | 0.0050 |
| 4 | .2 | 8 | S | 0.178 | 0.0072 |
| | | | D | **0.238** | 0.014 |
| | | | LB | 0.0092 | 0.0022 |
| | | | RAND | 0.176 | - |
| | | | G | 0.189 | 0.0069 |
| 4 | .8 | 2 | S | **0.195** | 0.0080 |
| | | | D | **0.195** | 0.0073 |
| | | | LB | 0.139 | 0.0099 |

sions are based on agent objective functions (i.e., local goals) which are constructed to be aligned with the global objective and be directly impacted by the actions of an agent.

The results demonstrate that for particularly easy configurations, the agent-based methods do not outperform (and in fact underperform by a slight margin) a multi-resource version of load balancing. For moderately difficult problems, the agent based approaches start to outperform load balancing. In those cases, a multiagent system in which all the agents attempt to optimize the same global objective function only provide marginal improvements over conventional load balancing. However, those marginal improvements are obtained without requiring a centralized controller (only requirement is for the global objective to be broadcast at regular intervals). Finally, for difficult problems, agents using the difference objective ($D$) outperform both team games ($G$), selfish agents ($S$) and load balancing (up to four times).

In this study we explored only cases where the number of resources is small (K=2 and K=4)allowing for an agent to be responsible for each permutation of resources (split into high-low for each resource). When the number of resources rises to preclude each agent being responsible for a particular permutation, job "types" need to be selected. This process can either be done using prior knowledge or by using correlations among the jobs. The key factor in achieving good results is in having both an appropriate number of agents in the system and in ensuring each agent has an action space that is appropriate for the task. Exploring how agents can be grouped or "teamed" at each server to provide good solutions is an intriguing avenue for future research.

## 7. ACKNOWLEDGMENTS

The authors would like to thank David Wolpert for helpful discussions.

## 8. REFERENCES

[1] A. K. Agogino and K. Tumer. Handling communication restrictions and team formation in congestion games. *Journal of Autonomous Agents and Multi Agent Systems*, 13(1):97–115, 2006.

[2] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi Agent Systems*, 17(2):320–338, 2008.

[3] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[4] R. Akkiraju, P. Keskinocak, S. Murthy, and F. Wu. An agent-based approach for scheduling multiple machines. *Applied Intelligence*, 14(2):867–872, 2005.

[5] J. A. Boyan and M. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems - 6*, pages 671–678. Morgan Kaufman, 1994.

[6] J. Bredin, R. T. Maheswaran, C. Imer, Tamer Başar, D. Kotz, and D. Rus. Computational markets to regulate mobile-agent systems. *Autonomous Agents and Multi-Agent Systems*, 6(3):235–263, May 2003.

[7] J. Bredin, R.T. Maheswaran, C. Imer, T. Basar, D. Kotz, and D. Rus. A game-theoretic formulation of multi-agent resource allocation. In *Proceedings of the fourth International Conference of Autonomous Agents*, pages 349–356, 2000.

[8] P. Buhler and J. M. Vidal. Towards adaptive workflow enactment using multiagent systems. *Information Technology and Management Journal*, 6(1):61–87, 2005.

[9] Andrew Byde. A comparison between mechanisms for sequential compute resource auctions. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1199–1201, New York, NY, USA, 2006. ACM Press.

[10] B. Camille, P. Pierrick, and B. Chaib-draa. R-FRTDP: a real-time DP algorithm with tight bounds for a stochastic resource allocation problem. In *Proceedings of the 20th Canadian Conference on Artificial Intelligence*, Montreal, Canada, May 2007.

[11] D. R. Cheriton and K. Harty. A market approach to operating system memory allocation. In S.E. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1995.

[12] C. Claus and C. Boutilier. The dynamics of reinforcement learning cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752, Madison, WI, June 1998.

[13] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.

[14] Denise de Oliveira, Paulo R. Ferreira Jr., and Ana L. C. Bazzan. A swarm based approach for task allocation in dynamic agents organizations. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1252–1253, Washington, DC, USA, 2004. IEEE Computer Society.

