# FANTOMAS
# Fault Tolerance for Mobile Agents in Clusters

Holger Pals, Stefan Petri, and Claus Grewe

Medical University of Lübeck
Institute of Computer Engineering
Ratzeburger Allee 160, D-23538 Lübeck
{pals, petri, grewe}@iti.mu-luebeck.de

**Abstract.** To achieve an efficient utilization of cluster systems, a proper programming and operating environment is required. In this context, mobile agents are of growing interest as base for distributed and parallel applications. As mobile and autonomous software units, mobile agents can execute tasks given to the system and allocate independently all the needed resources. However, with growing cluster sizes, the probability of a failure of one or more system components and therewith the loss of mobile agents rises. While fault tolerance issues for applications based on "traditional" processes have been extensively studied, current agent environments provide only insufficient, if at all, extensions for a capable reaction on such kinds of failures.
We examine fault tolerance with regard to properties and requirements of mobile agents, and find that independent checkpointing with receiver based message logging is appropriate in this context. We derive the FANTOMAS (**Fa**ult-Toler**a**n**t** appr**o**ach for **M**obile **A**gen**t**s) design which offers a user transparent fault tolerance that can be activated on request, according to the needs of the task. A theoretical analysis examines the advantages and drawbacks of FANTOMAS.

## 1 Introduction and Motivation

Clusters of standard components, PCs or workstations, can (almost) reach the performance of "real" parallel computers, but at much better price-performance ratio. They have already entered the Top 500 list of supercomputers.

The agent concept, originally developed in the domain of artificial intelligence, attracts growing interest in many application areas. According to a simple definition [20], *mobile agents* are active and autonomous software entities, which are able to migrate through the network to get from one node to the other. During migration, a mobile agent keeps its program code, data and possibly execution state. Despite their autonomy, mobile agents are no loners but they try potentially to reach common goals by interacting. Furthermore, mobile agents are proactive, reactive and cognitive [26]. So, mobile agents can take subtasks of the parallel application and travel independently within the cluster to allocate all the resources that are needed to perform their tasks. Furthermore, they can respond to dynamically changing load situations in the cluster by traveling from highly loaded nodes to lightly loaded ones. Therefore, not the user but the mobile agents are responsible for managing the cluster utilization. Also, migration has been shown to be more light weighted than with "traditional" programs [13, 5].

All these capabilities predestine mobile agents for many applications in distributed environments, e. g. data collecting, searching and filtering, network management, entertainment, e-commerce or parallel processing [2, 24]. One example application we are working on is mapping of data flow graphs for mobile robot control onto agents [8]. In contrast to alternative implementations, where in general only data can be transported to processing units, mobile agents bring operators to the sources of data. This can improve the efficiency and simultaneously reduce the network traffic in a drastic manner [4].

Cluster, and therefore parallel applications running on them, are very susceptible for failures of components of the cluster. However, programmers and users of distributed applications are interested in their algorithms and solutions. They expect fault tolerance as a service from the underlying run time system. These considerations show the necessity for a design, which enables user-transparent fault tolerance in agent environments. Current agent systems, and also the underlying operating systems, provide this feature only insufficiently, if at all.

In this paper we introduce an approach for such a design. It can be applied to different agent systems, if they fulfill certain requirements as discussed below. The approach (FANTOMAS, Fault-Tolerant approach for mobile agents) offers a user-transparent fault tolerance which can be selected by the user for every single application given to the environment. Thereby, the user can decide for every application wether it has to be treated fault-tolerant or not.

In the next sections, we briefly introduce agent systems, and give overview of related work. Section 3 examines fault tolerance with regard to properties and requirements of agent environments.

## 2 Related Work: Fault Tolerance for Mobile Agents

Current operating systems do not have support for mobile agents. Thence, special Agent systems are needed to provide suitable infrastructure as middleware beween mobile agents and operating system. The particular functions agent systems have to provide can be found in [9]. [17] presents an overview of existing agent systems.

