Approximation results for probabilistic survivability*

Yingqian Zhang
University of Manchester
zhangy@cs.man.ac.uk

Efrat Manister, Sarit Kraus
Bar-Ilan University
{manister,sarit}@macs.biu.ac.il

V.S. Subrahmanian
University of Maryland
vs@cs.umd.edu

Abstract

As multiagent systems (MASs) are increasingly used in industrial applications, the need to make them more robust and resilient against disruption increases dramatically. Kraus et al. [1] has developed a probabilistic model (assuming complete ignorance of dependencies between node failures) of survivability based on deploying each agent in a MAS on one or more nodes. Finding a deployment that maximizes survivability is highly intractable for two reasons: firstly, computing the survivability of any deployment is intractable, and secondly, going through an exponential number of deployments to find the best one adds another layer of intractability. In this paper, we study what happens when node failures are independent. We show that computing survivability in this environment is still intractable. We propose various heuristics to compute the survivability of a given deployment. We have implemented and tested all these heuristics. We report on the advantages and disadvantages of different heuristics in different environmental settings.

1. Introduction

As the use of Multiagent Systems (MASs) becomes more and more widespread, there is a growing need to ensure that MASs are robust under attack. This can be ensured in many ways. One way is to ensure that security protocols are built into MASs so that they do *not* get compromised. Another way is to ensure that when a network node on which an agent is located goes down, that does not cause the MAS itself to go down. This latter property can be ensured by replicating the agents wisely across the network. This idea was first proposed by [3] and later studied by [1].

This paper builds upon work of [1] where a probabilistic model of survivability is proposed. They assume that each agent a in an MAS M has a *size* $\mathsf{size}(a)$ and that there is a fully connected overlay network of nodes. Each node n has a size $\mathsf{size}(n)$ denoting the amount of memory that that node makes available to the MAS M for storing agents: $\mathsf{size}(n)$ can be zero, it can be small or large as the owner of that node wishes. In addition, each node has an associated *disconnect probability* that denotes the probability that node will "go down" within a finite window of time. A *deployment* μ of the MAS M takes a node as input, and tells us which agents in M are located at that node. For a deployment to be valid, the space constraints of each node must be satisfied (i.e. the sum of the sizes of the agents placed at that node by the deployment must not exceed the size of the node). In addition, each agent of M must be placed somewhere. [1] uses a linear programming model to define the probability of survival of μ under the assumption that we are completely *ignorant* about node failure dependencies. However, this assumption is not always valid—for example, if there is an attack on US government computers, it is unlikely that UNESCO computers are under attack as well. The events "UMD computers go down" and "UNESCO computers go down" are probably independent. Likewise, because of the ignorance assumption of [1], survival probabilities tend to be extraordinarily pessimistic (low). [1] show that finding an optimal deployment is intractable under their assumptions and provide an exponential algorithm to compute the survivability of a given μ and double exponential algorithms to find a deployment that maximizes the probability of survival. They provide heuristics to find suboptimal deployments in polynomial time.

As we can see above, there are many cases where the ignorance assumption of [1] is not appropriate and the independence assumption is valid. In this paper, we make the following contributions under the independence assumptions:

1. We show that even if we assume independence, finding an optimal deployment and computing the survivability of a given deployment are both NP-hard.
2. We show that if we use a polynomial approximation to find a suboptimal deployment, there will be instances where the polynomial approximation says the survivability of a deployment is 0, when in fact the true survivability is 1. Thus, polynomial approximations are guaranteed to find at least one terrible solution.
3. We provide two survivability functions $SF1_n$ and $SF1_a$. $SF1_n$ is exponential in the number of nodes, while $SF1_a$ is exponential in the number of agents.
4. We provide five approximations that underestimate the survivability of a deployment under the independence assumption.

* Sarit Kraus is also affiliated with University of Maryland. This work is supported in part by NSF grant #0222914, ISF grant #8008, ARO grant DAAD190310202, ARL grants DAAD190320026 and DAAL0197K0135, AFOSR grant FA95500510269, NSF grants IIS0329851 and 0205489, UC Berkeley contract number SA451832441 (subcontract from DARPA's REAL program), and EPSRC grant GR/R57843/01.

5. We describe results of an exhaustive set of experiments we conducted to assess both the computation time and the quality of solutions found by the above approximations. We identify the situations under which different approximations work well.

2. Related work

The file allocation problem [2, 4, 8] tries to place a single file in a location that minizes read/write and/or communications cost. Johnson et al. [9] add an additional constraint that forces every object to have at least t copies on different nodes so that the solutions tolerate up to $(t-1)$ failures. Our problem differs from theirs in two ways: first, we optimize survival probability and second, we ensure that an entire set of agents survive rather than just one file.

[3, 14] clone and merge agents to support load balancing. Fan [5] furnishes each local agent with the capability of load-balancing. He proposes a BDI mechanism to formally model agent cloning for balancing agent workload. Fedoruk and Deters [6] hide agent replication methods inside each agent. Mishra and Huang [13, 12] develop three protocols for recovering node and communication failures. Marin et al. [11] develop a framework to design reliable distributed applications. They use simulations to assess migration and replication costs. Kumar et al. [10] replicate broker agents to ensure that they are available when system failures occur. They use teamwork theory to specify how to recover from broker failure. The RecoMa system [7] uses multiple servers to support matching agents to computers. None of these works assume a probabilistic failure model for nodes and they do not optimize a probabilistic objective function that captures survivability.

3. Survivability of MASs

We now quickly recapitulate the concepts in [1]. A MAS M is a finite set of agents—these agents can be written in any language. We assume the existence of a network $Ne = (V, E)$ where V is the set of nodes in the network and $E = V \times V$, i.e. Ne is a *fully* connected *overlay* network. [1] A *deployment w.r.t.* M, Ne is a mapping μ from V to 2^M such that:

1. For each $n \in V$, $\Sigma_{a \in \mu(n)} \mathsf{size}(a) \leq \mathsf{size}(n)$.
2. For each $a \in M$, there exists a $n \in V$ such that $a \in \mu(n)$.

$\mu(n)$ specifies the set of agents at node n.

1 This is a reasonable assumption as it does not require full connectivity of the underlying physical network (just that all nodes in the physical network are reachable - perhaps through multiple physical links - from all other nodes).

Throughout the rest of this paper, we will assume that M is an arbitrary but fixed MAS, and that Ne is an arbitrary, but fixed network. As a consequence, we will just say "deployment" instead of "deployment w.r.t. M, Ne".

