

Specification of Fault-Tolerant Mobile Agent Execution and its Building Block

Sanaz Rahimi, Shahram Rahimi, Bidyut Gupta
Department of Computer Science
Southern Illinois University
Carbondale, Illinois 62901
[srahimi, rahimi, bidyut]@cs.siu.edu

Abstract

Mobile agent technology is one of the fastest growing areas of research for application development on the Internet today. The potential of mobile agents to perform many tasks, from the menial and routine, to the more complex ones has been widely recognized and thus spurred. However, the lack of a fault tolerant infrastructure and methodologies that address fault tolerant execution of mobile agents highlights a major draw back of this technology. In this paper, we explore the conditions of fault tolerant mobile agent execution and investigate a possible building block for a fault-tolerant execution platform.

1 Introduction

A mobile agent is an executing program that can migrate during execution from machine to machine in a heterogeneous network. In other words, the agent can suspend its execution, migrate to another machine, and then resume execution on the new machine from the point at which it left off. On each machine, the agent interacts with stationary agents and other resources to accomplish its task [1].

Failures in such an environment may lead to a partial or complete loss of the agent. For instance, a failure of the device on which the agent is currently executing causes all information about the agent not held in stable storage to be lost. In the worst case, the information about the entire agent is discarded. The agent owner has the problem of uncertainty, meaning that he cannot determine whether the agent is lost or whether its execution has only been delayed due to slow processors or communication links. This uncertainty may lead to the following situations: 1) The agent owner believes that the agent has been lost, when in fact it has not been. Launching another agent may cause multiple executions of the agent code on some devices. 2) The agent owner waits for the agent to finish its execution, but the agent has failed. Clearly, this is a blocking situation [2]. Fault-tolerant mobile agent execution removes this uncertainty and ensures that the agent eventually reaches its destination or at least notifies the agent owner of a potential problem.

This paper identifies the requirements for fault tolerant mobile agents: *nonblocking* and *exactly-once*. The nonblocking property ensures that the failure of an infrastructure component (e.g., a machine, place, agent, or communication link) does not prevent progress in the agent execution [3]. A blocking execution is undesirable because it can lead the agent owner to potentially wait a long time for the return of the agent. Replication of the agent enables prevention of blocking without requiring a reliable failure detection mechanism. In other words, the failure detection mechanism is allowed to erroneously suspect failures. Adding redundancy at the stage execution masks system failures and enables the agent to continue execution despite failures, as shown in Figure 1.

Replication may result in multiple executions of the agent; which is undesirable for exactly-once property; which requires that the code of an agent be executed exactly once. This is particularly important for operations with side effects, for example, an agent withdrawing money from an account. Moreover, a basic building block is defined which is fundamental to enforce the exactly-once property. Basically, the stage executions as local transactions allows us, by controlling the commit/abort decision, to enforce the exactly-once property [4].

In this paper, the places where the first and last stages of an agent execute (i.e., p_0 and p_n) are called the agent *source* and *destination*, respectively. The sequence of places between the agent source and destination (i.e., p_0, p_1, \dots, p_n) is called the itinerary of a mobile agent. Logically, a mobile agent executes in a sequence of stage actions. Each stage action sa_i consists of potentially multiple operations op_0, op_1, \dots . Agent a_i ($0 \leq i \leq n$) at the corresponding stage S_i represents the agent a that has executed the stage actions on places P_j ($j < i$) and is about to execute on place p_i . The execution of a_i on place p_i results in a new internal state of the agent as well as potentially a new state of the place (if the operations of an agent have side effects) [2].

In the rest of the article, first we specify fault-tolerant mobile agent execution in terms of two properties: nonblocking and exactly-once. Then, the basic building blocks to ensure these properties are identified. Finally, a summary section concludes the paper.

2 The Specification

The specification of the desired fault-tolerant mobile agent execution is specified in terms of two properties: nonblocking and exactly-once execution.

2.1. Infrastructure Failures and the Blocking Problem

While a mobile agent is executing on a place p_i , an infrastructure failure of p_i might interrupt the execution of a_i and prevent any progress of the mobile agent execution. During the time p_i is down, the execution of a_i and consequently the entire mobile agent execution cannot proceed. We say that the execution of a_i is *blocked*. Provided the availability of suitable recovery mechanisms, the execution of a_i on p_i proceeds when p_i recovers from the failure. Generally, a mobile agent execution is called *blocking* if a single failure renders progress in the mobile agent execution impossible until the failed component (e.g., machine, place, agent, or communication link) recovers. In contrast, a *nonblocking* mobile agent execution can continue the execution despite a single failure.

Generally, blocking mobile agent executions are undesired. In particular, if the failed component does not recover, then the agent is lost and never returns to the agent owner. Moreover, long downtimes of components lead to very long response times and may be unacceptable for the agent owner. Hence, mobile agent executions are preferably nonblocking.