In the area of mobile agents, only few work can be found relating to fault tolerance. Most of them refer to special agent systems or cover only some special aspects relating to mobile agents, e. g. the communication subsystem. Nevertheless, most people working with mobile agents consider fault tolerance to be an important issue [5, 25].

Johansen et al. [7] detect and recover from faulty migrations inside the TACOMA [6] agent system. When an agent migrates, a *rear-guard agent* is created that stays on the origin node. It monitors the migrated agent on the destination node. This very simple concept does not tolerate network partitioning.

In the scope of the TACOMA project, Minsky et al. [10] propose an approach based on an itinerary concept, i. e. a plan, which nodes the mobile agent has to visit and what it has to do there. They assume that the itinerary is known at start time and the order of visited nodes is fixed. Fault tolerance is obtained by performing every itinerary step on multiple nodes concurrently and sending the results (resp. the Mobile Agent) to all nodes of the next step. The majority of the received inputs from the last step

becomes the input of the new task for this step and so on. Thus, a certain number of faults can be tolerated in each step. Disadvantages of this fault tolerance concept are the very inflexible description of the itinerary and the simple model of Mobile Agents. For example, communication between different mobile agents is not included in this concept, so it is not suited for distributed or parallel applications.

Strasser and Rothermel present a more flexible itinerary approach for fault tolerance [22] within the Mole system [1]. Independent items in this enhanced itinerary can be reordered. Each itinerary stage comprises the action that has be to done on one node. When the mobile agent enters a new stage by migrating to the next node of the itinerary, called a *worker* node, it is also replicated onto a number of additional nodes, called *observers*. If the *worker* becomes unavailable (due to a node or network fault), the *observer* with the highest priority is selected as the new worker by a special *selection protocol*. A *voting protocol* ensures the abortion of wrong multiple *workers* in case of a network fault. The *voting protocol* is integrated into a 2-phase commit protocol (2PC) that encloses every stage execution. These protocols cause a significant communication overhead. Just as in the work of Minsky et al., nothing is said about the interaction between different mobile agents that are executed concurrently.

Vogler et al. [23] introduce a concept for reliable migrations of mobile agents based on distributed transactions. The migration protocol is derived from known transaction protocols like the already mentioned 2PC. Besides the migrations, no other fault tolerance aspects of mobile agents are treated.

Murphy and Picco [12] also treat only a special aspect of agent systems, the problem of messages that never reach their destination because of the high agent mobility, not because of failures in the agent system or the network.

These problems can arise in almost every message delivery scheme. Their approach is based on the Chandy-Lamport algorithm for consistent global snapshots [3]. This algorithm is used to force all messages to be delivered at least once. The network graph can grow dynamically, but nodes that fail at runtime cannot be removed. Due to the underlying snapshot algorithm, messages can be initiated at any time, but delivery at each node is restricted to one message per source.

Concordia [11], a commercial agent system from the Mitsubishi Electric Information Technology Center America (ITA), advertises fault-tolerant execution of mobile agents, but no further information about the tolerated failures and their treatment is available.

## 3 Concepts for a Fault Tolerance Approach for Mobile Agents

As shown in the previous sections, despite the generally agreed-upon necessity, existing agent systems contain only limited provisions for fault tolerance, if at all. Especially treatment of node or agent failures with support for communicating agents is covered only insufficiently. However, such support is essential for distributed and/or parallel applications. This section discusses possibilities and approaches to augment an agent system to achieve fault tolerance, with focus on these fault and application types.

First, the goals are described, then the fault model for an agent system is explained, and the faults examined with respect to their occurrence probability and treatment in

existing systems. From this, a set of faults is determined, for which further treatment is still needed. After that, an overview over fault tolerance approaches in known environments is given, and examined, if and how they are suited for mobile agents. From these investigations, the FANTOMAS concept is developed.

### 3.1 Goals and Requirements

The goals for a fault tolerance approach for mobile agents result from the the properties of agents and agent systems, and from the intended application area, parallel computation.

