Move-efficient fault-tolerant simulation of message-passing algorithms by mobile agents

Tsuyoshi Gotoh[*] Fukuhito Ooshita[†] Hirotsugu Kakugawa[*] Toshimitsu Masuzawa[*]

Abstract

We propose a fault-tolerant algorithm to simulate message-passing algorithms in mobile agent systems. We consider a mobile agent system with k agents where f of them may crash for a given f $(\leq k-1)$. The algorithm simulates a message-passing algorithm, say Z, with $O((m+M)f)$ total agent moves where m is the number of links in the network and M is the total number of messages created in the simulated execution of Z. The previous algorithm [5] can tolerate $k-1$ agent crashes but requires $O((m+nM)k)$ total agent moves. Therefore, our algorithm improves the total number of agent moves for $f = k-1$ and requires a smaller number of total moves if f is smaller.

1 Introduction

A *distributed system* is composed of many computers (nodes) that can communicate with each other. Recently distributed systems have become larger, which makes it difficult to design them. As a paradigm to circumvent the difficulty, *mobile agents (agents)* have attracted a lot of attention [3]. An agent is a software program which can move autonomously in a distributed system, collect information at visited nodes, exchange the information with other agents and execute actions at visited nodes using the information. An agent can be considered as encapsulation of data and actions, and the number of agents in a network restricts concurrency of actions executed in the network. This makes algorithm design easier in mobile agent systems than in message-passing systems. So far many agent-based algorithms have been proposed for several tasks, such as leader election, naming, locating agents, rendezvous, stabilization, termination detection, exploring and topology recognition [3]. From view of security, algorithms for intruder capture [1, 2] and network decontamination [10, 12] have been proposed.

While most works stated above assume agents and nodes work correctly, recent large-scale distributed systems can no longer make such an assumption. For this reason, some researches consider faulty nodes where their states are disrupted [6, 7] or visiting agents are destroyed [8, 11]. In addition, we should consider the scenario such that agents themselves become faulty. For example, if the system spreads to all over the world, agents may move a long distance by passing through lots of physical links. During the movement, agents may crash (or disappear) when one of the links suffers from an error. Hence algorithms tolerant to faults of agents are required for many tasks.

As an approach to realize agent-based algorithms for many tasks, we focus on simulation of *message-passing algorithms* in mobile agent systems [4, 5, 13]. If agents can simulate message-passing algorithms efficiently, they can execute many tasks efficiently because efficient message-passing algorithms have been proposed for many tasks in literature [9]. The only existing work to simulate message-passing algorithms in a fault-tolerant manner is the one by Das et al. [5]. In this work, the authors propose two algorithms to simulate message-passing algorithms by asynchronous agents when at most $k-1$ agents crash, where k is the number of agents. One algorithm simulates a message-passing algorithm with $O((m+nM)k)$ total agent moves by agents with distinct IDs, where m is the number of links, n is the number of nodes and M is the number of messages created in the simulated execution of the message-passing algorithm. Another algorithm simulates a message-passing algorithm with $O((m+nk)M)$ total moves by anonymous agents. Note that, in the algorithm for agents with distinct IDs, the number of moves per message is (or the multiplication factor of M) $O(nk)$.

In this paper, we propose a new *fault-tolerant algorithm* to simulate message-passing algorithms by asynchronous agents with distinct IDs. Our algorithm assumes at most f agents crash for a given f, and simulates a message-passing algorithm with $O((m+M)f)$ total agent moves. That is, the number of moves per message is $O(f)$ when $M = \Omega(m)$ holds. Note that, because f agents can become faulty and agents move asynchronously (i.e., the time required to move along a link is unbounded and unpredictable), every message should be delivered

[*]Graduate School of Information Science and Technology, Osaka University
[†]Nara Institute of Science and Technology

by $f+1$ agents in the worst case. This means our algorithm is asymptotically optimal concerning of the number of agent moves per message.

