

# Tradeoffs in Minimizing Computation Load and Communication Cost in Multi-Agent Systems

Kiran Bhutani \* Bilal Khan † Birendro Roy ‡  
Center for Computational Science  
Naval Research Laboratory, Washington D.C.  
<http://www.nrl.navy.mil/ccs/project/public/DC/web/>

September 14, 2003

**Abstract**

## 1 Introduction

*Agents* are mobile, intercommunicating, computational units that can act autonomously within a complex dynamic environment in order to realize a set of goals or tasks for which they are designed [?]. An agent’s environment consists of other agents as well as the physical infrastructure. The infrastructure in turn is comprised of *machines* that provide computational resources for agents and a *network* that provides resources for agent intercommunication. At any point in time, an agent “resides” on a particular machine, and attempts to realize its goals by communicating with other agents over the network and by performing computation using the resources of the machine on which it resides. Agent communication and computation result in consumption of both network and machine resources. The extent to which each of these resources is consumed depends largely on the distribution of agents to machines. We refer to an entity that determines the assignment of agents to machines as a *controller*. The controller’s logic implements an *agent mobility algorithm*. In this paper, we consider the problem of designing agent mobility algorithms which permit the system of agents to make optimal use of computation and communication resources.

---

\*Permanent address: The Catholic University of America. Department of Mathematics, Washington D.C. 20064.

†ITT Industries, Center for Computational Science, Naval Research Laboratory.

‡Massachusetts Institute of Technology, Computer Science Department.

The NRL Center for Computational Science has developed CHIME [?], a multi-agent system for heterogeneous distributed computing. Within CHIME, agent mobility is a crucial enabler of performance optimization, since it permits balancing computational load and minimization of latency in inter-agent communications. This is particularly important for performance-critical applications written over the CHIME framework—one such application is as Optiprism [?], a hierarchical network management system for all-optical networks. Until recently, CHIME supported agent mobility services, but did not assist agents in determining when a change in their physical location might improve application performance. The algorithms exposted in this paper have been implemented and tested within the CHIME framework.

In considering agent mobility algorithms, it is illustrative to consider two degenerate extremes:

- ◆ *Assign all agents to a single machine.* Such an assignment certainly makes the system communication cost very small, but causes a very high computational load on a single machine.
- ◆ *Assign  $\lfloor \frac{n}{m} \rfloor$  agents to each machine,* where  $n$  is the number of agents and  $m$  is the number of machines. While this assignment balances the load evenly on machines, it typically results in prohibitively high communication cost between agents.

Our goal here is to investigate the space of solutions between these two degenerate approaches. In doing so, we will quantify the intuition that the minimization of communication cost and computational load are inversely related. We shall then describe an algorithm that incorporates a tunable trade-off between optimizing communication cost and optimizing machine load. The proposed algorithm will take into account the computational needs and communication patterns of agents at time  $t$ , and use these to determine the assignment of agents to machines at time  $t + 1$ .

## 2 Aspects of the Agent Mobility Model

We have identified the key actors within the system to be: the agents and the controllers. The infrastructure resources consists of a set of machines and a network interconnecting them. We denote the set of agents as  $V$ , the set of controllers as  $A$ , the set of machines as  $M$ , and the network links as  $E$ . A model of agent mobility has several important and independent aspects, which we identify and describe.

## 2.1 Infrastructure Characteristics

The phrase “infrastructure characteristics” refers to the properties of machines and the network that are independent of the presence of agents. The model must describe the computation and communication resources of the infrastructure. Possible approaches include:

- I1 *Static Uniform*— Network characteristics and machine capabilities do not change over time. In particular there are a fixed number of machines, and a fixed topology of links interconnecting them. All links in the network have the same characteristics and all machines have the same capabilities.
- I2 *Static Non-Uniform*— Network characteristics and machine capabilities do not change over time, but network links may have different characteristics and machines may have different capabilities.
- I3 *Dynamic Non-Uniform*— Network characteristics and machine capabilities may change over time. The number of machines and the topology of links interconnecting them may change. Network links may have different characteristics and machines may have different capabilities.