A *disconnect probability function* (dp for short) is a mapping $dp : V \rightarrow [0, 1]$ that assigns to each node $n \in V$, a probability of "going down" or somehow being disconnected from the network.

Example 1 *Consider a fully connected network with* $V = \{n_1, n_2, n_3\}$ *and suppose the disconnect probability function* dp *is given by:* $dp(n_1) = 0.7, dp(n_2) = 0.6, dp(n_3) = 0.4$. *A deployment* μ *is given by:* $\mu(n_1) = \{a_1, a_2, a_3\}, \mu(n_2) = \{a_1\}, \mu(n_3) = \{a_2, a_3\}$.

We are now ready to define a *survivability function*.

Definition 3.1 *A survivability function* SF *is a mapping that takes as input, a deployment function* μ *and a disconnect probability function* dp*, and returns the probability with which it is guaranteed that the MAS will survive, i.e., at least one copy of each agent will be accessible.*

This generalizes the probabilistic definition of survivability given in [1] who makes the strong assumption that we are ignorant of any dependencies between nodes.

4. Complexity results for survivability under independence assumption

Any node in V can "go down" or somehow get "disconnected" from the network. Thus, any $(N, N \times N)$ where $N \subseteq V$ is a possible network that can arise in the future. The survivability of N (*under independence*) is given by

$$surv(N) = \prod_{n_p \in N} (1 - dp(n_p)) \cdot \prod_{n_q \in V \setminus N} dp(n_q). \quad (1)$$

We say that μ *is valid* w.r.t. N iff for each agent a, $\{n \mid a \in \mu(n)\} \cap N \neq \emptyset$. Suppose $ValidN(\mu) = \{N_i \mid N_i \subseteq V$ and μ is valid w.r.t $N_i\}$. Then the probability of survival of μ is given by $\Sigma_{N_i \in ValidN(\mu)} surv(N_i)$.

Finding the probability of survival of a deployment is at least NP-hard even if we make the independence assumption. Similarly, finding an optimal deployment is at least NP-hard.

Proposition 4.1 *The problem of computing the survival probability of a given deployment under the independence assumption and the problem of finding an optimal deployment are at least* NP*-hard.*

One may be tempted to believe that we can find an approximation algorithm that is guaranteed to terminate in polynomial time and give a deployment whose survival probability is within ϵ of the survival probability of the optimal deployment ($\epsilon > 0$). Unfortunately, the best such ϵ is 1 (under the assumption that $P \neq$ NP.)

Theorem 4.2 *If $P \neq NP$, then for each polynomial algorithm to compute a sub-optimal deployment, there are instances in which the optimal deployment's survival probability is 1, but the algorithm returns a deployment with survival probability 0.*

Proof. Suppose that the claim above is not correct. Then there exists a polynomial algorithm AL that always returns a deployment with survival probability larger than 0, when the optimal deployment survival probability is 1. We will use AL in order to solve the NP-complete problem "subset sum". Given a set $S = \{s_1, \cdots, s_n\}$ and a sum $S1$, we will build the following network.

Each member of the set $s \in S$ will be represented by an agent a_s, whose memory requirement is s, $mem(a_s) = s$. There are 2 nodes, n_1 and n_2, with available memory $mem(n_1) = S1$ and $mem(n_2) = \sum_{s \in S} s - S1$, respectively. The disconnect probability for each node is 0, i.e. $\forall_i dp(n_i) = 0$. It is easy to see:

1. The survivability of the optimal deployment is 1, if there exists a subset $S' \subseteq S$ such that $\sum_{s' \in S'} s' = S1$.
2. If there is no subset $S' \subseteq S$ which its sum is S_1, the optimal deployment's survival probability is 0.

Therefore we can define the following algorithm in order to solve subset sum:

- By given a set S, build the corresponding network N_s (as we described above).
- Run algorithm AL on N_s.
- If AL returns a deployment that its survival probability is larger than 0, return Yes (There exists a subset $S' \subseteq S$ which its sum is $S1$).
- Otherwise return No. ∎

5. Computing Survivability under the independence assumption

Unless $P = NP$, we know (as finding the survivability of a deployment is NP-hard) that any algorithm to compute survivability must be exponential. We provide two algorithms for this purpose (neither of which uses the expensive linear programming approach of [1]). The $SF1_n$ algorithm is a "naive" algorithm which is exponential in the number of nodes (and is therefore suitable for use when V is small) while the $SF1_a$ algorithm is exponential in the number of agents (and hence is suitable when M is small).

5.1. $SF1_n$: a naive algorithm

One way to find the survivability of μ under the assumption of independence is to use the definition directly. We find all subsets of $N_i \subseteq V$ w.r.t. which μ is a valid deployment. For each N_i, we compute the probability that N_i survives. The probability that μ survives (under independence) is the sum of the probabilities that the N_i's survive. This is illustrated below.

Example 2 *Suppose that a network and a deployment is given in Example 1. The possible future minimal networks are:* $N_1 = \{n_1\}, N_2 = \{n_2, n_3\}, N_3 = \{n_1, n_2, n_3\}, N_4 = \{n_1, n_2\}, N_5 = \{n_1, n_3\}$. *The survivability of each network is:* $surv(N_1) = (0.3)(0.6)(0.4) = 0.072$ *and* $surv(N_2) = (1 - 0.6)(1 - 0.4)(0.7) = 0.168$. *Similarly we have* $surv(N_3) = 0.072$, $surv(N_4) = 0.048$, $surv(N_5) = 0.108$. *The survivability of the deployment is given by:*
$surv(\mu) = 0.072 + 0.168 + 0.072 + 0.048 + 0.108 = 0.468$.

5.2. $SF1_a$: an agent-based algorithm

Given a deployment μ, let A_a be the event that all the nodes that agent a is deployed on are disconnected. Let A_d be the event that at least one of A_a occurs. The probability that event A_a does not occur is given by $surv(A_a) = \prod_{a \in \mu(n_k)}(1 - dp(n_k))$. In order for μ to survive, none of the events A_a should occur. Unfortunately, the A_as events are not mutually exclusive. Thus, in order to compute the survivability of μ using A_a we will need to apply the rule of the probability of the disjunction of not mutually exclusive events as presented below.