2 Preliminaries

2.1 Network

A network is modeled by a connected undirected graph $G=(V,E)$, where V is a set of nodes and E is a set of communication links. Each link $e \in E$ connects distinct nodes in V. A link that connects nodes u and v is denoted by e_{uv} or e_{vu}. In this paper, we denote $n=|V|$ and $m=|E|$. The degree of u is defined as the number of incident links of u, and is denoted by deg_u. The maximum degree $\max\{deg_u | u \in V\}$ of the network is denoted by Δ. The neighbors of u are nodes directly connected to u, and the set of them is denoted by N_u. Each link incident to node u is labeled locally at u by bijection $\lambda_u : \{(u,v) : v \in N_u\} \to \{1,2,\ldots,deg_u\}$ and u distinguishes its neighbors by the labels. Note that, $\lambda_u(e_{uv}) \neq \lambda_u(e_{uw})$ holds for distinct neighbors v and w of u. The labeling is independent from those of other nodes; for an edge e_{uv}, $\lambda_u(e_{uv}) \neq \lambda_v(e_{uv})$ may hold. We say $\lambda_u(e_{uv})$ is a port number (or port) of e_{uv} on u.

We consider two different computation models of a network, a message-passing model and a mobile agent model, which are defined in the following subsections.

2.2 Message-passing model

The definition of the message-passing model in this paper follows [5].

In the message-passing model, each node u is modeled as a state machine (S_u, δ_u), where S_u is a set of (possibly infinite) node states and δ_u is a state transition function. The state machine may be dependent on its node ID if exists: nodes with different IDs may be modeled as different state machines. Two states in S_u are designated as initial states: one is for a (spontaneous) initiator and the other for a non-initiator. The transition function δ_u is defined as $\delta_u : S_u \times M \times P \to S_u \times 2^{M \times P}$, where M is a set of all possible messages (including a special message null) and P is a set of port numbers. The function δ_u determines, from a current state and a received message with its incoming port, a subsequent state of the node and a set of messages to be sent with their outgoing ports. The initial state for the initiator and the special message null are used only for the first action of the initiator, which is independent of the incoming port of the message null. The state machine can depend on the degree and the ID (if exists) of the node.

Each node executes the following operations atomically in each step: 1) it receives a message or initiates an algorithm spontaneously, 2) executes local computation and updates its own state, and 3) if necessary, sends messages to its neighboring nodes by using the primitive $SEND(c, \lambda_u(e_{uv}))$ repeatedly (node u can send a message c to node v by using the primitive $SEND(c, \lambda_u(e_{uv}))$). There exists at least one spontaneous initiator, which is designated as the special initial state and initiates an algorithm spontaneously (by receiving the null message). Except for the initial steps of initiators, every process takes a step only when it receives a message. Note that, since the set of initiators is unknown in advance, algorithms should work correctly for any set of initiators.

Communication in the message-passing model are reliable, that is, it satisfies the following:

[A1] Every message sent by node u to its neighbor v is eventually received by v exactly once.

[A2] A message is received by node u only when it was previously sent by a neighbor v.

Each link in the network is FIFO, that is, when u sends messages c_1 and c_2 to v in this order, v receives c_1 before c_2. The system is asynchronous, that is, the time required to transfer a message between neighbors is finite but unbounded.

2.3 Mobile agent model

In the mobile agent model, all the actions (i.e., computation and communication) on a network are carried out by mobile agents. Let A be a nonempty set of mobile agents existing on the network and $k=|A|$. Each agent has its own memory, called a *notebook*. In this model, a node works only as a repository and a memory on a node is called a *whiteboard*.

Each agent a has a unique ID $a.id$ and we assume each ID is represented in $O(\log k)$ bits. Every agent knows neither k nor n. Each agent a is initially allocated to some node v called a *homebase* of a. We assume $k \leq n$ and homebases of agents are mutually distinct.

