Probabilistically Survivable MASs *

Sarit Kraus
Dept. of Computer Science
Bar-Ilan University
Ramat-Gan, 52900 Israel
sarit@cs.biu.ac.il

V.S. Subrahmanian and **N. Cihan Taş**
Dept. of Computer Science
University of Maryland
College Park, MD 20742
{vs,ctas}@cs.umd.edu

Abstract

Multiagent systems (MAS) can "go down" for a large number of reasons, ranging from system malfunctions and power failures to malicious attacks. The placement of agents on nodes is called a *deployment* of the MAS. We develop a probabilistic model of survivability of a deployed MAS and provide two algorithms to compute the probability of survival of a deployed MAS. Our probabilistic model does not make independence assumptions though such assumptions can be added if so desired. An optimal deployment of a MAS is one that maximizes its survival probability. We provide a mathematical answer to this question, an algorithm that computes an exact solution to this problem, as well as several algorithms that quickly compute approximate solutions to the problem. We have implemented our algorithms - our implementation demonstrates that computing deployments can be done scalably.

1 Introduction

As multiagent systems (MASs) are increasing used for critical applications, the ability of these MASs to survive intact when various external events occur (e.g. power failures, OS crashes, etc.) becomes increasingly important. However, one never knows when and if a system will crash or be compromised, and hence, any model of MAS survivability must take this uncertainty into account.

We provide for the first time, a formal model for reasoning about survivability of MASs which includes both a declarative theory of survivability, as well as implemented algorithms to compute optimal ways of deploying MASs across a network.

A MAS-deployment specifies a placement of agents on various network nodes. Based on probabilistic information about the survivability of a given node, we develop a formal theory describing the probability that a given deployment will survive. This probability reflects the best guarantee we have of the MAS surviving. Our model does not assume that node failures are independent, though independence information can be easily added if so desired. The technical problem we need to grapple with is that of finding a MAS-deployment of the agents having the highest probability of survival. As we do not make unrealistic independence assumptions, this problem turns out to be intractable. As a consequence, heuristics are required to find a deployment (even if it is sub-optimal). We develop algorithms for the following tasks:

1. Given a MAS-deployment, how do we compute its probability of survival?
2. Find a MAS-deployment with the highest probability of survival - this algorithm is infeasible to implement in practice due to the above mentioned complexity results.
3. We develop a suite of heuristic algorithms to find (sub-optimal) MAS-deployments.

We have conducted detailed experiments with our algorithms - for space reasons, only some of them are described here. The experiments show that our heuristic algorithms can find deployments very fast.

2 Preliminaries

Agents. The only assumptions we make about agents is that they provide one or more services. We further assume that all host computers on which agents are located have a finite amount of memory resources, and that each (copy of an) agent a requires some amount of memory, denoted by mem(a). A *multiagent application* MAS is a finite set of agents - we make the assumption that all agents in a multiagent application are needed for it to function.

Networks. A *network* is a triple $(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ where $\mathcal{N}$ is a set of called *nodes*, $\mathsf{edges} \subseteq \mathcal{N} \times \mathcal{N}$ specifies which nodes can communicate with which other nodes, and $\mathsf{mem} : \mathcal{N} \to \mathbf{R}$ specifies the total memory[1] available at node n for use by agents situated at n. A network is *fully connected* iff $\mathsf{edges} = \mathcal{N} \times \mathcal{N}$.

*The first author is also affiliated with UMIACS. This work was supported in part by the Army Research Lab under contract DAAL 0197K0135, the CTA on Advanced Decision Architectures, by ARO contract DAAD190010484, by NSF grant IIS0222914 and an NSF ITR award 0205489.

[1] Note that the symbol mem is used to both denote the memory requirements of an agent, as well as the memory available at a node. It is easy to determine the intended meaning of this expression from context.

Definition 2.1 *Suppose* MAS *is a multiagent application and* $Ne = (\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ *is a network. A deployment for* MAS *on* Ne *is a mapping* $\mu : \mathcal{N} \rightarrow 2^{\mathsf{MAS}}$ *specifying which agents are located at a given node. (As usual, if* X *is a set,* 2^X *is the power set of* X*).* μ *must satisfy the following condition:*

- $(\forall \mathsf{a} \in \mathsf{MAS})(\exists n \in \mathcal{N})\ \mathsf{a} \in \mu(n)$. *This condition says that every agent must be deployed somewhere.*
- $(\forall n \in \mathcal{N})\mathsf{mem}(n) \geq \Sigma_{\mathsf{a} \in \mu(n)}\mathsf{mem}(\mathsf{a})$. *This condition says that the agents deployed at a node cannot use more memory than that node makes available.*

Intuitively, $\mu(n) = \{\mathsf{a}_1, \mathsf{a}_2\}$ says that agents $\mathsf{a}_1, \mathsf{a}_2$ are deployed at node n_1.

Example 2.1 *Suppose* $\mathcal{N} = \{n_1, n_2, n_3, n_4\}$ *and* $\mathsf{MAS} = \{a, b, c, d\}$. *An example deployment is given by:* $\mu(n_1) = \{a, b\}$, $\mu(n_2) = \{c, d\}$, $\mu(n_3) = \{a, b, c, d\}$ *and* $\mu(n_4) = \{d\}$. *This example will be used throughout this paper.*

3 Related Work

To our knowledge, there are no probabilistic models of survivability of a MAS. However, there are many works that are in related areas.

[Shehory *et al.*, 1998] use agent-cloning and agent-merging techniques to mitigate agent over-loading and promote system load balancing. Fan [Fan, 2001] proposes a BDI mechanism to formally model agent cloning to balance agent workload.

[Fedoruk and Deters, 2002] propose transparent agent replication technique - though an agent is represented by multiple copies, this is an internal detail hidden from other agents. Several other frameworks also support this kind of agent fault tolerance [Mishra, 2001].