In this paper we consider a static, non-uniform approach where machines capabilities are uniform but network links may have different delays associated with them. Machines are modelled as a set of  $m$  vertices  $M$ , while network links are modelled as weighted edges of a complete graph on  $M$ . The weight of the link connecting machines  $M_i$  and  $M_j$  is denoted  $d_{ij}$ , and represents the cost (in terms of delay) of sending a single message across the link. We take  $0 < d_{ii} < d_{ij}$  for all  $j \neq i$  to reflect the fact that inter-machine communication is more costly than intra-machine communication. We also assume a symmetric case where all machines have the same computational capabilities. The graph  $(M, E, d)$  captures the infrastructure characteristics.

## 2.2 Agent Characteristics

The term “agent characteristics” refers to the computational and communication needs of agents. The model can approach agent characteristics under various hypotheses:

- A1 *Static*— An agent’s computation and communication needs do not change over time. The number of agents in the system is fixed.
- A2 *Dynamic*— An agent’s computation and communication needs can change over time. The number of agents in the system is changing.

In this paper we consider the static approach. We assume that an agent’s activities are well-approximated by a random process that can be described by a

finite set of parameters. Specifically, the intercommunication needs of the agents are modelled as a symmetric  $|V|$  by  $|V|$  real matrix  $C$  whose rows and columns each sum to 1. Entry  $C_{u,v}$  represents the probability with which agents  $u$  and  $v$  communicate. The computation needs of agents are represented as a length  $|V|$  real vector  $\delta$ . Entry  $\delta_u$  represents the relative computational requirements of agent  $u$ . The vector  $(C_{u,v})_{v \in V}$  and value  $\delta_u$  capture the characteristics of agent  $u$ . Note that agent  $u$  does not typically have explicit knowledge of  $(C_{u,v})_{v \in V}$  or  $\delta_u$ —the model merely hypothesizes that the values exist. In what follows, the special cases when  $(C_{u,v})_{v \in V} = \frac{1}{n-1}$  and  $\delta_u = 1$  are referred to as *uniform agent communication patterns* and *uniform agent computation needs*, respectively.

### 2.3 Controller Characteristics

The term “controller characteristics” refers to constraints on the behavior of controllers. The model must address the following questions: *How do controllers move agents? Can controllers move several agents simultaneously? Is there an inherent cost for moving an agent?*

- C1 *Sequential*— Agent movements are determined by the controllers, which operate sequentially; no two agents can migrate at the same time.
- C2 *Parallel synchronous*— Agents movements are determined by the controllers, which operate concurrently at discrete time intervals.

In this paper we consider only the sequential approach. The assignment of agents to machines at time  $t$  is given by a function  $f_t : V \rightarrow M$ . The controllers determine  $f_t$  at each time  $t$ . In the sequential approach  $f_t$  and  $f_{t+1}$  differ in at most one value of their domain. In this work we consider controllers which can move agents without paying any cost.

### 2.4 Infrastructure State

The phrase “infrastructure state” refers to the properties of machines and the network that are dependent on the presence of agents. The model must describe the utilization of computation and communication resources infrastructure. Here we consider:

- S1 *Computational Load*— The effect of agents on the computational resources. The computational load on a machine  $M_i$  at time  $t$  is denoted  $L_t(m)$ , and defined to be the sum of the computational needs of all agents that reside on the machine  $M_i$  at time  $t$ .

$$L_t(M_i) \stackrel{\text{def}}{=} \sum_{v \in V; f_t(v)=M_i} \delta_v$$

S2 *Link Utilization*— The effect of agents on the communication resources. The utilization of a link  $(M_i, M_j)$  is defined to be the expected communication per unit time between all agents that reside on  $M_i$  and  $M_j$ .

$$\gamma_t(M_i, M_j) \stackrel{\text{def}}{=} \sum_{u,v \in V; f_t(u)=M_i; f_t(v)=M_j} C_{u,v}$$

## 2.5 Controller Distribution

The model must address the following question: *How many controllers are operating?*

D1 *Centralized*— There is one controller.

D2 *Distributed (machine-based)*— There is one controller associated with each machine.