Proposition 5.1 *Suppose μ is a deployment w.r.t. an overlay network $Ne = \{V, E\}$ and suppose node failures are independent. Then*

$$SF1_a(\mu) = 1 - Pr(A_d) \text{ where} \quad (2)$$

$$\begin{aligned} Pr(A_d) &= Pr(A_{a_1} \vee A_{a_2} \vee \ldots \vee A_{a_{|M|}}) \\ &= 1 - \sum_{a \in M} Pr(A_a) + \sum_{a_i \neq a_j \wedge a_i, a_j \in M} Pr(A_{a_i} \wedge A_{a_j}) \\ &+ \ldots (-1)^{|M|+1} Pr(A_{a_1} \wedge \ldots \wedge A_{a_{|M|}}) \end{aligned} \quad (3)$$

$SF1_a$ finds all the A_{a_i}'s and then computes the above formula. The following example shows how it works.

Example 3 *Consider the network and deployment in Example 1. Agent a_1 is in nodes n_1 and n_2: the probability that both get disconnected is:* $Pr(A_{a_1}) = dp(n_1)dp(n_2) = 0.7 \times 0.6 = 0.42$. *Similarly, we have:* $Pr(A_{a_2}) = Pr(A_{a_3}) = dp(n_1)dp(n_3) = 0.28$; $Pr(A_{a_1} \wedge A_{a_2}) = Pr(A_{a_1} \wedge A_{a_3}) = dp(n_1)dp(n_2)dp(n_3) = 0.168$, $Pr(A_{a_2} \wedge A_{a_3}) = dp(n_1)dp(n_3) = 0.28$; $Pr(A_{a_1} \wedge A_{a_2} \wedge A_{a_3}) = dp(n_1)dp(n_2)dp(n_3) = 0.168$
Thus, the survivability of the deployment is given by:
$SF1_a(\mu) = 1 - (Pr(A_{a_1}) + Pr(A_{a_2}) + Pr(A_{a_3})) + (Pr(A_{a_1} \wedge A_{a_2}) + Pr(A_{a_1} \wedge A_{a_3}) + Pr(A_{a_2} \wedge A_{a_3})) - Pr(A_{a_1} \wedge A_{a_2} \wedge A_{a_3}) = 0.468$.

Note that $SF1_n$ is exponential *in the number of nodes* while $SF1_a$ is exponential *in the number of agents*. This is not a surprise as the problem of finding the survivability of an agent deployment is at least NP-hard even when the independence assumption is made.

We can improve efficiency if we use an idea in [1] that eliminates irrelevant agents—an agent a is irrelevant if there is an agent a' such that $\{n \mid a' \in \mu(n)\} \subseteq \{n' \mid a \in \mu(n')\}$. Throughout this paper, we will assume irrelevant agents are eliminated.

Example 4 *Consider the network Ne and a deployment μ given in Example 1. We denote the nodes of an agent a_i locates by $Loc(a_i)$. For each agent, we have $Loc(a_1) = \{n_1, n_2\}$, $Loc(a_2) = \{n_1, n_3\}$, and $Loc(a_3) = \{n_1, n_3\}$. As $Loc(a_3) \subseteq Loc(a_2)$, we remove a_3 from the deployment μ and update μ as:*
$\mu'(n_1) = \{a_1, a_2\}$, $\mu'(n_2) = \{a_1\}$, *and* $\mu'(n_3) = \{a_2\}$. *We compute the survivability of μ' by $SF2a$:*
$surv(\mu') = 1-(Pr(A_1)+Pr(A_2))+Pr(A_1 \wedge A2) = 1-(0.42+0.28)+0.168 = 0.468$. *Clearly, $surv(\mu') = surv(\mu)$.*

5.3. An upper bound on $SF1_a$

As $SF1_a, SF1_n$ both take exponential time, we now develop a fast algorithm to compute an upper bound on $SF1_a$. The upper bound we provide can be used to evaluate heuristics proposed later in the paper. Event A_d is the event that all nodes on which some agent is located get disconnected. We are therefore interested in the complement of event A_d. If we find a lower bound for $Prob(A_d)$ and subtract it from 1, we will get an upper bound on the survivability of μ. It is easy to see that $\sum_{a \in M} Pr(A_a) - \sum_{a_i \neq a_j \wedge a_i, a_j \in M} Pr(A_{a_i} \wedge A_{a_j})$ is a lower bound for $Prob(A_d)$. Similarly, any even number of terms in the expression of equation 3 provides a lower bound. This lower bound can be calculate incrementally until we run out of time or the difference between what we add to the expression (an odd term) and what we subtract from the expression (an even term) is very small. We can then can take the maximum among all the lower bounds that we computed. Subtracting this value from 1 will give us an upper bound on the survivability of μ.

We now propose several approximations to compute deployments. We are interested in finding lower bounds for $SF1_n$ and $SF1_a$ — this is because we want to be sure that when we say deployment μ has survival probability exceeding some threshold, this is in fact true.

5.4. SF2: an anytime algorithm

Using the same idea that we used for computing the upper bound, we can also compute a lower bound on the survivability of μ. Again, looking at the complementary event A_d, if we compute an upper bound of $Prob(A_d)$ and subtract it from 1, we get a lower bound on the values $SF1_a, SF1_n$ return. Any odd number of terms of equation 3 provides an upper bound. An anytime algorithm can iteratively add terms until we run out of time or the ratio between the maximum among the lower bounds and the min among the upper bounds is smaller than a specified ratio. Space limitations prevent us from presenting pseudo code for the algorithm.

5.5. $SF3$: tree-based approximation

In order to compute the survivability of μ using $SF1_n$ we sum up the probabilities of all possible future networks in which μ is valid. Instead of considering all possible future networks, $SF3$ only considers a polynomial number of future networks (using a search tree defined below) and returns the sum of their probabilities. The root of the tree is labeled by V. The probability of V is computed. For every $n \in V$ there is a vertex labeled $V - \{n\}$ in the second level of the tree. For each label of such a vertex the algorithm checks if μ is valid. For all such vertices, the probability of their labeled possible future networks is computed. Only the α vertices with the highest probability are further expanded in the same way. If a vertex labeled N_i is expanded, its children will be labeled by $N_i \setminus \{n\}$ for $n \in N_i$. Again only α vertices will be expanded and so on. We stop when there are no more nodes to expand. SF3 sums the probability of all the future networks in the search tree that μ is feasible with respect to them.