[Marin *et al.*, 2001] develop adaptive fault tolerance techniques for MASs. They use simulations to assess migration and replication costs. However, [Marin *et al.*, 2001] concludes by saying that they do not address the questions of which of the agents to replicate, how many replicas should be made, where those replicas should be allocated. These questions are addressed in the current paper, but we do not propose a mechanism to synchronize agent replications.

[Kumar *et al.*, 2000] focus on the problem of broker agents that are inaccessible due to system failures. They use the theory of teamwork to specify robust brokered architectures that can recover from broker failure. We, on the other hand, consider the possible failure of any agent in the multi-agent systems.

The problem of network reliability has been studied extensively — [Gartner, 1999] provide an excellent survey. In this paper we build on top of these studies and assume, as discussed below, that there is a disconnect probability function for a network specifying the reliability of each node of the network.

The problem of fault-tolerant software systems has some similarities to our agent survivability problem. An extensive study was performed to solve this problem using the N-Version Problem (NVP) approach. The NVP is defined as the "independent generation of $N \geq 2$ functionally equivalent programs from the same initial specification" [Lyu and He, 1993]. In this approach, the reliability of a software system is increased by developing several versions of special modules and incorporating them into a fault-tolerant system [Gutjahr, 1998]. However, these works (i) make unnecessary or unwarranted independence assumptions, (ii) provide only a measure of *expected* survivability rather than guaranteed survivability, (iii) do not consider replication.

4 A Probabilistic Model of Survivability

Multiagent applications can "go down" because nodes on which agents are located can crash. Alternatively, agents are on a mobile node (e.g. a vehicle) may wander beyond communications range, thus dropping out of the network.

Definition 4.1 *A disconnect probability function for a network* $(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ *is a mapping* $\mathsf{dp} : \mathcal{N} \rightarrow \mathcal{C}[0, 1]$ *where* $\mathcal{C}[0, 1]$ *is the set of all closed subintervals of* $[0, 1]$.

Intuitively, if $\mathsf{dp}(N) = [0.2, 0.3]$, then this says that there is a $20 - 30\%$ probability that node N will get disconnected from the network. Note that this model supports the situation where we do not know the probability of node n getting disconnected - in this case, we can set $\mathsf{dp}(N) = [0, 1]$. Likewise, if we know that a node will get disconnected with 80% probability with a $\pm 3\%$ margin of error, then we can set $\mathsf{dp}(n) = [0.77, 0.83]$. One possibility to compute dp in a specific setting is by collecting statistical data on the past failures of each node. This would give us both a mean probability of failure for a given node as well as a standard deviation which would jointly result in a probability interval. In other applications (e.g. where statistics are not available) expert opinions can be used.

Given a network $(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ and a disconnect probability function dp, there is a space of possible networks that may arise in the future.

Definition 4.2 *Suppose* $(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ *is a network and* $\mathcal{N}' \subseteq \mathcal{N}$. *Then* $(\mathcal{N}', \mathsf{edges}', \mathsf{mem})$ *is a possible future network where* $\mathsf{edges}' = \{(n_1, n_2) \mid (n_1, n_2) \in \mathsf{edges}$ *and* $n_1, n_2 \in \mathcal{N}'\}$.

We use $\mathsf{PFN}_{\mathsf{dp}}(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ to denote the set of all possible future networks associated with a network $(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ and a disconnect probability function dp. Note that we can infer probabilities of possible future networks from such disconnect probabilities on nodes. Even though many future networks are possible at a given time t, only one of them will in fact occur at time t. So at time t, $\mathsf{PFN}_{\mathsf{dp}}(\mathcal{N}, \mathsf{edges}, \mathsf{mem})$ represents the space of possible network configurations. Given a network $Ne' = (\mathcal{N}', edges', \mathsf{mem})$ we write $N \in Ne'$ iff $N \in \mathcal{N}'$. Furthermore, since in this paper we do not discuss the failure of edges, for space reasons, we will omit them from the networks in the rest of the paper.

Suppose $\mathbf{prob}(Ne)$ denotes the probability of a possible future network Ne. For any $N \in \mathcal{N}$ we can write the constraint:

$1 - \mathsf{dp}(N).\mathsf{UB} \leq \Sigma_{Ne' \in \mathsf{PFN}_{\mathsf{dp}}(\mathcal{N}, \mathsf{mem})\, \wedge\, N \in Ne'} \mathbf{prob}(Ne')$
$\leq 1 - \mathsf{dp}(N).\mathsf{LB}$.

This constraint says that the sum of the probabilities of all future networks in which node N survives must be between $1-\mathsf{dp}(N).\mathsf{UB}$ and $1-\mathsf{dp}(N).\mathsf{LB}$. We take all such constraints (one for each node) and add a constraint which says that the only possible future networks are those in $\mathsf{PFN}_{\mathsf{dp}}(\mathcal{N}, \mathsf{mem})$. Last, but not least, we know that the probability of each future network is at least 0. This gives us:

$$\Sigma_{Ne' \in \mathsf{PFN}_{\mathsf{dp}}(\mathcal{N},\mathsf{mem})} \mathbf{prob}(Ne) = 1.$$

For any $Ne' \in \mathsf{PFN}_{\mathsf{dp}}(\mathcal{N}, \mathsf{mem})$, $\mathbf{prob}(Ne') \geq 0$.

If $Ne = (\mathcal{N}, \mathsf{mem})$, then $CONS(\mathsf{dp}, Ne)$ denotes the set of all such constraints.[2] We can use $CONS(\mathsf{dp}, Ne)$ to determine the survival probability of a given deployment.