For scalability reasons, we will consider the distributed (machine-based) approach. A natural question then arises: *What information does each controller have about agent/infrastructure characteristics and infrastructure state, and how accurate is this information?*

In our model, each controller  $A_i$  has all infrastructure characteristics but can only estimate agent characteristics for agents on its own machine. Since even agent  $u$  itself does not typically know  $(C_{u,v})_{v \in V}$  or  $\delta_u$ , these quantities have to be estimated based on the historical communicative and computational actions of  $u$ . Thus, while controller  $A_i$  has accurate information about  $(M, E, d)$ , it has only estimates of  $(C_{u,v})_{v \in V}$  and  $\delta_u$  for each agent  $u$  residing on  $M_i$ . Specifically, we assume that  $A_i$  can ask machine  $M_i$  the following information:

Q1 What is the cumulative frequency of agent  $u$ 's communication with agents on machine  $M_j$ ? (where  $u$  is presently on machine  $M_i$ , and  $j$  is in  $1, \dots, m$ ).

Q2 How much computation has agent  $u$  been doing? (for any  $u$  currently on  $M_i$ ).

These questions provide  $A_i$  with estimates for

$$\gamma_t^{M_i}(u, M_j) \stackrel{\text{def}}{=} \sum_{v \in V; f_t(v)=M_j} C_{u,v}$$

and  $\delta_u$ , respectively, for each agent  $u$  presently on machine  $M_i$ . In practice, these quantities can be measured by machine  $M_i$ , since it provides computational resources for agent  $u$  and mediates between  $u$  and the communication network.

In addition,  $A_i$  can obtain some information about infrastructure state by asking machine  $M_i$ :

Q3 What is the current (estimated) load on the machine  $M_j$  ( $j \in 1, \dots, m$ )?

The answer to this question is denoted  $L_t^{M_i}(M_j)$ . In implementation, each machine measures its own computational load, and this information can be propagated to all machines using either an independent protocol, or piggybacked on top of inter-agent communication.

Based on the answers to questions of the form Q1-Q3, controller  $A_i$  must make a decision about ejecting an agent  $v$  to another machine  $M_j$ .

### 3 Objective Functions

Given the assignment  $f_t$  of agents  $V$  to machines  $M$  the *computational load* of the system at time  $t$  is defined to be

$$\Lambda_t \stackrel{\text{def}}{=} \max_{i=1\dots m} L_t(M_i)$$

while the *communication cost* of the system is defined as

$$\Gamma_t \stackrel{\text{def}}{=} \sum_{M_i, M_j \in M; M_i \neq M_j} \gamma_t(M_i, M_j) \cdot d_{ij}.$$

#### 3.1 Assessing Local Reassignments

The controller  $A_i$  must determine how to reduce the computational load and communication cost contributed by agents on machine  $M_i$ . Suppose agent  $u$  is on machine  $M_i$  at time  $t$ , and controller  $A_i$  wants to evaluate whether  $u$  should be sent to machine  $M_k$ .  $A_i$  can ask  $M_i$  question Q1 to obtain an estimate of  $u$ 's communication patterns with agents on  $M_k$ . (Of course  $A_i$  has no guarantees of the future behavior of  $u$ , but we are assuming that agents communications are adequately modelled by relative frequencies.) The communications of  $u$  presently contribute

$$\sum_{j=1}^m \gamma_t^{M_i}(u, M_j) \cdot d_{ij}$$

to the overall communication cost  $\Gamma_t$  of the system. Sending agent  $u$  to machine  $M_k$  could be expected to change this contribution to

$$\sum_{j=1}^m \gamma_t^{M_k}(u, M_j) \cdot d_{kj}$$

The above quantity represents the hypothetical cost that communications of  $u$  would have incurred if  $u$  had been placed on machine  $M_k$ . We define *communi-*

cation benefit of moving  $u$  from  $M_i$  to  $M_k$  as

$$\partial\Gamma^{M_i}(u, M_k) \stackrel{\text{def}}{=} 1 - \frac{\sum_{j=1}^m \gamma_t^{M_i}(u, M_j) \cdot d_{kj} - \chi_{ij} C_{u,u} d_{ij}}{\sum_{j=1}^m \gamma_t^{M_i}(u, M_j) \cdot d_{ij}}$$

where  $\chi_{ij} = 1$  if  $i = j$  and 0 otherwise. Then  $\partial\Gamma^{M_i}(v, M_k)$  takes values between  $(-\infty, 1)$ , with the optimal choice occurring close to the value 1. This occurs when the cost of assigning an agent  $v$  onto machine  $M_k$  becomes very small.