2.2. Agent Replication and the Exactly-Once Execution Problem

2.2.1. Replication to Prevent Blocking

Blocking can only be overcome by introducing redundancy. More specifically, if a place fails, the agent is executed on another place. However, redundancy of execution may result in multiple executions of (parts of) the mobile agent. While this is not a problem for idempotent operations (e.g., operations without side effects), it should not occur for nonidempotent operations. Take, for instance, an agent that retrieves money from the agent owner's bank account. This is clearly a nonidempotent operation and multiple executions of this operation have the undesired effect of multiple money retrievals. Therefore, nonidempotent stage actions must be executed exactly-once [4]. On the other hand, operations such as reading an account balance allow multiple executions. Clearly, blocking in a mobile agent execution consisting only of idempotent operations is easily prevented by sending multiple agents. The redundancy introduced by replication masks failures and ensures progress of the mobile agent execution. Figure 1 illustrates the replication approach.

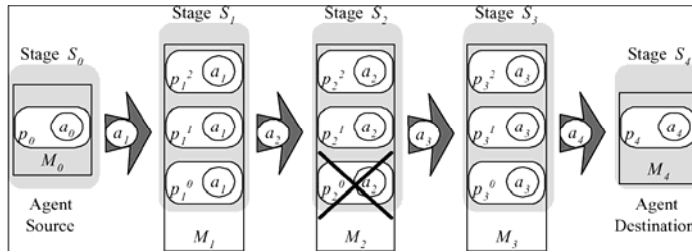


Fig. 1. Agent execution with redundant places, where a place fails; the redundant places mask the place.

At stage S_i , a set of places $M_i = \{p_i^0, p_i^1, p_i^2, \dots\}$ executes the agent a_i . Even if place p_i^0 fails the agent a_i is not lost, as the other places in M_i have also received a_i and can start executing it.

2.2.2. Replication and the Exactly-Once Problem.

Replication allows executions to be nonblocking. However, it may also lead to multiple agent executions. Assume, for instance, that p_i^0 fails (see Figure 2). Place p_i^1 starts executing a_i , which results in agent a_{i+1} and M_{i+1} . In the meantime, p_i^0 recovers and continues the execution of a_i . Clearly, this requires that the agent's state and code have been checkpointed to stable storage upon arrival of the agent on p_i^0 .

If p_i^0 and p_i^1 commit the agent's stage action, the agent is executed multiple times and results in duplicate agents a_{i+1} and a_{i+1} . Although blocking of the agent execution because of a failure to p_i is prevented, the mechanism to prevent

blocking results in multiple agent executions. Consequently, the problem of multiple agent executions and blocking are related problems in the sense that preventing blocking may lead to multiple agent executions.

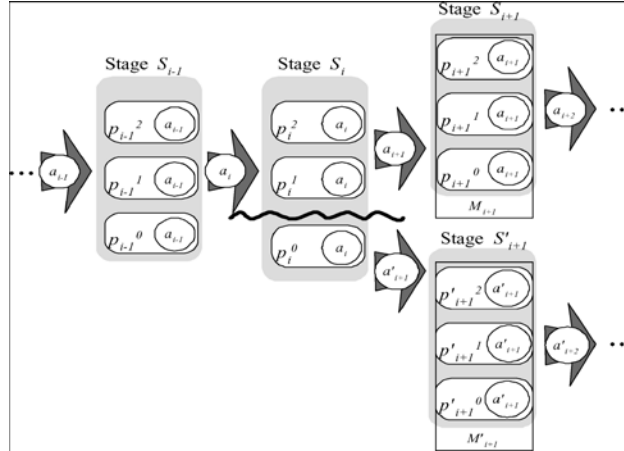


Fig. 2. Replication potentially leads to a violation of the exactly-once property

In summary, we require that a fault tolerant mobile agent execution satisfy the following:

- 1) *Nonblocking.*: failures must not prevent the termination of the agent execution.
- 2) *Exactly-once.*: The mobile agent's stage actions are executed exactly once.

3. Basic Building Block: Local Transaction

In Section 2.1 we have specified the fault tolerant mobile agent execution in terms of the non-blocking and exactly-once properties. In this section, we define a basic building block that is fundamental to enforce the exactly-once property and thus implicitly also the non-blocking property: the local transaction.

3.1. Local Transaction

The stage action sa_i of mobile agent a_i encompasses a set of operations op_0, op_1, \dots , that act on the local services (see Figure 3). Locally, on the place p_i , the agent executes the set of operations, thereby transforming a consistent state of the agent and the place into another consistent state (consistency). The effects of executing sa_i have to be durable, that is, reflected by the place (new state of the place) as well as by the agent a_{i+1} , and not to be lost anymore (durability).