Definition 4.3 *Given a network Ne, a disconnect probability function* dp, *and a deployment μ, we say that the probability of survival of μ is given by the following linear program:*
minimize $\Sigma_{Ne' \in \mathsf{PFN}_{\mathsf{dp}} \wedge \mu \textit{ is a deployment w.r.t. } Ne'} \mathbf{prob}(Ne')$
subject to $CONS(\mathsf{dp}, Ne)$.

The solutions of $CONS(\mathsf{dp}, Ne)$ are possible probabilities of possible future networks arising. Clearly, any of these probability assignments is possible. The objective function above adds the probabilities of all possible future networks where at least one copy of each agent in MAS survives. This expression must be minimized because different solutions of $CONS(\mathsf{dp}, Ne)$ assign different values to this sum - as any of these solutions is possible, the only guarantee we can give about survivability of μ is that it exceeds the minimal such value.

Computing Optimal Deployment (COD) Problem. Given a network Ne, and a disconnect probability function dp, find a deployment whose probability of survival is maximal.

This is the key problem that we will solve.

5 Computing the Survival Probability of a Deployment

A naive way to find the probability of survival of a given μ is to solve the linear program of Definition 4.3 using classical linear programming algorithms [Hiller and Lieberman, 1974; Karmarkar, 1984]. However, the size of the linear program involved in enormous. Our *Compute Deployment Probability* (*CDP*) algorithm will avoid this problem. *CDP* uses a function called Loc which takes an agent a, a network $(\mathcal{N}, \mathsf{mem})$, and a deployment μ as input, and returns the set of all nodes $N \in \mathcal{N}$ such that $\mathsf{a} \in \mu(N)$ as output. One way of pruning the search is to use the following results.

Proposition 5.1 *Suppose* MAS *is a multiagent application and Ne is a network and suppose there is at least one multiagent application deployment for* MAS *on Ne. Further suppose that for all agents* a, $\mathsf{mem}(\mathsf{a}) > 0$. *Then there exists an optimal multiagent deployment μ (i.e. μ has maximal probability of survival) such that for all agents $\mathsf{a}_1, \mathsf{a}_2$, the set of locations of agent a_1 according to μ is not a strict subset of the set of locations of agent a_2 according to μ.*

[2] It is important to note that if, for example, we know that the disconnect probabilities of n_1 and n_2 are independent, then we can expand $CONS(\mathsf{dp}, Ne)$ to include the constraint $\mathbf{prob}(\{n_1, n_2\}) = 1 - \mathsf{dp}(n_1) * \mathsf{dp}(n_2)$. For space reasons, we do not go into this in further detail.

The above result says that when trying to find an optimal multiagent deployment μ, we must ensure that no agent is located in a set of nodes that is a strict subset of the set of nodes that another agent is located in. As we shall see, this property allows us to prune our search a fair amount. Before describing our algorithm, we need to introduce some notation,

- An agent a is *relevant* w.r.t. μ and Ne if there is no other agent which is deployed at a strict subset of nodes at which a is deployed. $\mathsf{ra}(Ne, \mu)$ denotes the set of relevant agents w.r.t. μ, Ne.
- The *necessary nodes* of Ne w.r.t μ is $\mathsf{nn}(Ne, \mu) = \{N \mid \exists \mathsf{a} \in \mathsf{ra}(Ne, \mu), N \in Loc(\mathsf{a}, Ne, \mu)\}$. Nodes in which no relevant agents are deployed are not important.

The following theorem says that survivability is unaffected if we get rid of unnecessary nodes.

Theorem 5.1 *Suppose* MAS *is a multiagent application, $Ne = (\mathcal{N}, \mathsf{mem})$ is a network,* dp *is a disconnect probability function and μ is a feasible multiagent application deployment for* MAS *on Ne. Let $Ne' = (\mathcal{N}', \mathsf{mem})$ where $\mathcal{N}' = \mathsf{nn}(Ne, \mu)$. If μ' is the restriction of μ to Ne', then* $\mathsf{surv}(\mu) = \mathsf{surv}(\mu')$.

Proof Sketch. It is easy to see that it is enough to show the claim for the case that only one node is removed from $\mathcal{N}$ when constructing Ne'. Without loss of generality, let us assume that $\mathcal{N}' = \mathcal{N} \setminus \{N_1\}$. We make the following observations:

- It can be shown that $\mathcal{N}'' \subseteq \mathcal{N}'$ is feasible w.r.t μ' iff $\mathcal{N}'' \cup \{N_1\}$ is also feasible w.r.t μ.
- Let $n = |\mathcal{N}|$. $CONS(\mathsf{dp}, Ne)$ consists of n equations, one for each node. $CONS(\mathsf{dp}, Ne')$ does not include an equation for N_1 and thus includes $n-1$ equations.
- We use $\mathbf{prob}$ (resp. $\mathbf{prob}'$) to denote the probability function in $CONS(\mathsf{dp}, Ne)$ (resp. $CONS(\mathsf{dp}, Ne')$). Consider the equations w.r.t. $N_i \in \mathcal{N}$, $N_i \neq N_1$. For both Ne and Ne', both equations have the same left side, viz. $1 - \mathsf{dp}(N_i)$. The right side of the relevant equation in $CONS(\mathsf{dp}, Ne)$ consists of 2^{n-1} elements of the form $\mathbf{prob}(\{N_i\} \cup \mathcal{N}'')$, $\mathcal{N}'' \subseteq \mathcal{N} \setminus \{N_i\}$. In $CONS(\mathsf{dp}, Ne')$ the corresponding equation consists of 2^{n-2} elements of the form $\mathbf{prob}'(\{N_i\} \cup \mathcal{N}''')$, $\mathcal{N}''' \subseteq \mathcal{N}' \setminus \{N_i\}$. Thus for each element in an equation of $CONS(\mathsf{dp}, Ne')$ of the form $\mathbf{prob}'(\mathcal{N}'')$ there are exactly two terms in the corresponding equation in $CONS(\mathsf{dp}, Ne)$: one of the form $\mathbf{prob}(\mathcal{N}'')$ and the other of the form $\mathbf{prob}(\mathcal{N}'' \cup \{N_1\})$.
- Similarly, if the minimization expression with respect to Ne' is of length (i.e. number of terms) k, then the minimization expression with respect to Ne is $2k$. In particular, for each $\mathbf{prob}'(\mathcal{N}'')$ there are two terms in the minimization expression of the form $\mathbf{prob}'(\mathcal{N}'')$ and $\mathbf{prob}'(\mathcal{N}'' \cup \{N_1\})$.