If α is polynomial in V, $SF3$ considers only a polynomial number of future networks. Therefore it may return very poor results if there is a large number of nodes. $SF3$ is bounded by the number of subsets considered multiplied by the largest subset probability. The largest subset probability is bounded by $\prod_{n \in V}(1 - dp(n))$. Assume the disconnect probability of nodes is distributed normally in $[0, 0.5]$. The survivability given by $SF3$ is usually no greater than $\alpha 0.9^{|V|}$, which is smaller than $10^{-\frac{|V|}{22}}$. Since the performance of $SF3$ could be very poor, we propose two heuristics to improve its value.

1. Prior to running $SF3$, we remove a set of agents M' whose locations (i.e. the nodes they are deployed on) are disjoint from the locations of any other agents. We can compute $surv(M')$ directly. We then apply $SF3$ on the remaining agents $M \setminus M'$. We return $surv(M') \cdot SF3(M \setminus M')$.
2. The second heuristic is based on the idea that if the number of nodes involved in $SF3$ is large (e.g. 20), we want to reduce the number of nodes by removing some nodes which contribute less to the survival of the μ. We sort the nodes in ascending order of $\sqrt[a']{1 - dp(n)}$, where a' denotes the number of agents on node n. The first K nodes can be deleted from the deployment. In this way, we discard nodes whose dp is very low or that have very few agents on it. [2]

2 It may be possible to get rid of more irrelevant agents by the idea in [1] in order to further simplify computation.

The algorithm below uses the 20-number as a bound.

Algorithm 1 $SF3(\alpha, \mu, Ne, M, dp)$

1. $disjoint_{surv} = rmvdisjoint(\mu, Ne, M, dp)$; *(* remove the agents with disjoint locations, return the survivability of the removed agent set *)*
2. **if** $|V| > 20$**, then** $rmvnodes(\mu, Ne, M, dp)$; **(* remove some nodes according to the criteria *)**
3. $best_{val} = calc\_surv(V)$; **(* compute the survivability of the future network V *)**
4. $temp = \{V\}$**,** $done = 0$**;**
5. **while (** $\neg done$ **), do**
   - **(a)** $flag = 0$**,** $X' = \emptyset$**,** $N_l = \emptyset$**; (*** N_l **is used for the feasible future networks of level** l ***)**
   - **(b) while (**$temp \neq \emptyset$ **)**
     - **i.** $X = headof(temp)$**,** $temp = temp \setminus \{X\}$**;**
     - **ii.** $X' = X' \cup \{X \setminus \{x_i\} \mid x_i \in X\}$**;**
   - **(c) while (**$X' \neq \emptyset$**), do (* remove invalid sets and repetitive sets in** X' ***)**
     - **i.** $X'_{sub} = headof(X')$**;**
     - **ii. if** $X'_{sub} \notin N_l$ **and** $\bigcup_{n \in X'_{sub}} \mu(n) = M$**, then (* checks whether** X'_{sub} **is feasible and wasn't considered earlier *)** $N_l = N_l \cup X'_{sub}$**,** $flag = 1$**;**
   - **(d) if (**$\neg flag$**), then** $done = 1$**;**
   - **(e) else, do**
     - **i. for** $(i = 0, i < |N_l|, i++)$ **(* add the survivability of feasible future networks of level** l ***)** $best_{val} = best_{val} + surv(N_i)$
     - **ii.** $N_l = sort(N_l, surv(N_l))$**; (* sort sets in** N **in descending order according to survivability *)**
     - **iii. for (**$j = 0, j < \alpha, j++$**) (* keep first** α **sets *)**
       - $temp = temp \cup headof(N_l)$**;**
       - $N_l = N_l \setminus headof(N_l)$**;**
6. **return** $(best_{val} \times disjoint_{surv})$**.**

The following proposition expresses that $SF3$ is a correct polynomial time approximation of $SF1_n$.

Proposition 5.2 *For any $\alpha > 0$, $SF3$ is less than or equal to the corresponding value returned by $SF1_n$.*

Proposition 5.3 *Suppose α is fixed. Then the time complexity to compute $SF3$ is $O(\alpha\ |V|^2\ log(\alpha\ |V|) + \alpha\ |V|^2\ M\ )$, i.e. the computation is polynomial if α polynomial in $|V|$.*

5.6. $SF4$: a disjoint based algorithm

For each agent $a_i \in M$, let $N^i = \{n^i_1, \cdots, n^i_k\}$ be the set of nodes where a_i is located. Let E^i_j be the event that the node n^i_j will survive, then the event that at least one copy of a_i will keep functioning is denoted by $E^i = E^i_1 \vee \ldots \vee E^i_k$. The probability of the event E^i can be computed by

$$P(E^i) = 1 - dp(n^i_1)dp(n^i_2)\ldots dp(n^i_k) \quad (4)$$

We can now define the event that a MAS deployment μ will survive by:

$$E(\mu) = (E^1_1 \vee \ldots \vee E^1_k) \wedge \ldots \wedge (E^{|M|}_1 \vee \ldots \vee E^{|M|}_l) \quad (5)$$

The probability of the event $E(\mu)$ represents the survivability of the deployment μ. Unfortunately, the E_is are not mutually exclusive. However, $SF4$ assumes that the events $E^1, E^2, \ldots, E^{|M|}$ are pairwise disjoint. We have

$$\begin{aligned} SF4(\mu) &= P(E^1)P(E^2)\ldots P(E^{|M|}) \\ &= (1 - dp(n^1_1)\cdots dp(n^1_k)) \times \ldots \\ &\times\ (1 - dp(n^{|M|}_1)\cdots dp(n^{|M|}_l)) \end{aligned}$$

Fortunately $SF4(\mu)$ gives an underestimate of the survivability of μ.

Proposition 5.4 *$SF4$ underestimates the actual survivability of μ.*

5.7. $SF4_g$: a group algorithm

$SF4$ computes each agent's survival probability and then returns the product of these survival probabilities. If no node contains more than one agent, then $SF4$ returns the exact answer. However, in general, when the number of agents is large and there is a large number of nodes in which many agents are located, $SF4$ can return a very low probability. To improve this, if there are agents in a deployment that coexist in various nodes, we would consider these agents as a group and compute the group's survivability. We divide all agents into several such groups, and then take the product of the survival probabilities of all groups as the survivability of the deployment. An intuitive way to group agents is to consider the agent a who has the lowest survivability. We group a with other agents who have the most common nodes with it. When we compute the survivability of each agent group, we use the algorithm $SF1_a$. As $SF1_a$ takes exponential time in the number of agents, we limit the size of each group.