- **Support for communicating agents:** this is essential for parallel distributed applications, but not covered by some of the related work.
- **Autonomy:** If the actions to achieve fault tolerance was dictated by a supervisor instance, the autonomy were limited. All decisions that are made by an agent according to its normal autonomous behavior would need to be coordinated with such a supervisor. Besides fault tolerance, also other aspects, e. g. load management, would be involved, because otherwise they could contradict the decisions of the fault tolerance supervisor.
- **Fault tolerance as optional feature:** Not every application in an agent environment is that important that it requires fault-tolerant execution. The user, the agent environment, or the application itself, should be able to decide individually, if and when fault tolerance is to be activated. Also, the activation of fault tolerance for one application should have no influence on other already running applications, so that the concurrent execution of fault-tolerant and non-fault-tolerant applications is possible.
- **User transparency:** Programmers and users of applications are usually interested in coding their algorithms and getting their problems solved. They do not want to care for fault tolerance in the applications. Thus, to enable fault-tolerant execution, it should not be necessary to change the application code.
- **Efficiency:** One major goal of using mobile agents for execution of parallel applications in cluster systems is efficient resource usage. Thus, fault tolerance measures should have a very low overhead during the fault free operation. This is also one more reason to require fault tolerance as optional feature. Also, of course, recovery after a fault should be fast.
- **No modifications to hardware, operating system, or run time environment:** The necessity for such modifications would limit the usability of fault tolerance measures to dedicated system environments, and portability would be severed.
- **Portability and reusability:** Changes to an existing agent environment should be an augmentation, not a redesign or reimplementation. Already existing function units should remain available at least for the non-fault-tolerant execution of applications. Reusing existing function units also guarantees that existing applications stay usable.

From these goals, we can already derive some requirements for the selection of an agent system, which can be used as base for a realization of the FANTOMAS concept:

- **Modular, exchangeable architecture:** With a monolithic agent kernel, it would not be possible to adapt specific functionalities to the requirements of fault tolerance measures. To enable activation of fault tolerance properties during run time, the complete agent would have to be replaced with one that already carries those functionalities with it, all the time. This would increase memory and computing time demands even in the case of non-fault-tolerant execution. To avoid this disadvantage, a modular composition of mobile agents is required, in which the functionalities are contained in specific modules which can be exchanged at run time.
- **Separation between application and agent kernel:** The partial tasks of a parallel application, which are submitted to the agent environment, should be integrated seamlessly into the modular composition of the environment. The separation between application and agent kernel, on one hand, facilitates user transparency, and, on the other hand, makes it possible that one agent can execute different applications modules in its life cycle. Nevertheless, the application must have the possibility to influence the agents behavior.
- **Function modules working in parallel:** The required modularity and separation between application and the rest of the agent suggests that the functional units should be contained in parallel working modules. This enables independent and concurrent execution of different strategies, e. g. concerning migration or resource usage. This increases the agent's degrees of reactivity and proactivity.
- **Adaptivity:** It must be possible to influence the behavior of a mobile agent during run time. Without this possibility, adding fault tolerance (e. g. on demand from the user) would not be possible.
- **Automatic detection of dependencies:** Some of the functional units in a mobile agent expect certain functionality and services from other units, which are mutually usable. For example, the fault tolerance unit could initiate a migration by using a service of the migration unit. As explained above, each agent should contain only those modules, that are actually needed. However, when a module is exchanged, it might happen that the new module expects more services than are already present in the agent. To maintain transparency, such dependencies must be detected and satisfied automatically.

### 3.2 Fault Model

Several types of faults can occur in agent environments. Here, we first describe a general fault model, and focus on those types, which are for one important in agent environments due to high occurrence probability, and for one have been addressed in related work only insufficiently.

- **Node failures:** The complete failure of a compute node implies the failure of all agent places and agents located on it. Node failures can be temporary or permanent.
- **Failures of components of the agent system:** Failures of agent places, or components of agent places become faulty, e. g. faulty communication units or incomplete agent directory. These faults can result in agent failures, or in reduced or wrong functionality of agents.