We are now ready to prove our claim. Suppose the minimization problem is solved with respect to Ne. We set $\mathbf{prob}'(\mathcal{N}'') = \mathbf{prob}(\mathcal{N}'') + \mathbf{prob}(\mathcal{N}'' \cup \{N_1\})$. It is easy to see that based on our observations that $CONS(\mathsf{dp}, Ne')$

will be satisfied and the values will minimize the relevant expression.

Suppose the minimization problem is solved with respect to Ne'. In this case, we add the following equations to $CONS(\mathsf{dp}, Ne)$: (i) $\mathbf{prob}'(\mathcal{N}'') = \mathbf{prob}(\mathcal{N}'') + \mathbf{prob}(\mathcal{N}'' \cup \{N_1\})$ (ii) We replace each expression in the minimization expression of the form $\mathbf{prob}(\mathcal{N}'') + \mathbf{prob}(\mathcal{N}'' \cup \{N_1\})$ by $\mathbf{prob}'(\mathcal{N}'')$. It is easy to see that the minimization expression is identical to the one associated with Ne' and all the constraints of $CONS(\mathsf{dp}, Ne)$ except the first one are identical to those of $CONS(\mathsf{dp}, Ne')$ and are satisfied. Hence, it is left to show that the constraint associated with N_1 is satisfied.

This constraint is of the form: $1 - \mathsf{dp}(N_1) = \mathbf{prob}(\{N_1\}) + \mathbf{prob}(\{N_1, N_2\}) + ... + \mathbf{prob}(\{N_1, ..., N_n\})$. We replace any term $\mathbf{prob}(\mathcal{N}'' \cup \{N_1\})$, $\mathcal{N}'' \subseteq \mathcal{N}'$ by $\mathbf{prob}'(\mathcal{N}'') - \mathbf{prob}(\mathcal{N}'')$. The sum of all the $\mathbf{prob}'(\mathcal{N}'')$ from the last constraint of $CONS(\mathsf{dp}, Ne')$ is equal to 1. Thus, we get that $\sum_{\mathcal{N}'' \subseteq \mathcal{N}'} \mathbf{prob}(\mathcal{N}'') = \mathsf{dp}(N_1)$ where $0 \leq \mathbf{prob}(\mathcal{N}'') \leq 1$. As $0 \leq \mathsf{dp}(N_1) \leq 1$, this equation is solvable. □

The net impact of this theorem is that only necessary nodes need to be considered. We demonstrate this using Example 2.1.

Example 5.1 *Consider the deployment of Example 2.1. Agent d is deployed at nodes $\{n_2, n_3, n_4\}$ and c is deployed at nodes $\{n_2, n_3\}$. Clearly, c is deployed at a strict subset of nodes at which d is deployed. In order for the deployment to survive in a given possible future network one of the nodes on which c is located, n_2 or n_3 must stay connected. But then d will also be deployed in the new network. However, if n_4 stays connected in a future network, but neither n_2 nor n_3 stay connected, the deployment will not survive. Thus, based on theorem 5.1 when computing the survivability of the deployment, there is no need to consider d and n_4.*

We are now ready to use the above theorem to formulate our algorithm *CDP* to compute probability of survival of a deployment. Our CDP algorithm will use the well known notion of a *hitting set.*

Definition 5.1 *Suppose $S = \{S_1, \ldots, S_n\}$ is a set of sets. A hitting set for S is any set $h \subseteq \bigcup_{i=1}^{n} S_i$ such that: (1) for all $1 \leq i \leq n$, $h \cap S_i \neq \emptyset$ and (2) there is no strict subset $h' \subset h$ satisfying condition (1) above.* **HitSet***(S) denotes the set of all hitting sets of S.*

HitSet can be implemented in any number of standard ways prevalent in the literature. We will focus on the following hitting sets with respect to a given network.

Definition 5.2 *Suppose* MAS *is a multiagent application, $Ne = (\mathcal{N}, \mathsf{mem})$ is a network and μ is a feasible multiagent application deployment for* MAS *on Ne. The set of hitting sets with respect to* MAS, *Ne and μ is* $\mathsf{hs}(Ne, \mu, \mathsf{MAS}) = \mathbf{HitSet}(\{Loc(\mathsf{a}, Ne, \mu) \mid \mathsf{a} \in \mathsf{MAS}\})$.

Intuitively, the hitting sets above describe minimal sets of nodes that must be present in a possible future network in order for the multiagent application to survive. We will use hitting sets to determine whether a deployment w.r.t. Ne can be a deployment w.r.t. a possible future network. This intuition leads to the following algorithm *CDP*.