5.8. $SF5$: a split algorithm

Given a specific node, $n \in V$, we can consider two possible disjoint events. In the first event, E_1, the node will stay connected (and E_1's probability is $1 - dp(n)$). Alternatively, in event E_2, the node will be disconnected (with probability $dp(n)$). If n stays connected, all the agents that are deployed on it will survive. Thus the survivability of the network, in this case, will depend on the survivability of the

rest of the agents that are located on $V \setminus \{n\}$. If n is disconnected, the survivability of the network depends on the rest of the nodes, i.e., $V \setminus \{n\}$. The survivability of the original network is thus $(1 - dp(n))Prob(E_1) + dp(n)Prob(E_2)$. In both events the problem of computing the probability is smaller than the original problem and could be solved recursively. The sub problems usually become even smaller when getting rid of irrelevant agents by the idea in [1]. There are several stopping rules that are specified in the three first lines of the pseudo code. The first two rules refers to situations in which it is possible to compute the exact survival probability of the future network. The third one has to do with future networks that has very small probability (computing through the recursion using p; $p = 1$ in the first call to $SF5$). For these networks of very low probability, $SF4$ is applied to underestimate the survivability.

Algorithm 2 $SF5(\mu, Ne, M, dp, p, \epsilon)$

1. *If $M = \emptyset$ return 1;*
2. *Else, if the agents of M are located on disjoint sets of nodes then return($\prod_{a \in M}(1 - \prod_{a \in \mu(n)} dp(n))$);*
3. *Else, if $p < \epsilon$ then return $SF4(\mu, Ne, M, dp)$.*
4. *Else, choose a node $n \in V$.*
   (a) *$V' = V \setminus \{n\}$; $Ne' = Ne'' = (V', V' \times V')$; $\mu' = \mu$; $M' = M \setminus \{a | a \in \mu(n)\}$; Get rid of irrelevant agents in M and M';*
   (b) *Adjust μ and Ne' w.r.t M and μ' and Ne'' w.r.t M';*
   (c) *return $dp(n) \times SF5(\mu, Ne', M, dp, dp(n)p, \epsilon) + (1 - dp(n)) \times SF5(\mu', Ne'', M', dp, (1 - dp(n))p, \epsilon)$.*

Several heuristics that depends on the number of agents deployed on each node and their dps can be applied to choose the node in line 4 above. We state in the following proposition that $SF5$ gives an underestimation of the survivability of μ.

Proposition 5.5 *$SF5$ underestimates the actual survivability of μ.*

6. Experiments

All algorithms in this paper were implemented on a Linux PC. We assessed the quality of a solution as follows:

- if the number of nodes (or agents) is small enough (say, less than 16), we compare the values returned by different approximations with the exact exponential algorithm $SF1_n$(in the case where there are more agents than nodes) or $SF1_a$ (when there are more nodes than agents); or
- if it is not feasible to compute either $SF1_n$ or $SF1_a$, we use the upper bound algorithm for comparison.

We considered various experimental settings. In this paper, we consider instances taken from a (fictitous) company that is using local servers, personal computers, and some web servers to locate and run multiagent applications. As we know, web servers and personal computers have high probabilities of going down, while local servers usually have low disconnect probabilities. In the next section, we describe the variations of the settings we used in our experiments. We use the term *space ratio* to refer to the ratio of the total amount of space available on nodes to the total space requirements of agents.

6.1. Environments

Suppose a MAS application M includes a large number of agents but a relatively small number of servers (or nodes) is available. We set the ratio of agents and nodes be $5/3$. We consider the following two environments for such problem size setting:

s1: The network consists of a small number of web servers N_w (30%) and many local servers N_l (70%). Their disconnect probabilities are either very high, i.e. $dp(N_w) \geq 0.9$, or very low, i.e. $dp(N_l) \leq 0.1$. The space ratio of nodes and agents is between 2 and 3.

s3: The network includes local servers only. Suppose that some of these servers are new, while the others are old, and therefore disconnect probability of the servers is distributed normally between 0 and 0.4. The space ratio of nodes and agents is around 4.

Consider another multiagent application M' which consists of a small number of agents. The company intends to deploy M' on many personal computers and local servers because the available resources on each server or PC are limited. We assume that the ratio of nodes and agents is $5/3$. The following environment is specified:

s2: Personal computers (30%) are employed, whose disconnect probabilities are over 0.9; they also use several local servers which have low dp's (less than 0.1). The space ratio of nodes and agents is around 2–3.

s4: Only local servers of different ages are used to host M'. The disconnect probability of the servers is distributed normally between 0 and 0.4. The space ratio of nodes and agents is around 4.

A sample of 31 existing IMPACT agents deployed in various applications agents was used to determine a distribution of agent sizes (in the range of 0 to 250 KB). We use the environments $s1 - s4$ described above to test the survivability algorithms.

6.2. Agent deployment

The method used to generate deployments is important as different survivability algorithms may work well with different types of deployments. In this paper, we generate deployments by the heuristics proposed in [1], namely node-based heuristics and agent-based heuristics. In addition, we use a random-based method to represent other pos-

sible deployment. The deployment heuristics work as follows.

Node-based: This is based on the knapsack problem. We first sort nodes in ascending order according to their disconnect probabilities. We then place agents on the sorted nodes starting from the node with the lowest dp. We put as many agents as possible on this node, then go to nodes with the second lowest dp and so on.

Agent-based: This is based on the idea that we should first deal with agents with high resource requirements. We sort agents in ascending order according to resource requirements, deploy the first agent, then choose the agent with the second highest resource requirement, as so on until there is no more space left for placing agents. When we deploy an agent, we always choose the node whose dp is the lowest among those capable of storing this agent.

Random-based: We first randomly choose a node, and then randomly select and place agents on it subject to space constraints. We make sure the deployment uses up all available resources on nodes.

6.3. Experimental results

In the results below, α in the tree based algorithm ($SF3$) is set as the number of nodes in the deployment. The threshold of approximation ratio in the anytime algorithm ($SF2$) is defined as 0.9 and the time limit on the main part of the algorithm was set to 5 seconds. The time units are of microseconds. Every recorded observation was averaged over 50 runs. We compare approximations in terms of the computation time and the approximation ratio (i.e. $\frac{SFs}{\text{actual value}}$) on various deployments with different environment settings $s1 - s4$. We varied the problem size, i.e. sum of the number of agents and nodes, in all experiments. We present the results as follows.