- **Failures of mobile agents:** Mobile agents can become faulty due to faulty computation, or other faults (e. g. node or network failures).
- **Network failures:** Failures of the entire communication network or of single links can lead to isolation of single nodes, or to network partitions.
- **Falsification or loss of messages:** These are usually caused by failures in the network or in the communication units of the agent systems, or the underlying operating systems. Also, faulty transmission of agents during migration belongs to this type.

Especially in the intended scenario of parallel applications, node failures and their consequences are important. Such consequences are loss of agents, and loss of node-specific resources. In general, each agent has to fulfill a specific task to contribute to the parallel application, and thus, agent failures must be treated. In contrast, in applications where a large number of agents are sent out to search and process information in a network, the loss of one or several mobile agents might be acceptable.

In the following, we assume that only one agent place is located on each network node, which is typical for parallel execution environments. Generalization to multiple agent places per node is easily derived. Also, currently only crash faults are regarded. The reason is the observation that other node faults are significantly less frequent, but require much higher effort to detect.

Failures of the entire network occur relatively rarely, and standard network protocols provide tolerance against loss or falsification of messages. Thus, those fault types are not regarded in the following. Treatment of network partitioning will be addressed in future work.

An 1-fault assumption is used, i. e., during the recovery after a node failure, no second node failure does occur. After recovery, further node failures can be tolerated, until only one node remains intact. Tolerating more than one simultaneous node failure would require using additional resources, which conflicts with the efficiency requirements. Also, the intended short recovery times yield a very low probability for the occurrence of further failures in that phase.

It is further assumed that failed nodes stay unavailable, since repair and re-use is not generally possible. Many agent systems or environments do not support dynamic reconfiguration during run time.

Also, intermittent faults are not regarded. They are difficult to diagnose, and occur with only very low probability.

Failures of single agents, independent from node failures, are regarded and should be tolerated. Here, more specific, the failures of agents which are currently being executed on a node are regarded. In contrast, the problem of agent failures during migration was already investigated in other work [23] and is not regarded here.

Further, correct communication is assumed. Messages are not falsified or intercepted during their transmission. These faults are already investigated in the literature, e. g. [12].

### 3.3 Discussion of Fault Tolerance Methods

Fault tolerance methods for distributed applications have been widely studied in the context of “traditional” processes. As shown above, agent environments and mobile

agents have some special properties and requirements, which must be regarded, but also can be exploited, when fault tolerance is introduced.

Recovery can be divided into *backward recovery* and *forward recovery*. Forward recovery avoids the rollback and re-execution phase and thus can provide fast recovery with (almost) seamless computation progress. However, for our primarily intended application area, this is not so important. To avoid the overhead that is incurred by forward recovery during fault-free execution, we choose backward recovery.

The state of an agent-based distributed application consists of the states of the agents, and of the communication links between them. When saving the state of the communication links, the *Domino Effect* [18] has to avoided. Because of the high dynamicity of agent systems, *independent checkpointing* techniques are beneficial against *coordinated checkpointing* algorithms.

To avoid the Domino Effect with these techniques, the exchanged messages must be logged, so that they can be regenerated during the re-execution phase. Though receiver based logging has more time overhead during fault-free operation, its advantage against sender based logging is faster recovery (see also e. g. [19]). Also, important for agents, recovery and pruning of the message log can be done autonomously, without interaction with other user agents.

### 3.4 The FANTOMAS Concept

From these considerations, we choose independent checkpointing with receiver based logging as base for our fault tolerance approach for mobile agents. Adhering to the agent paradigm, and exploiting the already available facilities of the mobile agent resp. the agent environment, an agent is used as the stable storage for the checkpointed state and the message log. For each mobile agent (called *user agent* in the following), for that fault tolerance is enabled, a *logger agent* is created. A user agent and its logger agent form an agent pair[1] (figure 1). The logger agent does not participate actively in the application's computation, and thus needs only a small fraction of the available CPU capacity. It follows the user agent at a certain, non-zero, distance on its migration path through the system. They must never reside on the same node, so that not a single fault destroys both of them. User and logger agent monitor each other, and if a fault is detected by one of them, it can rebuild the other one from its local information.