Controller  $A_i$  must similarly determine how to reduce the computational load on  $M_i$ . The computation of  $u$  presently contributes  $\delta_u$  to the load on  $M_i$ . We define the *computation benefit* of moving  $u$  from  $M_i$  to  $M_k$  as

$$\partial\Lambda^{M_i}(u, M_k) \stackrel{\text{def}}{=} 1 - \frac{L(M_k) + \delta_u}{L(M_i) - \delta_u}$$

The quantity  $\partial^{M_i}\Lambda(v, M_k)$  takes values between  $(-\infty, 1)$ , with the optimal choice occurring close to 1. This occurs when the present load of machine  $M_k$  is much smaller than the load on machine  $M_i$ .

## 4 The $(\alpha, \beta)$ Algorithm

Since controllers must optimize with respect to two objective functions, it is reasonable to consider a mixed strategy. We represent the relative importance of computational and communication resources by two numbers  $\alpha, \beta \in \mathbb{R}^+$  satisfying  $(\alpha + \beta = 1)$ . The total cost of the system at time  $t$  is defined to be

$$w_t \stackrel{\text{def}}{=} \alpha\Gamma_t + \beta\Lambda_t$$

The assignment  $f_t$  is said to be  $(\alpha, \beta)$ -*optimal* if  $w_t$  is minimal over all choices of functions  $f_t : V \rightarrow M$ . The overall benefit of moving agent  $v$  to machine  $M_k$  is then taken as

$$\partial\omega^{M_i}(v, M_k) \stackrel{\text{def}}{=} \alpha \cdot \partial\Gamma^{M_i}(v, M_k) + \beta \cdot \partial\Lambda^{M_i}(v, M_k).$$

At time  $t$ , each controller  $A_i$  ( $i = 1 + (t \bmod m)$ ) computes a table of values for each agent  $v$  which resides on  $M_i$ . The table describes the communication, computation, and overall benefit of moving  $v$  to various machines  $M_1, \dots, M_m$ .

$v$	$M_1$	$M_2$	$\dots$	$M_j$	$\dots$	$M_m$
$\partial\Gamma^{M_i}$			$\dots$	$\partial\Gamma^{M_i}(v, M_j)$	$\dots$	$\partial\Gamma^{M_i}(v, M_m)$
$\partial\Lambda^{M_i}$			$\dots$	$\partial\Lambda^{M_i}(v, M_j)$	$\dots$	$\partial\Lambda^{M_i}(v, M_m)$
$\partial\omega^{M_i}$			$\dots$	$\partial\omega^{M_i}(v, M_j)$	$\dots$	$\partial\omega^{M_i}(v, M_m)$

By examining these tables for agents  $v$  residing on  $M_i$ , the controller  $A_i$  determines which agent  $v$  and machine  $M_j$  provides the maximal value of  $\partial\omega^{M_i}(v, M_j)$  which is greater than  $\partial\omega^{M_i}(v, M_i)$ . If more than one such agent-machine pair exist, the controller picks from one of the candidate pairs uniformly at random. The controller then ejects agent  $v$  from  $M_i$  to machine  $M_j$ , effectively dictating that  $f_{t+1}(v) = M_j$ .

## 5 Convergence of the $(0, 1)$ Algorithm

Here we show that the computational load minimizer converges “fast”.

**Theorem 5.1.** *In an agent system with  $n$  agents on  $m$  machines, where agent computational needs are bounded in the range  $1/k \leq \delta_u \leq k$  ( $k \in \mathbb{R}$ ), the  $(0, 1)$  algorithm converges to a  $(0, 1)$ -optimal solution in  $O(nmk^2)$  steps.*

*Proof.* Clearly, the  $(0, 1)$ -optimal solution is an assignment where the number of agents on any two machines differs by at most 1. Let  $M$  denote the set of machines, and  $L_t : M \rightarrow \mathbb{N}$  be the function whose value describes the load on each machine  $M_i \in M$ . Suppose the optimal assignment of agents to machines entails a load of  $\lambda$ . We define

$$*M_i \stackrel{\text{def}}{=} \max\{(L_t(M_i) - \lambda), 0\}$$

to be the extent of *overloading* on machine  $M_i$ . Similarly, we put

$$*_M_i \stackrel{\text{def}}{=} \max\{(\lambda - L_t(M_i)), 0\}$$

to represent the extent of *loading* on machine  $M_i$ .