Experiment 1. In Experiment 1, we ran and compared five approximations in setting $s1$ where the space ratio of nodes to agents is between 2 and 3 and the dps are distributed either in 0–0.1 or in 0.9–1. There are two kinds of deployments: one generated by agent-based heuristic and the other by node-based heuristic. Table 1 illustrates the results of approximation ratio by different algorithms. The problem size varied from 48, 64 to 80. In the table, $n18, a30$ refers to a MAS of 30 agents deployed over 18 nodes. $SF3$ and split algorithms return very good approximations. In particular, $SF3$ always gives higher accuracy than $SF4$, anytime and group algorithm. The split algorithm has the best approximation, though the difference between it and $SF3$ is very small. Figure 1 shows the computation time (in logarithm scale) taken by different approximations with varying problem size on agent-based deployments. It is obvious that $SF3$ needs much less computation time than split algorithm and anytime algorithm (in the order of 10^3, 10^4, $10^5 - 10^7$ respectively), while $SF4$ is the fastest algorithm among all. It only takes several microseconds to compute the survivability of a given deployment. The time needed by group algorithm is close to that of $SF3$.

Overall, in experimental setting $s1$, if the space ratio of nodes to agents is below 3, $SF3$ outperforms other algorithms both w.r.t. approximation ratio and computation time into account.

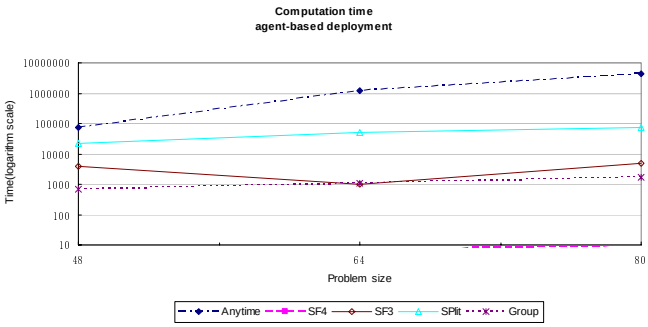

Figure 1. Experiment 1: computation time using different algorithms with setting S1

Experiment 2. Table 2 and Figure 1 report the results of various approximations in experimental setting $s2$. In terms of accuracy, the split algorithm is still the best, followed by $SF3$, which outperforms the group algorithm both on the node-based deployments with problem size 48, 64, 80, and on the agent-based deployments with size 80. In addition, the performance of $SF4$ and group algorithm is better than that in Experiment 1. This is not surprising as in $s2$, there are more agents who have disjoint locations with others compared to those deployments in setting $s1$. Thus $SF4$ and group algorithm should work better. The computation time taken by different approximations is similar to that shown in Figure 1. We did not include it in the paper due to space constraints. The results imply that $SF3$ is best among all approximations as far as both time and accuracy concerned.

Experiment 3. Figure 2 and 3 illustrate the approximation ratios and the computation times for different algorithms in setting $s3$. In both figures, the x-axis represents the problem size, varying from 32 to 98 in step of 16. The figures show computation times on various deployments (i.e, node-based, agent-based, and random-based) respectively. We did not include the results of $SF3$ since its approximation ratio is much lower (below 0.8) than others in this setting. Figure 2 shows the advantage of the split algorithm, which gives the best approximation ratio no matter what kind of deployments it works on. The figure illustrates that all algorithms

Problem size	deployment	Anytime algorithm	SF3	SF4	Split algorithm	Group algorithm
$n18, a30$	node-based	0.951475	0.998919	0.879377	0.999515	0.948644
	agent-based	0.930565	0.972725	0.879446	0.973295	0.947005
$n24, a40$	node-based	0.85197	0.964037	0.836856	0.965813	0.923682
	agent-based	0.861461	0.881713	0.796948	0.889531	0.859259
$n30, a50$	node-based	0.899324	0.937145	0.775934	0.939865	0.893412
	agent-based	0.811792	0.851059	0.731787	0.852492	0.827303

Table 1. Experiment 1: Approximation ratio of different algorithms with setting S1

Problem size	deployment	Anytime algorithm	SF3	SF4	Split algorithm	Group algorithm
$n30, a18$	node-based	0.928353	0.990334	0.974706	0.99961	0.989189
	agent-based	0.939275	0.996562	0.973966	0.999569	0.998401
$n40, a24$	node-based	0.880772	0.980832	0.952544	0.98856	0.979202
	agent-based	0.907723	0.968986	0.941974	0.991491	0.988068
$n50, a30$	node-based	0.923707	0.981228	0.958711	0.973521	0.96628
	agent-based	0.910653	0.972855	0.900914	0.976648	0.967943

Table 2. Experiment 2: Approximation ratio of different algorithms with setting S2

size	deployment	Anytime	SF4	Split	Group
n20,a12	node-based	0.99988	0.999971	0.999996	0.999996
	agent-based	0.999845	0.99981	0.99996	0.99994
	random	0.991724	0.999837	0.999998	0.999999
n30,a18	node-based	0.999922	0.999965	0.999983	0.999988
	agent-based	0.99979	0.999535	0.999852	0.999823
	random	0.991622	0.999918	0.999985	0.999983
n40,a24	node-based	0.999723	0.999969	0.999985	0.999983
	agent-based	0.999588	0.999269	0.999729	0.999681
	random	0.991255	0.999639	0.999895	0.999889
n50,a30	node-based	0.999661	0.999968	0.99989	0.999895
	agent-based	0.999398	0.999136	0.999649	0.999638
	random	0.990589	0.999655	0.999775	0.999787
n60,a36	node-based	0.999681	0.999923	0.999957	0.999971
	agent-based	0.999217	0.998105	0.999344	0.999097
	random	0.976279	0.998948	0.999663	0.999713

Table 3. Experiment 4: Approximation ratio of different algorithms with setting s4

have high approximate ratio (over 0.96) on node-based deployments. As for agent-based deployments, all algorithms return over 0.90 approximate ratio with problem size no larger than 80. Only the ratio of $SF4$ drops to 0.86 when the problem size goes up to 96. All algorithms but anytime algorithm get above 95% accuracy on random-based deployments. The approximate performance of all algorithms decrease as the problem size increases. The anytime algorithm converges much faster on node-based deployments than agent-based deployments with setting $s3$—the computation time taken on the latter is 100 times or so when the problem size is over 48 according to Figure 3. The time taken by split algorithm is about 10 times that taken by the group algorithm. $SF4$ is the fast algorithm among all.