The creation of the agent pair is readily derived from the already existing migration facilities. To create a logger agent, the user agent serializes its state in the same way as for a migration, and sends it to a remote agent place. There, a new agent is created from this data. Different from migration, the new agent does not start the application module that was sent with the state information, and the user agent continues normal execution. Further, the communication unit of the agent is exchanged against a version that first forwards each incoming message to the logger agent before delivering it locally.

The checkpointing strategy in the fault tolerance unit is responsible for chosing apropriate times to save checkpoints. Besides doing it in periodic intervals, migration is a convenient occasion, since it already involves serialization of the agent state.

[1] Note that generalization to agent groups can also be derived.

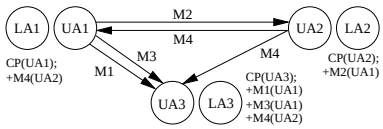


**Fig. 1.** Example for an application with three user and logger agents with checkpoints (CP) and messages (M) [16]

This approach supports tolerance against node and agent failures, even for communicating agents, and thus for distributed applications, and it does not rely on a predetermined itinerary for the agents.

### 3.5 Diagnosis

Following the agent concept, diagnosis should be done autonomously and co-operatively. A user transparent, generally applicable approach implies that the fault tolerance unit of an agent does not have further knowledge about the application, especially about the meaning of its outputs, and thus cannot check them for correctness. So far, the user and logger agents of an agent pair monitor each other through I-am-alive messages and timeouts to detect crash faults.

To exploit application knowledge and enable detection of erroneous computation results, the modularity enables the optional addition of acceptance tests to the application, if the programmer cares to provide them. Via an interface between agent kernel, resp. the fault tolerance unit in the agent kernel, and the application module, the acceptance tests are invoked, and their results given to the fault tolerance unit.

Modification of the standby logger agent to a hot spare, and comparison of messages or checkpoints to detect erroneous computation via differences in the output is possible. However, when this is done in a user transparent way, this requires that no location dependent information (network addresses, time stamps, ...) must be contained in the output [15]. Since the serialization is left to the agent system, this requirement cannot be generally assumed as fulfilled. Thus, this approach is currently not pursued further.

## 4 Analytic Evaluation

This section presents first analytic evaluation results of the FANTOMAS approach based on a Markov model [21]. For the model, the system behavior is abstracted as follows:

The cluster system consists of $n$ nodes. On each node, one agent place exists. Failures of these places caused by failures of the physical node or failures of the runtime environment (e. g. the agent system) will be tolerated. After such a failure, all affected user and logger agents will be recovered. During this recovery time the system is in an insecure state and a second failure generally cannot be tolerated. This assumption is very pessimistic, because a second failure during the recovery could be tolerated if from every agent pair at least one agent is not effected from these failures. The detection of this would need an additional protocol step and is therefor not regarded here.

After completion of all recoveries the system comes back into a secure state and the next failure will be tolerated again. In this model, failed agent places are not assumed to be repaired by repairing the physical node or setting up the runtime environment again. Repairs only apply to the agent system. This means, the agent system is repaired in such way that all affected mobile agents are recovered and all running applications are executed on the remaining agent places.

Two different sets of system states can be distinguished:

1. The states $s_j,\ j \in [0, 2, \ldots, 2n-4]$ are the secure system states after $\frac{j}{2}$ failures of agent places.
2. The states $s_j,\ j \in [1, 3, \ldots, 2n-3]$ characterize the insecure system states during the recovery after $\frac{(j+1)}{2}$ failures of agent places.

In principle, the state $s_{2n-2}$ can be counted as a secure system state. However, in this state only one agent place remains and therefore no other failure can be tolerated. The last state ($s_{2n-1}$) is the final (absorbing) state. This state can be reached by two ways:

1. A failure of an agent place occurs in an insecure system state.
2. A failure of the last remaining agent place occurs.