Each agent a is modeled as a state machine (S_a, δ_a), where S_a is a set of agent states and δ_a is a state transition function. A state in S_a is designated as an initial state. The transition function δ_a is defined as

$\delta_a : S_a \times W \times (P \cup \{0\}) \to S_a \times W \times (P \cup \{0\})$, where W is the set of all possible whiteboard states and P is the set of port numbers. The inputs of the transition function δ_a are a current state of an agent, a state of the whiteboard on its current node, and a port number (or 0 explained in the below) through which the agent arrived, and the outputs are a subsequent state of the agent, a new state of the whiteboard, and a port number (or 0 explained in the below) through which the agent leaves. Port number 0 in the inputs implies the agent initiates the algorithm at its current node or the agent stays at the current node from its previous action, while $p > 0$ implies the agent arrives at the current node from port p. Port number 0 in the output implies the agent stays at the current node, and $p > 0$ implies the agent leaves the current node through port p. The state machine can depend on the agent ID. The state transition can depend on the degree and the ID (if exists) of the node which the agent is staying at or visiting. This is implemented by storing the node degree and ID in the whiteboard.

Each agent executes the following operations atomically in each step: 1) It arrives at a node or initiates an algorithm at its homebase, 2) executes local computation to update its own state (including its notebook contents) and the whiteboard contents of its current node, and 3) moves to a neighbor of its current node or stays at its current node.

This paper considers simulation of the message-passing model on the mobile agent model. We assume that the target model (or the message-passing model) is reliable but the host model (or the mobile agent model) is prone to faults. An agent may crash (or disappear) when it moves through a link, but it never crashes when it is on a node. We assume at most $f \leq k - 1$ agents crash and every agent knows the upper bound f. After an agent leaves a node, it arrives at the next node eventually unless it crashes during the movement. Once an agent crashes, it disappears from the network forever. We say an agent is faulty (resp., non-faulty) if it crashes (resp., never crashes) during the execution. Note that agents cannot recognize faulty agents as long as they work correctly. Each link in the network is FIFO, that is, when agents a_1 and a_2 move from node u to node v in this order, a_1 arrives at v before a_2 unless a_1 crashes during the movement. The system is asynchronous, that is, the time required for an agent to move from a node to its neighbor is finite but unbounded.

3 Agent-based simulation of message-passing algorithms

In this section, we present a simulation algorithm, correctness proof and evaluation of move complexity. We denote Z as the simulated message-passing algorithm for hereafter. A message-passing algorithm that eventually terminates uses a finite number of messages and is a target algorithm of the simulation algorithm in Section 3.1. Thus, most of algorithms designed so far can be the target of the simulation.

3.1 The execution of a algorithm

In this subsection, we propose an agent-based simulation algorithm of a message-passing algorithm with a finite number of messages (i.e., an eventually terminating algorithm). Our algorithm consists of two parts, 1) searching initiators (search part) and 2) simulating an execution of nodes and delivering messages (delivery part).

First, we present the search part, 1) searching initiators. Each agent starts to search initiators from its homebase by the depth-first search. When it finds an initiator, it starts the delivery part, 2) simulating an execution of nodes and delivering messages. After completing the delivery part, it resumes the search part to find another initiator. The agent records its searching path of the search part by writing the incoming port in the whiteboard of each visited node so that it can backtrack.

In the search part, an agent backtracks to the previous node when at least one of following conditions is satisfied.

1. There is no unsearched port at the current node.
2. A cycle is detected in its searching path of the search part.
3. The agent detects that other $f + 1$ agents which execute the search part have already visited the current node.

Conditions 1 and 2 come from the depth-first search. Condition 3 is introduced to save the total number of agent moves. Our algorithm can tolerate agent crashes by using multiple agents to transfer a message, however it is enough that each message is transfered by $f + 1$ agents since at most f agents crash (there is at least one non-faulty agent in $f + 1$ agents). Thus, an agent backtracks when it detects that other $f + 1$ agents which execute the search part. The agent terminates its execution when it completes the search part and returns to its homebase.

Next, we present the delivery part, 2) simulating an execution of nodes and delivering messages.

An agent starts the simulation when it finds an initiator during the depth-first search of the first part. Note that, by Condition 3 of the first part, at most $f + 1$ agents visit each initiator and start simulation.