Experiment 4. Experiment 4 was carried out with environment setting $s4$, where the ratio of nodes to agents is $5 : 3$, and dp's are distributed normally in $[0, 0.4]$. Table 3 and Figure 4 show the effect of problem size on the approximation ratio and computation time by various approximations. We did not include the results of $SF3$ because of its low approximation ratio (below 0.8). As far as accuracy concerned, the performance of all approximations is very good—the accuracy is always over 0.97. In particular, $SF4$, split and group algorithms could always achieve over 0.998 approximation ratio in Experiment 4 according to Table 3. Since there are more nodes than agents, the size of each node decreases compared with that in Experiment 3, thus most of agents are disjoint with others w.r.t their locations, which makes algorithms, especially $SF4$ and group algorithm, perform well. Unlike that shown in Experiment 3, anytime algorithm in Experiment 4 converged pretty fast with no matter what kind of deployments. The computation time taken by $SF4$, split and group algorithm is very close to those in Experiment 3.

Experiment 5. In Experiment 5, we repeated experiment 1 with setting $s1$ but increased the space ratio of nodes to agents from 2–3 to 3–4. Table 4 illustrates that $SF3$ returns lower survivabilities than split and group algorithms, moreover, its accuracy is even lower than $SF4$ when the number of nodes and agents is 30 and 50. In addition, Figure 5 shows that $SF3$ is the most time-consuming algorithm with the problem size 80 where $SF3$ needs 100 times of computation time than split and group algorithms. The experiment results imply that with such setting used in Experiment 5, $SF3$ is no longer the preferred approximations especially with the large problem size. Instead, split algorithm demonstrated its capability of fast and accurate computation.

Due to space constraints, we are unable to report full details of all our experiments. A brief summary is given below:

- As shown in Experiment 1 and 2, $SF3$ is preferable to

Problem size	deployment	Anytime algorithm	SF3	SF4	Split algorithm	Group algorithm
$n18, a30$	node-based	0.981061	0.98647	0.96855	0.998956	0.98707
	agent-based	0.973769	0.986316	0.964855	0.999059	0.988083
$n24, a40$	node-based	0.979052	0.991792	0.978524	0.99866	0.994861
	agent-based	0.980291	0.992817	0.976784	0.998752	0.99174
$n30, a50$	node-based	0.98326	0.975502	0.982723	0.998131	0.994358
	agent-based	0.981894	0.975746	0.978989	0.997898	0.994035

Table 4. Experiment 5: Approximation ratio of different algorithms in setting S1 but with larger space ratio (3-4)

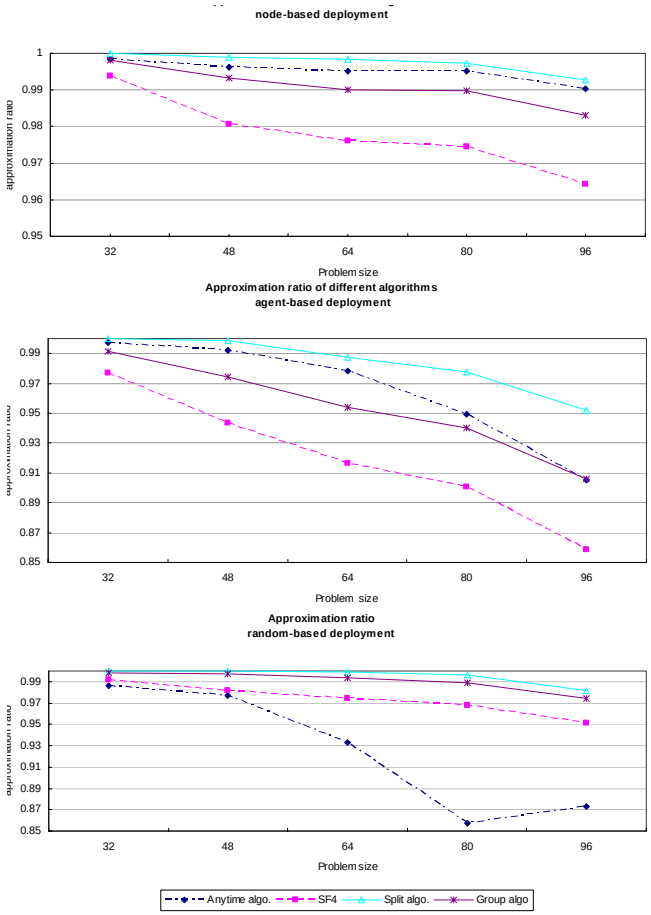

Figure 2. Experiment 3: approximation ratio of different algorithms with setting S3

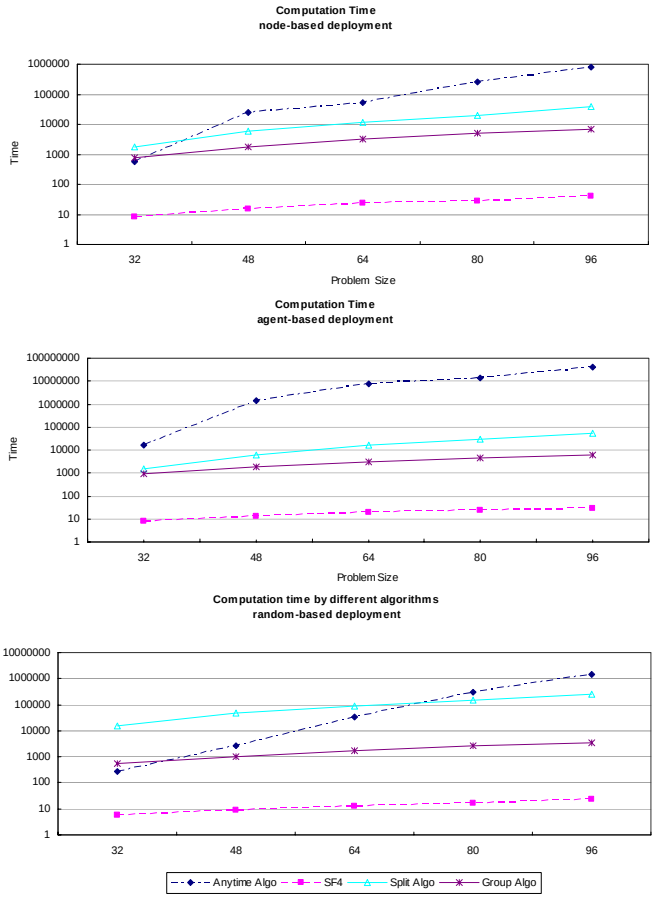

Figure 3. Experiment 3: computation time of different algorithms with setting S3

other approximations as far as time and accuracy are concerned in settings $s1$ and $s2$, where (1) the disconnect probabilities of the nodes are distributed either in $[0, 0.1]$, or in $[0.9, 1]$; and (2) the space ratio of nodes to agents is less than 3. However, Experiment 5 shows that when the space ratio is increased to 4, $SF3$ no longer give the best performance. The split algorithm is the best instead.