The rate of failures of agent places is denominated by $\lambda$, the rate of repair by $\mu$. Therewith, the transition rate from a secure state $s_j,\ j \in [0, 2, \ldots, 2n-4]$ to the insecure state $s_{j+1}$ is $\left(n - \frac{j}{2}\right) \cdot \lambda$, because $\frac{j}{2}$ agent places have already failed and $\left(n - \frac{j}{2}\right)$ agent places that can fail are left. The transition rate from an insecure state $s_j,\ j \in [1, 3, \ldots, 2n-3]$ to the following secure state occurs with transition rate $\mu$. The transition rate from an insecure state $s_j,\ j \in [1, 3, \ldots, 2n-3]$ to the final state is $\left(n - \frac{(j+1)}{2}\right) \cdot \lambda$, because $\frac{(j+1)}{2}$ agent places already failed and $\left(n - \frac{(j+1)}{2}\right)$ agent places can fail during this recovery phase. Figure 2 shows this Markov model.

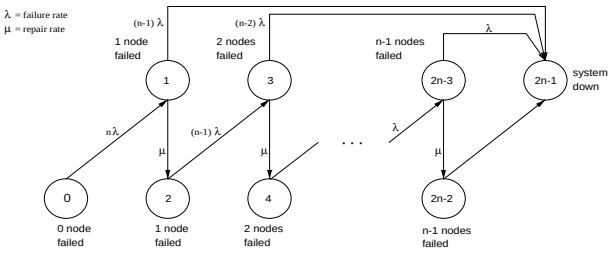


**Fig. 2.** Markov model of the FANTOMAS approach

Let $P(t)$ be the vector of probabilities $P_i(t)$, $i = 0, \ldots, 2n-1$, for the system being in state $i$ at time $t$. $R(t)$ is the reliability of the system, i. e. the probability of not being in state $s_{2n-1}$ at time $t$, $R(t) = P_{2n-1}(t)$. To compute it, the Laplace transform is used, with $\widetilde{P}(s)$ denoting the transform of vector $P(t)$, and $I$ the identity matrix. In

general, $\widetilde{P}(s) = (s \cdot I - Q)^{-1} \cdot P(0)$. At start time $t = 0$, the system is in state $s_0$. The transition matrix $Q$ is obtained from the given Markov model. With this knowledge, we do not need to invert $(s \cdot I - Q)$ entirely, but calculate only the transform of being in the final state, $\widetilde{P}_{2n-1}(s)$. It can be calculated recursively and results to:

$$\widetilde{P}_{2n-1}(s) \;=\; 1 + \left[ \frac{1}{s} \sum_{i=1}^{n-1} \left( -(n-1)\;\lambda\; \frac{\frac{n!}{(n-i)!}\,\lambda^{i}\,\mu^{(i-1)}}{\mathrm{A}(i)\,\mathrm{B}(i)} \right) \;-\; \frac{n!\,\lambda^{n}\;\mu^{(n-1)}}{(s^2 + s\lambda)\;\mathrm{A}(n-1)\,\mathrm{B}(n-1)} \right]$$

with:

$$\mathrm{A}(l) := \prod_{j=1}^{l} (s + (n+1-j)\,\lambda)\,, \quad \mathrm{B}(l) := \prod_{j=1}^{l} (s + (n-j)\,\lambda + \mu)\,.$$

The reverse transformed $P_{2n-1}(t)$ cannot be written in a closed form. Here, we show curves for different parameter sets.

Figure 3 compares the reliability of a simplex system with a system using the FANTOMAS concept, with varied number of nodes. The assumed MTTF of 1000 hours for one agent place can be regarded as a realistic estimation. The MTTR of 2 minutes was chosen after first measurements with the exemplary implementation of the FANTOMAS concept within the FLASH environment. A significantly better reliability for the system using the FANTOMAS concept can be seen, increasing with rising system sizes.