The agent delivers messages successively in the depth-first fashion, that is, if there exists a message to transfer to another node in the node that the agent visits to deliver a message, it takes a message from the node and delivers it. It records its delivering path in the same way as the search part so that it can backtrack.

Since each message is transfered by at most $f+1$ agents for fault-tolerance, it may be delivered multiple times. However it is processed only once, that is, an agent simulates the action of a node on receipt of a message only when it has not been simulated.

When an agent takes a message from the node, it stores its ID to *send-member* of the message in the whiteboard of the node to indicate that the agent transfered the message. The message is deleted from the node when one of its *send-member* agents returns and, at this time, *send-member* of the current node is reset to empty.

In the delivery part, an agent backtracks to the previous node when at least one of following conditions is satisfied.

1. There is no message to transfer at the current node.
2. A cycle is detected in its delivering path of the delivery part.
3. The current node is locked using the port other than the one the agent arrives through. We describe the locking mechanism later.
4. When the agent backtracks to the current node, the node is locked but the agent is not a *lock-member* agent of the node.

Condition 1 realizes message deliveries in the depth-first fashion. Condition 2 is introduced to prevent the delivering path from growing so long, which saves the whiteboard space of nodes. Conditions 3 and 4 are introduced to guarantee that all the messages are delivered since using only Conditions 1 and 2 makes some messages remain undelivered.

Consider the case of Fig. 1. First, an agent a arrives at t and delivers messages from t to u and agent b follows a and arrives at u. Then, a backtracks to t earlier than b and messages at t and v are deleted. Agent b is still in transit in link e_{uv}. Second, an agent c arrives at y from x and delivers messages from y to v through z and w. Then, c crashes after generating two messages at v, one is to y and the other is to u in this order. After that, agent b arrives at v from u and delivers messages from v to v through y, z and w. Then, b detects a cycle at v, backtracks to w, and deletes the message from w to v. Then, while b backtracks from w to z, b crashes. Here, node v has a message to transfer to u but it is possible that no agent arrives at v after the situation since there is no message toward v. Thus, in this case, the message from v to u may be left undelivered.

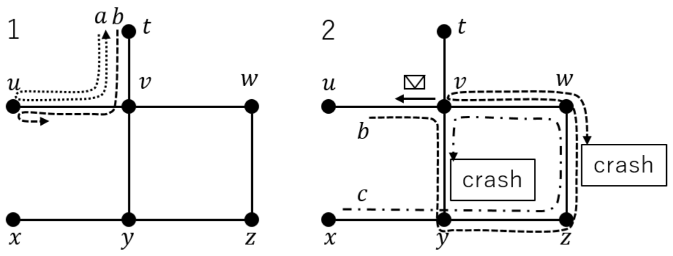

Figure 1: An example where conditions 1 and 2 allow a message to remain undelivered.

A possible way to avoid such undelivered messages is not to introduce Condition 2. In this case, an agent continues to deliver messages as long as the current nodes have messages to transfer. But this allows the delivering path to become so long when a long message chain exists. It requires large whiteboard spaces since the delivering path is recorded in the whiteboards of nodes. Thus, we insist on Condition 2 to save the whiteboard space. So we introduce the locking of nodes as another way to guarantee deliveries of all messages.

A reason why the above case happens is that agents which have distinct delivering paths deliver the same message. So we design the locking so that it prevents distinct delivering paths from merging and branching.

An agent locks the current node by writing, to the whiteboard, the port through which it arrives when the current node is unlocked. An agent that arrives at the locked node can deliver a message from the node only when it arrives through the port that is used for the locking. Otherwise, it has to backtrack to the previous node in the delivering path. Note that, since all the delivering paths of the delivery part of agents start from an initiator, by repeating above, agents which deliver the same message must have the same delivering path.

An agent records its ID as a *lock-member* in the whiteboard of a locked node when it locks the node or arrives through the port that is used for the locking. When a *lock-member* agent backtracks from the locked node, it unlocks the node and resets the *lock-member* of the node to empty.