Algorithm 5.1 (CDP(Ne,dp,MAS,μ))

($\star$ **Input**:*(1) A network* $Ne = (\mathcal{N}, \mathsf{mem})$ $\star$)
($\star$ *(2) a disconnect probability function* dp, $\star$)
($\star$ *(3) a multiagent application* MAS $\star$)
($\star$ *(4) a feasible deployment μ and* $\star$)
($\star$ **Output**: *the survivability of μ.* $\star$)

1. $PossNe = \emptyset$; $\mathsf{MAS}' = \mathsf{ra}(Ne, \mu)$; $\mathcal{N}' = \mathsf{nn}(Ne, \mu)$;
2. $H = \mathsf{hs}(Ne', \mu', \mathsf{MAS}')$;
3. **For** *any* $\mathcal{N}'' \in 2^{\mathcal{N}'}$ **do**
   (a) $temp = H$; $flag = true$;
   (b) **While** $temp \neq \emptyset$ *and* $flag$ **do**
      i. $h = headof(temp)$; $temp = temp \setminus h$;
      ii. **If** $h \subseteq \mathcal{N}''$ **then do**
         A. $PossNe = PossNe \cup \{(\mathcal{N}'', \mathsf{mem})\}$;
         B. $flag = false$;
4. **Return** *the result of the following linear program:*
   minimize $\Sigma_{Ne'' \in PossNe} \mathbf{prob}(Ne'')$
   subject to $Cons(\mathsf{dp}, Ne')$.

CDP works by first focusing on the necessary nodes. Then, for each agent $\mathsf{a} \in \mathsf{MAS}'$, all nodes where that agent is located are identified. It then computes all hitting sets of these nodes. For any possible future network it checks whether one of the hitting sets is a subset of the nodes of the network. It is easy to see that *CDP* is exponential in the number of the necessary nodes. The following example illustrates the working of this algorithm.

Example 5.2 *Consider the network, Ne and the deployment μ of example 2.1. Further, assume that* $\mathsf{dp}(n_1) = 0.1, \mathsf{dp}(n_2) = 0.2, \mathsf{dp}(n_3) = 0.3$ *and* $\mathsf{dp}(n_4) = 0.4$. *In the first step CDP sets* $\mathsf{MAS}' = \{a, c\}$, $\mathcal{N}' = \{n_1, n_2, n_3\}$.

There are 8 possible sets of nodes to be considered for future networks: $\{\mathcal{N}_1 = \{n_1\}, \mathcal{N}_2 = \{n_2\}, \mathcal{N}_3 = \{n_3\}, \mathcal{N}_4 = \{n_1, n_2\}, \mathcal{N}_5 = \{n_1, n_3\}, \mathcal{N}_6 = \{n_2, n_3\}, \mathcal{N}_7 = \{n_1, n_2, n_3\}, \mathcal{N}_8 = \emptyset\}$. *We denote the associated networks by* $Ne_1, Ne_2, ..., Ne_8$.

In Step 2, CDP sets $H = \{h_1, h_2\}$ where $h_1 = \{n_1, n_2\}$ and $h_2 = \{n_3\}$. In step 3b it checks which of the node sets of possible future networks are supersets of a hitting set of H and will also set $PossNe = \{Ne_3, Ne_4, Ne_5, Ne_6, Ne_7\}$.

Denote $\mathbf{prob}(Ne_i)$ with p_i. The linear program to be solved in step 4 is:
$Minimize$ $p_3 + p_4 + p_5 + p_6 + p_7$ *subject to the constraints:*
(1) $p_1 + p_4 + p_5 + p_7 = 0.9$
(2) $p_2 + p_4 + p_6 + p_7 = 0.8$
(3) $p_3 + p_5 + p_6 + p_7 = 0.7$
(4) $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 1$
(5) $p_i \geq 0$.
Using the linear programming tool Lindo, we found the following results: $p_7 = 0.7$, $p_1 = 0.2$, $p_2 = 0.1$, $p_3 = p_4 = p_5 = p_6 = p_8 = 0$ which yields the minimum value of 0.7 for the objective function.

Example 5.3 *In example 5.2, there were no constraints on the dependencies of the disconnect probabilities of the nodes. Suppose we know, in addition, that the probability that both* *nodes n_1, n_3 get disconnected is* 0.05, *i.e.* $\mathsf{dp}(\{n_1, n_3\}) = 0.05$. *In this case we should consider all possible future networks whose set of nodes is not a superset of* $\{n_1, n_3\}$. *Those sets are* $\mathcal{N}_2$ *and* $\mathcal{N}_8$ *of example 5.2. Thus, the new constraint is* $p_2 + p_8 = 0.05$.

If we run the linear program of example 5.2 again with the additional constraint, the results are as follows: $p_4 = 0.05$, $p_6 = 0.05$, $p_7 = 0.65$, $p_1 = 0.2$, $p_2 = 0.05$, $p_5 = p_3 = p_8 = 0$ *which yields the minimum value of 0.75 for the objective function.*

6 Computing Optimal Deployments

We are now ready to develop algorithms to find an optimal multiagent deployment. We first present the **COD** algorithm to compute optimal deployments. We also present two heuristic algorithms, **HAD**1 and **HAD**2, which may find suboptimal deployments (but do so very fast).

6.1 The COD Algorithm

One may wonder if **COD** can be solved via a classical problem such as facility location problem (FLP)[Shmoys *et al.*, 1997]. In FLP, there are a set of facility locations and a set of consumers. The task is to determine which facilities should be used and which customers should be served from which facility so as to minimize some objective function (e.g. sum of travel times for customers to their assigned facility). One may think that we can directly use FLP algorithms to solve **COD** - unfortunately, this is not true.

Theorem 6.1 *The problem of checking if a MAD-deployment* μ *is optimal is* NP^{NP}*-hard.*

This theorem says that even if we have a polynomial oracle to solve NP-complete problems, checking if a MAD-deployment is optimal is still NP-hard ! In fact, it is easy to reduce the facility location problem to that of finding an optimal deployment. Even if we have an oracle for facility location, the MAD-deployment problem is still NP-hard.