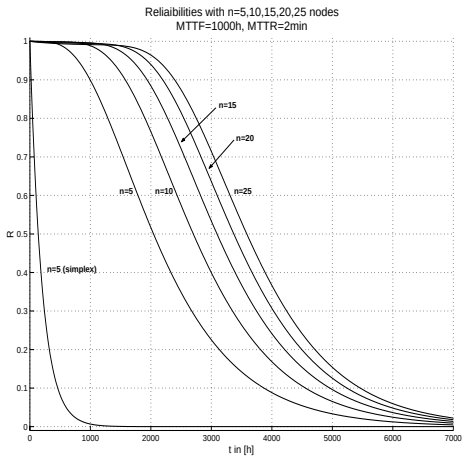


**Fig. 3.** Comparison of systems with varied size (MTTF=1000h)

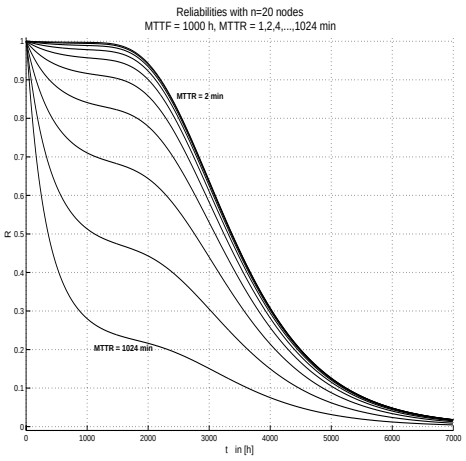


**Fig. 4.** Comparison of system behavior with varied MTTR

Additional system components like the agent system or other used middleware can raise the probability of a failure of an agent place, leading to a worse MTTF. Remembering the possible reasons for a total system shutdown mentioned above, the system's reliability is determined by both kinds of failures. However, decreasing MTTF/MTTR ratios heighten the danger of two or more failures of agent places within a temporal

distance less than the MTTR, leading to a worse reliability in the initial phase. After a certain time, the reliability is more influenced by failures of the second kind and the reliability graphs look like those in figure 3. Figure 4 shows this effect of the relation between the two possibilities of system failures. It shows the reliabilities of systems with varied MTTR, because rising MTTR values lead to the same effect.

## 5 Conclusions and Future Work

We have motivated the use of mobile agents as base for parallel and distributed applications in clusters, and shown the need for user transparent fault tolerance measures for such application environments. From a discussion of the possible fault types, their coverage in related work, and their relevance for agent environments, this work's focus on node and agent failures was derived. General possibilities for achieving fault tolerance in such cases were regarded, and their respective advantages and disadvantages for mobile agent environments, and the intended parallel and distributed application scenarios shown. This leads to an approach based on warm standby and receiver side message logging. It can exploit and augment the facilities, which are already present in the agent to provide the mobility. The analytical evaluation of the approach with a Markov model shows that our approach is feasible and beneficial. First measurements on a prototype implementation within the FLASH environment have been performed and showed a notable overhead during fault-free execution. Currently, the reasons for this overhead are examined closer with the aim to reduce it.

Further work includes the investigation of user transparent hot replication for better diagnosing possibilities and the possibility to employ roll-forward techniques. Also, forward recovery without explicit replication is examined.

## References


1. J. Baumann, F. Hohl, and K. Rothermel. Mole – Concepts of a Mobile Agent System. Technical Report TR-1997-15, Universität Stuttgart, Fakultät Informatik, Germany, 1997.
2. L. F. Bic, M. B. Dillencourt, and M. Fukuda. Mobile Network Objects. In John G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, March 1999.
3. K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.
4. D. Chess and C. Harrison and A. Kershenbaum. Mobile Agents: Are they a Good Idea? Research Report RC 19887 (88465), IBM T.J. Watson Center, March 1995.
5. M. Izatt, P. Chan, and T. Brecht. Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications. In *Proc. ACM 1999 Conference on Java Grande*, pages 15–24, June 1999.
6. D. Johansen, R. van Renesse, and F. B. Schneider. An Introduction to the TACOMA Distributed System, Version 1.0. Report, Institute of Mathematical and Physical Science, Department of Computer Science, University of Tromsø, Norway, 1995.
7. D. Johansen, R. van Renesse, and F. B. Schneider. Operating System Support for Mobile Agents. In *Proceedings of the 5th. IEEE Workshop on Hot Topics in Operating Systems, Orcas Island, USA, May 1994*, pages 42–45, May 1995.