Finally, we define

$$\mathcal{O} \stackrel{\text{def}}{=} \{M_i \in M \mid *_M_i \neq 0\}$$

to be the set of machines that are overloaded. Then

$$M \setminus \mathcal{O} = \{M_i \in M \mid *_M_i = 0\}$$

is the set of machines which are underloaded.

Clearly, the  $(0, 1)$ -optimal solution is an assignment where  $\mathcal{O}$  is the empty set, or equivalently, an assignment for which

$$\Phi \stackrel{\text{def}}{=} \sum_{i=1}^m *_M_i = 0$$

We will show that the  $(0, 1)$  algorithm converges to this state in no more than  $O(nmk^2)$  steps.

At each step of the algorithm, the controller  $A_i$  may eject at most one of the agents residing on machine  $M_i$ . There are two cases:  $M_i \in M \setminus \mathcal{O}$ , and  $M_i \in \mathcal{O}$ , which we consider now in turn.

CASE  $M_i \in M \setminus \mathcal{O}$ :

For each agent  $v$  on machine  $M_i$ , define

$$\Lambda_{\max}^{M_i}(v) \stackrel{\text{def}}{=} \max_{M_k \in M} \partial \Lambda^{M_i}(v, M_k).$$

If agent  $v$  on machine  $M_i$  satisfies  $\partial \Lambda^{M_i}(v, M_i) = \Lambda_{\max}^{M_i}(v)$ , then  $A_i$  will certainly set  $f_{t+1}(v) = f_t(v) = M_i$ . Two scenarios may arise:

(i) For all agents  $v$  on machine  $M_i$ ,  $\partial \Lambda^{M_i}(v, M_i) = \Lambda_{\max}^{M_i}(v)$ . Then no agent will be ejected from machine  $M_i$ .

(ii) Otherwise let

$$E_i = \{v \mid f(v) = M_i \text{ and } \partial \Lambda^{M_i}(v, M_i) \neq \Lambda_{\max}^{M_i}(v)\} \neq \emptyset.$$

For each agent  $v$  in  $E_i$ , there exists some machine  $M_{j(v)} \in M \setminus \mathcal{O}$ , where  $j(v)$  is in  $1, \dots, m$ ,  $j(v) \neq i$ , such that  $\partial \Lambda^{M_i}(v, M_{j(v)}) = \Lambda_{\max}^{M_i}(v)$ . The algorithm  $A_i$  will set  $f_{t+1}(v) = M_{j(v)}$  for exactly one  $v$  in  $E_i$ . Thus, at time  $t + 1$  one agent  $v$  is ejected from  $M_i$  to the machine  $M_{j(v)}$  which had the least load at time  $t$ . Note that after ejecting  $v$  to  $M_{j(v)}$ , the machine  $M_{j(v)}$  is still in  $M \setminus \mathcal{O}$ , and in particular, the membership of  $\mathcal{O}$  cannot increase.

CASE  $M_i \in \mathcal{O}$ :

Let  $L_t(M_j) = \min_{M_k \in M} L_t(M_k)$ . Since  $\partial \Lambda^{M_i}(v, M_j) \geq \partial \Lambda^{M_i}(v, M_k)$  for all  $k$ ,  $A_i$  will set  $f_{t+1}(v) = M_j$ . Note that the machine  $M_j$  is necessarily in  $M \setminus \mathcal{O}$ , since otherwise  $L_t(M_k) > L_t(M_j) > \lambda$  for all  $k$ , which would imply  $\sum_{M_k \in M} L_t(M_k) > m\lambda$ , a contradiction. Thus if an agent  $v$  is kicked off an overloaded machine  $M_i$  at time  $t$ , then it is necessarily sent to a non-loaded machine  $M_j$  at time  $t + 1$ , and  $*M_i$  decreases by 1.

If the algorithm has not yet converged, then there must be some  $M_i \in \mathcal{O}$ . The quantity  $\Phi$  decreases by at least  $1/k$  with each cycle through the  $m$  controllers. Since  $\Phi$  starts as a positive finite integer  $\leq nk$ , the algorithm converges in no more than  $nmk^2$  steps, at which point  $f_{t+1}(v) = f_t(v)$  for all  $v$ ,  $\Phi = 0$  and  $|\mathcal{O}| = 0$ .  $\square$

We remark that since  $k$  can be exponential in the size of the problem instance, this algorithm does not imply a polynomial time solution to bin-packing types of problems.