Computing an optimal MAD-deployment involves two sources of complexity. The first is the exponential space of possible deployments. The second is that even if we have a given deployment, finding its probability of survival is exponential.

Thus, to solve **COD** exactly, we do a state space search where the initial state places all agents in a multiagent application MAS on all nodes of the network Ne. If this placement is a deployment, then we are done. Otherwise, there are many ways of removing agents from nodes and each such way leads to a possible deployment. The value of a state is the survivability of the state, which can be computed using the CDP algorithm. As soon as a deployment is found, we can bound the search using the value of that deployment. The reason is that given any state in the search, all states obtained from that state by removing one or more agent has a lower survivability than the original state. Before presenting the **COD** algorithm, we first present the SEARCH routine used by it.

Algorithm 6.1 SEARCH *(*Ne*,* dp*,* MAS*,* μ*,* $best_\mu$*,* $best_{val}$*)*

(⋆ **Input**: *(1) A network* $Ne = (\mathcal{N}, \mathsf{mem})$ ⋆)
(⋆ *(2) a disconnect probability function,* dp ⋆)
(⋆ *(3) a multiagent application,* MAS ⋆)
(⋆ *(4) a deployment,* μ ⋆)
(⋆ *(5) best deployment found so far,* ⋆)
(⋆ $best_\mu$ *(global variable)* ⋆)
(⋆ *(6) best survivability value found thus far,* ⋆)
(⋆ $best_{val}$ *(global variable)* ⋆)
(⋆ **Output**: *The procedure changes the global variables* ⋆)
(⋆ $best_\mu$*, and* $best_{val}$ ⋆)

1. $prob(\mu)$ =*CDP(*Ne*,* dp*,* μ*,* MAS*);*
2. **if**$(best_{val} < prob(\mu))$ **then do**
   (a) **if** μ *satisfies the* mem *constraints* **then do**
      i. $best_\mu = \mu$
      ii. $best_{val} = prob(\mu)$
   (b) **else do**
      i. **For** *any* $N \in \mathcal{N}$ **do**
      A. **For** *any* a $\in \mu(N)$ **do**
         - $temp_\mu = \mu$;
         - $temp_\mu(N) = temp_\mu(N) \setminus \{\mathsf{a}\}$
         - **if** $temp_\mu$ *is a deployment* **then** $search(Ne, \mathsf{dp}, \mathsf{MAS}, temp_\mu, best_\mu, best_{val})$

We are now ready to present the **COD** algorithm.

Algorithm 6.2 **COD***(*Ne*,* dp*,* MAS*)*

(⋆ **Input**: *(1) A network* $Ne = (\mathcal{N}, \mathsf{mem})$ ⋆)
(⋆ *(2) a disconnect probability function* dp*,* ⋆)
(⋆ *(3) a multiagent application* MAS ⋆)
(⋆ **Output**: *an optimal deployment* ⋆)

1. $best_\mu = null;\ best_{val} = 0$
2. **For any** $N \in \mathcal{N}$ **do** $\mu(N) = \mathsf{MAS}$;
3. $search(Ne, \mathsf{dp}, \mathsf{MAS}, \mu, best_\mu, best_{val})$;
4. **return** $best_\mu$

The correctness of **COD** depends on the correctness of bounding the search in step 2 of algorithm 6.1. We present the correctness result below.

Theorem 6.2 *Suppose* MAS *is a multiagent application,* $Ne = (\mathcal{N}, \mathsf{mem})$ *is a network and* dp *is a disconnect probability function. Then* $\mathbf{COD}(Ne, \mathsf{dp}, MAS)$ *returns an optimal deployment of* MAS *on* Ne.

The astute reader may notice that CDP is computed for every placement. Many of these placements are very similar to each other. Hence, one may wonder whether it is possible to use the results of computing CDP applied to a previous placement to a placement that is very similar to the previous placement. The two propositions below show that this can be done.

Proposition 6.1 *Suppose* MAS *is a multiagent application,* $Ne = (\mathcal{N}, \mathsf{mem})$ *is a network and* dp *is a disconnect probability function. Suppose the placement* μ' *was obtained from the placement* μ *in step 2(b)iA of algorithm 6.1 and suppose* $Ne' = (\mathcal{N}', \mathsf{mem})$*, such that* $\mathcal{N}' = \mathsf{nn}(Ne, \mu')$*. Then the set of hitting sets with respect to* μ' *and* Ne' *is a subset of the set of hitting sets with respect to* μ *and* Ne*. That is,* $\mathsf{hs}(Ne', \mu', \mathsf{MAS}) \subseteq \mathsf{hs}(Ne, \mu, \mathsf{MAS})$.

The following example demonstrate a situation where $\mathsf{hs}(Ne', \mu', \mathsf{MAS}) \subset \mathsf{hs}(Ne, \mu, \mathsf{MAS})$.

Example 6.1 *Suppose* $\mathcal{N} = \{n_1, n_2, n_3\}$, $\mathsf{MAS} = \{a, b, c\}$ *and the deployment* μ *is as follows:* $\mu(n_1) = \{a, c\}$, $\mu(n_2) = \{a, b\}$ *and* $\mu(n_3) = \{b\}$. *Thus,* $Loc(a, Ne, \mu) = \{n_1, n_2\}$, $Loc(b, Ne, \mu) = \{n_2, n_3\}$, $Loc(c, Ne, \mu) = \{n_1\}$. *The hitting sets are,* $h_1 = \{n_1, n_2\}$, $h_2 = \{n_1, n_3\}$. *If we remove agent* a *from node* n_1, *then the set* h_2 *is no longer a hitting set.*

Proposition 6.2 *Suppose* MAS *is a multiagent application,* $Ne = (\mathcal{N}, \mathsf{mem})$ *is a network and* dp *is a disconnect probability function. Suppose the placement* μ' *was obtained from the placement* μ *in step 2(b)iA of algorithm 6.1 and suppose* $Ne' = (\mathcal{N}', \mathsf{mem})$, *such that* $\mathcal{N}' = \mathsf{nn}(Ne, \mu')$. *Suppose there is* $h \in \mathsf{hs}(Ne, \mu, \mathsf{MAS})$ *such that* $N \in h$ *but* $N \notin \mathcal{N}'$. *Then,* $h \notin \mathsf{hs}(Ne, \mu', \mathsf{MAS})$.