- The Split algorithm outperforms others in setting $s3$, where (1) the disconnect probabilities of the nodes follow normal distribution in $[0, 0.4]$; and (2) the ratio of nodes to agents is $3 : 5$.
- in setting $s4$, where the dp's are normally distributed in $[0, 0.4]$ and the ratio of nodes to agents is $5 : 3$, although every approximation does well in terms of accuracy, $SF4$ is preferable to others when computation time is taken into account.
- All experiment results demonstrate that $SF4$ has an unbeatable fast computation time.

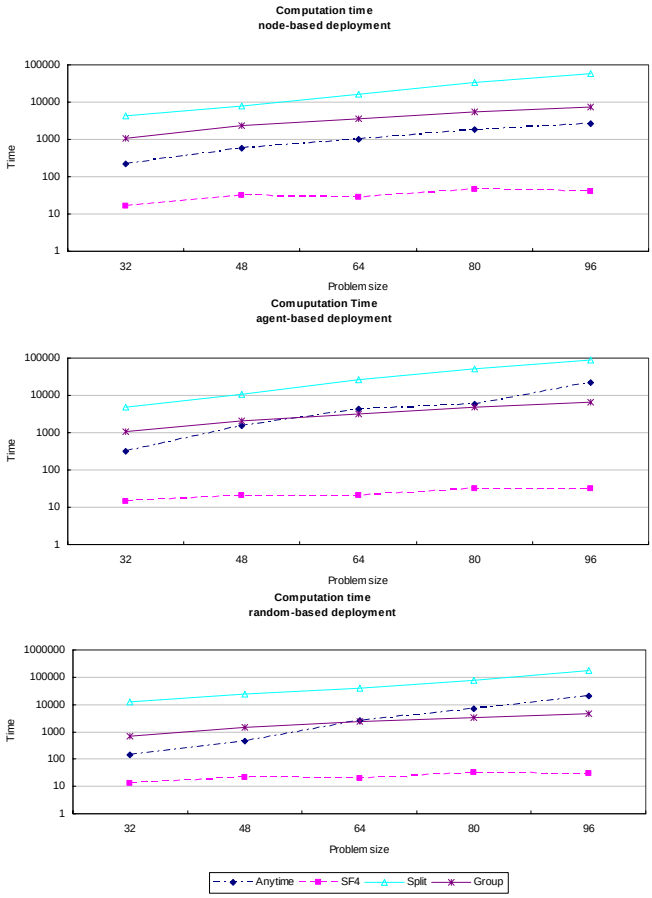

Figure 4. Experiment 4: computation time of different algorithms with setting S4

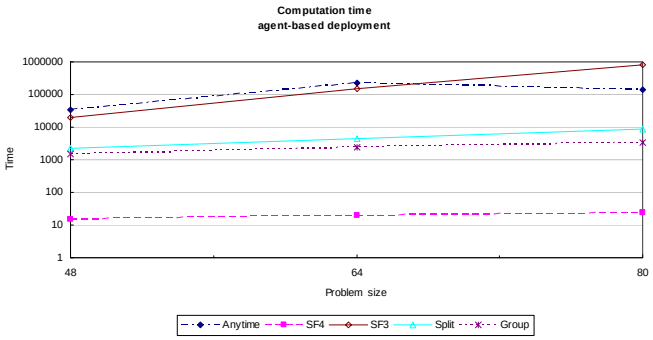

Figure 5. Experiment 5: computation time using different algorithms in setting S1 but with larger space ratio (3-4)

7. Conclusion

In this paper, we extended the work presented in [1]. We studied what happens when we assume that node failures in the network are independent. We provided complexity results on computing optimal deployments under the independence assumption and showed that all polynomial approximations are bound to be terrible in some cases. We proposed exponential exact algorithms to compute the survivability of a deployment and also provided fast heuristics. These algorithms to compute survivability of a deployment may be used in conjunction with algorithms in [1] to find optimal deployments. We conducted a set of experiments to assess the survivability of deployments found by these algorithms and also the computation time involved. Our results show that different algorithms work well in different environments.

References

[1] S. Kraus, V. S. Subrahmanian, and N. C. Tacs. Probabilistically survivable MASs. In *Proc. IJCAI-03*, pp 789–795, 2003.

[2] P. M. G. Apers. Data allocation in distributed database systems. *ACM Trans. Database Syst.*, 13(3):263–304, 1988.

[3] K. S. Decker, K. Sycara, and M. Williamson. Cloning in intelligent, adaptive information agents. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems: Methodologies and Applications*, pp 63–75. Springer-Verlag, 1997.

[4] X. Du and F. J. Maryanski. Data allocation in a dynamically reconfigurable environment. In *Proceedings of the Fourth International Conference on Data Engineering*, pp 74–81. IEEE Computer Society, 1988.

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

[6] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *Proc. AAMAS-02*, pp 737–744, 2002.

[7] J. A. Giampapa, O. H. Juarez-Espinosa, and K. P. Sycara. Configuration management for multi-agent systems. In *Proc. of AGENTS-01*, pp 230–231, 2001.

[8] Y. Huang and O. Wolfson. A competitive dynamic data replication algorithm. In *ICDE*, pp 310–317, 1993.

[9] G. F. Johnson and A. K. Singh. Stable and fault-tolerant object allocation. In *Symposium on Principles of Distributed Computing*, pp 259–268, 2000.

[10] S. Kumar, P. Cohen, and H. Levesque. The adaptive agent architecture: achieving fault-tolerance using persistent broker teams. In *Proc. of ICMAS*, pp 159–166, 2000.

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

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

[13] S. Mishra and Y. Huang. Fault tolerance in agent-based computing systems. In *Proc. of the 13th ISCA*, 2000.

[14] 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.