Condition 4 makes an agent backtrack to the previous node in the delivering path when it backtracks to a node but is not a *lock-member* of the node. This implies that the node was already unlocked for the locking such that the agent was a *lock-member*, that is, agents which delivered the message from the node may have distinct delivering path. This makes the agent keep backtracking along the delivering path until it reaches a node where it is a *lock-member* or it started the simulation of nodes and delivering messages.

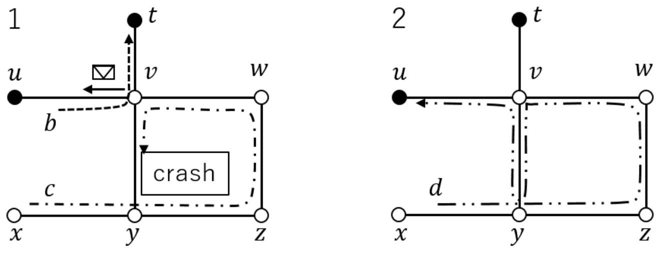

Figure 2: An example where conditions 3 and 4 are applied

For clarification, consider the case of Fig. 2. In Fig. 2, white nodes are locked. Like Fig. 1, agent c delivers messages in order of y, z, w and v and crashes while b is still transit in link e_{uv}. In this case, since v is locked, b backtracks to t through v. Thus, d which arrives at x delivers messages in order of y, z and w. Then it can arrive at v from w through the port used for locking v, so it continues to deliver messages stored at v.

An agent resumes the message delivery when it finds its ID in *lock-member*. It terminates the delivery part and resumes the search part (i.e., searching an initiator) when it reaches the starting node but is not a *lock-member*.

A Following theorem holds.

Theorem 1. *The proposed algorithm simulates Z correctly with $O((m+M)f)$ total agent moves when at most f agents are faulty.*

4 Conclusion

In this paper, we proposed a new algorithm to simulate a message-passing algorithm in the mobile agent model. It requires $O((m+M)f)$ agent moves to tolerate at most $f \leq k-1$ agents crash where m is the number of links in the network and M is the number of messages in the simulated execution of the message-passing algorithm. The previous algorithm requires $O((m+nM)k)$ agent moves when at most $k-1$ agents crash. Thus our algorithm improves the previous algorithm in the number of agent moves.

Our algorithm simulates different executions of the message-passing algorithm. The actual number of agent moves depend on the number and the creation pattern of messages in the simulated execution. Thus, it is interesting as a future work to investigate the actual number of agent moves for concrete examples of message-passing algorithms.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP26280022, JP26330084, JP16K00018, JP17K19977.

References

[1] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209. ACM, 2002.

[2] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. In *International Colloquium on Structural Information and Communication Complexity*, pages 70–84. Springer, 2006.

[3] J Cao and S. K Das. *Mobile Agents in Networking and Distributed Computing*. John Wiley & Sons, 2012.

[4] Jérémie Chalopin, Emmanuel Godard, Yves Métivier, and Rodrigue Ossamy. Mobile agent algorithms versus message passing algorithms. In *International Conference On Principles Of Distributed Systems*, pages 187–201. Springer, 2006.

[5] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In *International Colloquium on Structural Information and Communication Complexity*, pages 289–303. Springer, 2007.

[6] Shantanu Das, Matúš Mihalák, Rastislav Šrámek, Elias Vicari, and Peter Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *International Conference On Principles Of Distributed Systems*, pages 463–480. Springer, 2008.

[7] Yoann Dieudonné and Andrzej Pelc. Deterministic network exploration by a single agent with byzantine tokens. *Information Processing Letters*, 112(12):467–470, 2012.

[8] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19(1):1–18, 2006.

[9] K Erciyes. *Distributed Graph Algorithms for Computer Networks*. Springer Science & Business Media, 2013.

[10] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, 18(03):547–563, 2007.

[11] Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science*, 384(2-3):201–221, 2007.

[12] Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Network decontamination in presence of local immunity. *International Journal of Foundations of Computer Science*, 18(03):457–474, 2007.

[13] Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Move-optimal gossiping among mobile agents. *Theoretical Computer Science*, 393(1-3):90–101, 2008.