We use the above propositions to give a new version, CDP1, of the CDP algorithm.

Definition 6.1 *Suppose* MAS *is a multiagent application,* $Ne = (\mathcal{N}, \mathsf{mem})$ *and* μ *is a deployment. A set* $h \subseteq \mathcal{N}$ *supports an agent* a $\in$ MAS *if there is* $N \in h$ *such that* a $\in \mu(N)$.

CDP1 is now defined below.

Algorithm 6.3 CDP1*(*Ne*,*dp*,* MAS*,*μ*,*H_p*,*a**,* N*)]*

(⋆ **Input**: *(1-4) as in CDP (algorithm 5.1.* ⋆)
(⋆ *(5) a set of hitting sets* H_p ⋆)
(⋆ *(6) an agent* a$^* \in$ MAS ⋆)
(⋆ *(7) a node* $N \in \mathcal{N}$ ⋆)
(⋆ **Output**: *(1) The survivability of* μ. ⋆)
(⋆ *(2) the set of hitting sets associated with* μ' ⋆)

Step 1 as in CDP.

2. *If* $H_p \neq \emptyset$ **then do**
   (a) $H = \emptyset$;
   (b) **For any** $h \in H_p$ **do**
   If $N \notin h$ *or* $h \setminus \{N\}$ *supports* a* **then** $H = H \cup (h \cap \mathcal{N}')$;
3. **else** $H = \mathsf{hs}(Ne', \mu', \mathsf{MAS}')$;
4. **For** *any* $\mathcal{N}'' \in 2^{\mathcal{N}'}$ **do**
   (a) $temp = H$; $flag = true$;
   (b) **While** $temp \neq \emptyset$ *and* $flag$ **do**
      i. $h = headof(temp)$; $temp = temp \setminus h$;
      ii. **If** $h \subseteq \mathcal{N}''$ **then do**
         A. $PossNe = PossNe \cup \{(\mathcal{N}'', \mathsf{mem})\}$;
         B. $flag = false$;
5. *Assign to* p *the result of the following linear program:*
   minimize $\Sigma_{Ne'' \in PossNe}\mathbf{prob}(Ne'')$
   subject to $Cons(\mathsf{dp}, Ne')$.
6. **Return** p *and* H;

The SEARCH and **COD** algorithms need to be modified in a straightforward way to use CDP1 - we do not go through the details for space reasons.

6.2 Heuristic Algorithms

In this section, we describe two fast heuristic algorithms, **HAD**1, **HAD**2. **HAD**1 iteratively solves knapsack problems [Cormen *et al.*, 1990] by trying to pack nodes with low disconnect probability first.

Algorithm 6.4 **HAD**1 *(*Ne*,*dp*,*MAS*)*

(⋆ **Input**: *As in algorithm* **COD** *(6.2).* ⋆)
(⋆ **Output**: *a deployment* ⋆)

1. $\mu = \emptyset$; *flag=true;*
2. **For all** $N \in \mathcal{N}$ **do** $res\_avail(N) = \mathsf{mem}(N)$;
3. **While** $flag$ **do**
   (a) $nodes = \mathcal{N}$; $agents = \mathsf{MAS}$; $flag = false$;
   (b) **While** *(*$nodes \neq \emptyset$*)* **do**
      i. **if** $agents = \emptyset$ **then** $agents = \mathsf{MAS}$;
      ii. $N = argmin_{\mathsf{dp}} nodes$;
      iii. $agents_{dep} = knapsack(N, \mathsf{mem\_}avail(N), agents \setminus \mu(N))$;
      iv. **if** $agents_{dep} \neq \emptyset$ **then** $flag = true$;
      v. $\mu(N) = \mu(N) \cup agents_{dep}$;
      vi. $res\_avail(N) = res\_avail(N) - \sum_{\mathsf{a} \in agents_{dep}} \mathsf{mem}(\mathsf{a})$;
      vii. $agents = agents \setminus agents_{dep}$;
      viii. $nodes = nodes \setminus \{N\}$;

The **HAD**2 algorithm is based on the intuition that we should first locate agents with high resource requirements, and then deal with agents with low resource requirements. Thus, we sort agents in ascending order according to resource requirements, place them, then go to agents with the second highest resource requirements, and so on. If at the end resources are still available, then we make replicas of the agents.

Algorithm 6.5 **HAD**2*(*Ne*,* dp*,* MAS*)*

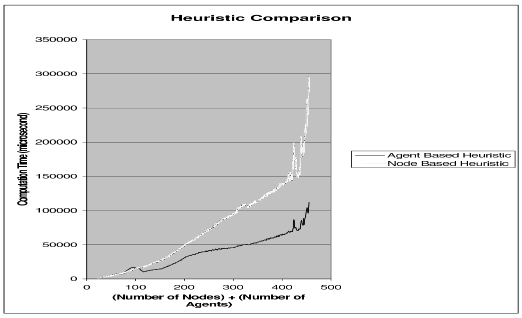
(⋆ **Input** *and* **Output** *as in node-based-heuristic (HAD1)* ⋆)