## 6 Convergence of the (1, 0) Algorithm

An agent system consisting of  $n$  agents is said to have *near-uniform agent communication patterns* if

$$|C_{u,v} - \frac{1}{n}| < \frac{1}{n(n+1)}$$

**Theorem 6.1.** *Given an agent system consisting of  $n$  agents on  $m$  machines, under the assumption of uniform communication costs and near-uniform communication patterns, the (1, 0) algorithm converges to a (1, 0)-optimal solution in  $O(nm)$  steps.*

*Proof.* Clearly, the (1, 0)-optimal solution is an assignment where all agents are placed on the same machine. We will show that under the assumption of uniform communication costs and near-uniform communication patterns, the (1, 0) algorithm converges to this state in no more than  $O(nm)$  steps.

If the system has not converged to a state where all agents are on the same machine, there are at least two machines with agents on them. Without loss of generality, suppose  $M_1, M_2$  are the two machines with the highest number of agents  $n_1$  and  $n_2$  respectively ( $n_1 \leq n_2$ ).

Let  $u$  be an agent on  $M_1$ . Then,  $\Gamma^{M_1}(u, M_1) = 0$ , while

Now  $\partial\Gamma^{M_1}(u, M_2)$  is by definition,

$$1 - \frac{\gamma_t^{M_1}(u, M_1) \cdot d_{21} + \gamma_t^{M_1}(u, M_2) \cdot d_{22} + \sum_{j=3}^m \gamma_t^{M_1}(u, M_2) \cdot d_{2j}}{\gamma_t^{M_1}(u, M_1) \cdot d_{11} + \gamma_t^{M_1}(u, M_2) \cdot d_{12} + \sum_{j=3}^m \gamma_t^{M_1}(u, M_2) \cdot d_{1j}}$$

But the hypothesis of near-uniform communication patterns implies that

$$\begin{aligned} \gamma_t^{M_1}(u, M_1) &< n_1 \left( \frac{1}{n} + \frac{1}{n(n-1)} \right) \\ \gamma_t^{M_1}(u, M_2) &> n_2 \left( \frac{1}{n} - \frac{1}{n(n-1)} \right) \end{aligned}$$

Thus

$$\gamma_t^{M_1}(u, M_1) \cdot d_{21} < \gamma_t^{M_1}(u, M_2) \cdot d_{12}.$$

Since  $d_{11} = d_{22} = 0$ , we get that

$$\partial\Gamma^{M_1}(u, M_2) > 0 = \partial\Gamma^{M_1}(u, M_1).$$

Thus  $u$  gets kicked to machine  $M_2$ . The algorithm terminates when all agents have been kicked to  $M_2$ , a process which requires at most  $nm$  steps.  $\square$

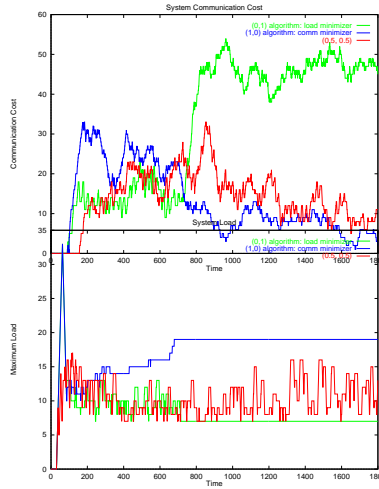
We note that if the system does not satisfy the condition of near-uniform agent communication patterns, then the  $(1, 0)$  algorithm may not converge to an optimal solution. This is illustrated by the following example.

\*\*\* INSERT EXAMPLE HERE \*\*\*

## 7 Experiments

Here we run some generic experiments to show performance for intermediate values of  $\alpha$  and  $\beta$  to show a smooth tradeoff.

\*\*\* EXPERIMENTS \*\*\*



## 8 Tradeoffs

Here we try and make quantitative sense of the intuitive idea that communication cost minimization and load minimization are inversely proportional, using whatever analysis and experiments appeared earlier.

## 9 Conclusion

We summarize.