8. R. Kluthe, W. Obelöer, and C. Grewe. Agent-based Load Balancing for Mobile Robot Application. In *Distributed and Parallel Embedded Systems – Proc. of the Int. Workshop IFIP WG 10.3/WG 10.5, October 1998 (DIPES'98)*, pages 117–126. Kluwer Academic Press, Boston, 1999.
9. D. S. Milojičić, S. Guday, and R. Wheeler. Old Wine in New Bottles, Applying OS Process Migration Technology to Mobile Agents. In *Proceedings of the 3rd Workshop on Mobile Object Systems, 11th European Conference on Object-Oriented Programming*, June 1997.
10. Y. Minsky, R. van Renesse, F. B. Schneider, and S. D. Stoller. Cryptographic Support for Fault-Tolerant Distributed Computing. In *Proc. 7th ACM SIGOPS European Workshop*, pages 109–114. ACM Press, September 1996.
11. Mitsubishi Electric ITA, Horizon Systems Laboratory. Mobile Agent Computing. White Paper, 1998.
12. A. L. Murphy and G. P. Picco. Reliable communication for highly mobile agents. Report (WUCS-99-15), Washington University in St. Louis, 1999.
13. W. Obelöer, C. Grewe, and H. Pals. Load Management with Mobile Agents. In *Proceedings of the 24th EUROMICRO Conference*, volume II, pages 1005–1012. IEEE Computer Society, August 1998.
14. H. Pals. Parallel, Fault-Tolerant Applications with Mobile Agents. Diploma thesis, Institut für Technische Informatik der MU Lübeck, December 1999. (in German).
15. S. Petri. A Common Framework for Transparent Checkpointing, Replication and Migration in Clusters. In *ARCS'99: Architektur von Rechensystemen, Vorträge der Workshops im Rahmen der 15. GI/ITG-Fachtagung*, pages 79–88, Jena, October 1999.
16. S. Petri and C. Grewe. A Fault-Tolerant Approach for Mobile Agents. In *Dependable Computing – EDCC-3, Third European Dependable Computing Conference, Fast Abstracts*. Czech Technical University in Prague, September 1999.
17. V. A. Pham and A. Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, 36(7):26–37, July 1998.
18. B. Randall. System Structure for Software Fault Tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232, June 1975.
19. S. Rao, L. Alvisi, and H.M. Vin. Hybrid Message Logging Protocols for Fast Recovery. In *Digest of FastAbstracts: FTCS-28 – The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pages 41–42. IEEE Computer Society, June 1998.
20. K. Rothermel and R. Popescu-Zeletin, editors. *Mobile Agents*, volume 1219 of *LNCS*. Springer, 1997.
21. R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliablity Analysis of Computer Systems*. Kluwer Academic Publishers, Boston, 1996.
22. M. Strasser and K. Rothermel. Reliability concepts for mobile agents. *International Journal of Cooperative Information Systems (IJCIS)*, 7(4):355–382, 1998.
23. H. Vogler, T. Kunkelmann, and M.-L. Moschgath. An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions. In *Proc. 1997 International Conference on Parallel and Distributed Systems (ICPADS '97)*. IEEE Computer Society, December 1997.
24. J. White. Mobile Agents. White Paper, General Magic, Inc., Mountain View, 1996.
25. D. Wong, N. Paciorek, and D. Moore. Java-based mobile agents. *Communications of the ACM*, 42(3):92–102, March 1999.
26. M. J. Wooldridge and N. R. Jennings. Agent theories, architectures and languages: A survey. In *ECAI-94 Workshop on Agent Theories, Architectures and Languages*, number 890 in LNAI, pages 1–39. Springer, August 1994.