1. $\mu = \emptyset$; *flag=true;*
2. **For all** $N \in \mathcal{N}$ *res_avail(N)=*mem*(N);*
3. **While** $flag$ **do**
   (a) $nodes = \mathcal{N}$; $agents = \mathsf{MAS}$; $flag = false$;
   (b) **While** *(*$agents \neq \emptyset$*)* **do**
      i. a$= argmax_{\mathsf{mem}} agents$
      ii. $nodes_{poss} = \{N \mid \mathsf{mem}(\mathsf{a}) \leq res\_avail(N)\}$;
      iii. **if** $nodes_{poss} \neq \emptyset$ **then do**
         A. $flag = true$;
         B. $N = argmax_{\mathsf{dp}} nodes_{poss}$;
         C. $\mu(N) = \mu(N) \cup \{\mathsf{a}\}$;
         D. $res\_avail(N) = res\_avail(N) - \mathsf{mem}(\mathsf{a})$
         E. $agents = agents \setminus \{\mathsf{a}\}$;

7 Experiments

We have implemented all the algorithms described in this paper. For space reasons, we only present experimental results on the heuristics for computing optimal deployments.

In our experiments, we varied the number of agents and the number of nodes. For each combination of agents and nodes, we ran several trials. In each trial, we randomly generated the memory available on each node, the node's disconnect probability, and the memory required for one copy of each agent. The experiments were conducted on a Linux box, using Red Hat 7.2 (Enigma). In all the experiments the number of nodes+ agents varied between 1 and 500. We randomly



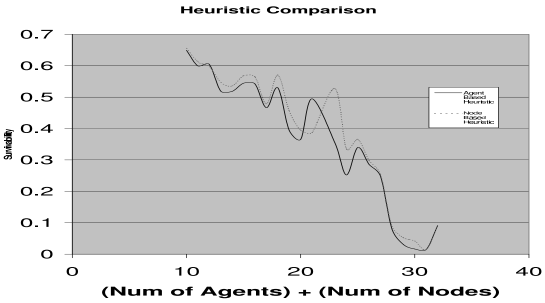


Figure 1: Heuristic comparision: Top figure:-Computation time (in miliseconds) as the function of the sum of the number of nodes and agents. Bottom figure: survivability as a function of the sum of nodes and agents. agents. The lighter line and the darker line refer to the nodes-based and agent-based heuristics, respectively.

generated numbers between 0 and 0.5 as the disconnect probability of each node. The sizes of the agents were uniformly distributed between 3 and 9 (units of memory) and the sizes of the nodes were uniformly distributed between 5 and 30 (units of memory).

The top graph of figure 7 demonstrates the time efficiency of both heuristics: they can find a deployment for 500 agents and sites in under a second.

When comparing $\mathbf{HAD}1$ and $\mathbf{HAD}2$, we noticed that (see bottom graph of figure 7): (1) as the sum of the number of agents and nodes increases, the survivability decreases. (2) The node based heuristic almost always finds better deployments than the agent-based heuristic. In addition, (3) When there are more agents than nodes, the node based heuristic will require less time, while when there are more nodes than agents the agent based heuristic will take less time.

The intuition behind observation (1) is as follows. As the number of agents increases, it becomes more difficult to maintain the feasibility of the system, and thus survivability decreases. In addition, when the number of nodes increases, there are more possible future networks and thus, the probability that there will be one network with a low probability increases. Since, the survivability depends on the worst case, its value decreases.

8 Conclusions

As more and more agents are deployed in mission critical commercial, telecommunications, business, and financial applications, there is a growing need for guarantees that such multiagent applications will survive various kinds of catastrophes. The scope of the problem is so vast that any one paper can only make a small dent in this very important problem.

In this paper, we have carved out such a small piece of the problem. Specifically, we study the problem of how to deploy multiple copies of agents in a MAS on nodes so that the probability of survivability of the MAS is maximized. We provide a formal, mathematical model for probabilistic MAS survivability, and develop an optimal algorithm for this purpose as well as some heuristic algorithms. We have conducted experiments showing the effectiveness of the approach.

References

[Cormen *et al.*, 1990] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[Fan, 2001] X. Fan. On splitting and cloning agents, 2001. Turku Center for Computer Science, Tech. Reports 407.

[Fedoruk and Deters, 2002] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *Proceedings AAMAS-02*, pages 737–744, Bologna, Italy, 2002.

[Gartner, 1999] F. C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, 1999.

[Gutjahr, 1998] W. J. Gutjahr. Reliability optimization of redundant software with correlate failures. In *The 9th Int. Symp. on Software Reliability Engineering*, Germany, 1998.

[Hiller and Lieberman, 1974] F. S. Hiller and G. J. Lieberman. *Operations Research*. Holden-Day, San Francisco, 1974.

[Karmarkar, 1984] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–396, 1984.

[Kumar *et al.*, 2000] S. Kumar, P.R. Cohen, and H.J. Levesque. The adaptive agent architecture: achieving fault-tolerance using persistent broker teams. In *Proc. of ICMAS*, 2000.

[Lyu and He, 1993] M. Lyu and Y. He. Improving the n-version programming process through the evolution of a design paradigm. *IEEE Trans. Reliability*, 42(2), 1993.

[Marin *et al.*, 2001] O. Marin, P. Sens, J. Briot, and Z. Guessoum. Towards adaptive fault tolerance for distributed multi-agent systems. In *Proceedings of ERSADS*. 2001.

[Mishra, 2001] S. Mishra. Agent fault tolerance using group communication. In *Proc. of PDPTA-01*, NV, 2001.

[Shehory *et al.*, 1998] O. Shehory, K. P. Sycara, P. Chalasani, and S. Jha. Increasing resource utilization and task performance by agent cloning. In *Proc. of ATAL-98, pp 413-426*, 1998.

[Shmoys *et al.*, 1997] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proc. of STOC-97*, pages 265–274, 1997.