Moreover, we require that sa_i execute entirely or not at all (atomicity). Only when sa_i has completed its execution should the results, including the modifications to the place, generated messages, or spawned child agents, be visible to other agents (isolation). These four properties correspond to the specification of an ACID (transaction). Hence, sa_i has to run transactionally. This is ensured using a *local transaction* to execute sa_i [3].

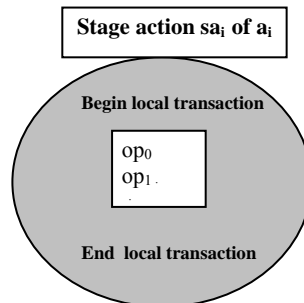


Fig. 3. Stage action of agent a_i runs as a local transaction.

The concept of a local transaction is an important building block for fault-tolerant mobile agent execution. The local transaction consisting of operations op_0, op_1, \dots terminates either by a commit or an abort decision. If the decision is to commit, the effects of executing op_0, op_1, \dots become durable, otherwise, all the modifications are undone. We classify

the approaches for fault-tolerant mobile agent execution according to when and by whom this commit/abort decision is taken.

3.2. Enforcing Exactly-Once Execution Property

Executing the stage action sa_i transactionally is the basic mechanism allowing us to enforce the exactly-once property for mobile agent executions. Actually, whereas the entire mobile agent is executed exactly-once, the local operations op_0, op_1, \dots of a stage action are executed *at-most-once*.

3.2.1. At-Most-Once Ensured on Place p_i

Failures during the execution of an agent's stage action sa_i potentially leave the execution in an inconsistent state. More specifically, some of the operations op_0, op_1, \dots that correspond to the stage action may have been executed, whereas others have not. The agent a_i as well as the place (or rather, its services) are thus in an incorrect, transitory state. Executing sa_i transactionally prevents such inconsistent states, as either all the operations op_0, op_1, \dots are executed or none at all.

3.2.2. Exactly-Once for the Entire Mobile Agent Execution.

In Section 2.2, we have shown how replication can prevent blocking. Replication may lead to multiple executions of a stage action sa_i on different places p_i^j and p_i^k . To prevent a violation to the exactly-once execution property, only one of the executions must be committed, whereas the other(s) have to be aborted. This is why stage actions are executed exactly-once. In Figure 2, for instance, the execution of a_i on place p_i^o has to be aborted, whereas the execution of a_i on p_i^j is committed.

Running the stage executions as local transactions allows us, by controlling the commit/abort decision, to enforce the exactly-once property. Note that terminating local transactions at stage S_i (i.e., issuing either abort or commit) requires that the place running the local transaction eventually recover from a failure and that potential link failures (i.e., network partitions) be resolved. However, such a link failure or place failure should not prevent the continuation of the agent execution, that is, they should not lead to blocking.

4 Summary

In this paper we identified the two requirements for fault tolerant mobile agents: 1) *nonblocking* and 2) *exactly-once*. Clearly, the nonblocking property ensures that the failure of an infrastructure component does not prevent progress in the agent execution. A blocking execution is undesirable because it can lead the agent owner to potentially wait a long time for the return of the agent. Replication allows us to address the issues of fault tolerance and blocking, but may result in multiple executions of the agent which is undesirable. The exactly-once property requires that the code of an agent be executed exactly once.

Also we have identified the local transaction as the basic building block for fault-tolerant mobile agent execution (i.e., for addressing infrastructure failures). The stage actions of the mobile agent are executed as local transactions, this allows us, by controlling the commit/abort decision, to enforce the exactly-once property. Basically, to prevent a violation to the exactly-once execution property; only one of the executions must be committed, whereas the other(s) have to be aborted. Even though mobile agent applications are still not very wide-spread; with more research into fault-tolerant, the development of mobile agent-based applications can be furthered.

References

- [1] Alpern, B. and Schneider, F. 1985. Defining liveness. Information Processing Letters 21. PLEISCH, S. AND SCHIPER, A. 2002. Non-blocking transactional mobile agent execution. In Proc. of 22nd IEEE Int. Conference on Distributed Computing Systems (ICDCS'02) (Vienna, Austria), 443–444.
- [2] Minsky, Y., Van-Rennesse, R., Schneider, F., and Stoller, S. 1996. Cryptographic support for fault-tolerant distributed computing. In Proc. of 7th European Workshop on ACM SIGOPS (Connemara, Ireland). ACM, New York, 109–114.
- [3] Rothmel, K. and Strasser, M. 1998. A faulttolerant protocol for providing the exactly-once property of mobile agents. In Proc. of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98) (West Lafayette, Indi.). 100–108.
- [4] Schneider, F. 1997. Towards fault-tolerant and secure agency. In Proc. of the 11th Int. Workshop on Distributed Algorithms, Saarbuckten, Germany. Invited paper.