[15] M. Dorigo and L. M. Gambardella. Ant colony systems: A cooperative learning approach to the

travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[16] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, 2nd edition, 2004.

[17] C. Georgousopoulos and O. F. Rana. Choosing a load balancing scheme for agent-based digital libraries. In *Int. Symposium on Parallel and Distributed Architectures (ISPA)*, Sorrento, Italy, December 2006. Springer Verlag.

[18] Bhaskar Ghosh and S. Muthukrishnan. Dynamic load balancing in parallel and distributed networks by random matchings (extended abstract). In *SPAA '94: Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 226–235, New York, NY, USA, 1994. ACM Press.

[19] A. Globus, J. Crawford, J. Lohn, and A. Pryor. Scheduling earth observing satellites with evolutionary algorithms. In *Proc. of International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2003.

[20] A. Greenwald, E. Friedman, and S. Shenker. Learning in network contexts: Experimental results from simulations. *Journal of Games and Economic Behavior: Special Issue on Economics and Artificial Intelligence*, 35(1/2):80–123, 2001.

[21] M.-T. T. Hsiao and A. A. Lazar. Optimal flow control of multi-class queueing networks with decentralized information. In *IEEE Infocom '89*, pages 652–661, 1987.

[22] Jinpeng Huai, Tianyu Wo, and Yunhao Liu. Resource management and organization in crown grid. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 10, New York, NY, USA, 2006. ACM Press.

[23] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.

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

[25] D. Kotz and N. Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of Supercomputing '94*, pages 640–649, Washington, DC, November 1994. IEEE Computer Society Press.

[26] J. F. Kurose and R. Simha. A microeconomic approach to optimail resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 35(5):705–717, 1989.

[27] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas. Load balancing across near-homogeneous multi-resource servers. In *Proceedings of the ninth heterogeneous Computing Workshop*, pages 61–70, Cancun, Mexico, 2000.

[28] Z. Li and M. Parashar. An infrastructure for dynamic composition of grid service. In *Proceedings of the 7th IEEE International Conference on Grid Computing*, pages 315–316, Barcelona, Spain, September 2006. IEEE Computer Society Press.

[29] S. Lynden and O. F. Rana. Coordinated learning to support resource management in computational grids. In *2nd IEEE International Conference on Peer-2-Peer Computing*, Linkoping, Sweden, September 2002. IEEE Computer Society Press.

[30] B. Di Martino and O.F. Rana. Grid performance and resource management using mobile agents. In V. Getov, M. Gerndt, A. Hoisie, A. Malony, and B. Miller, editors, *Performance Analysis and Grid Computing*. Kluwer, November 2003.

[31] Debasis Mishra and Bharath Rangarajan. Cost sharing in a job scheduling problem using the shapley value. In *EC '05: Proceedings of the 6th ACM conference on Electronic commerce*, pages 232–239, New York, NY, USA, 2005. ACM Press.

[32] M. E. J. Newman. Models of the small world (a review). *Journal of Statistical Physics*, 101:819–841, 1987.

[33] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

[34] P. Stone. TPOT-RL applied to network routing. In *Proceedings of the Seventeenth International Machine Learning Conference*, pages 935–942. Morgan Kauffman, 2000.

[35] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[36] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[37] H. Tang and H. Tianfield. Self-organizing networks of communications and computing. *International Transactions on Systems Science and Applications*, 1(4):421–431, September 2006.

[38] H. Tianfield and R. Unland. Towards self-organization in multi-agent systems and grid computing. *Multiagent and Grid Systems*, 1(2):89–95, October 2005.

[39] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 330–337, Honolulu,HI, May 2007. **Best Paper Award**.

[40] K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 378–385, Bologna, Italy, July 2002.

[41] D. Watts. *Small Worlds*. Princeton University Press, Princeton, NJ, 1999.

[42] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.

[43] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616–623, Montreal, January 2001. ACM Press.

[44] Xuefeng Yu and Bala Ram. Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups. *International Journal of Production Research*, 44(22):4793–4813, November 2006.