Transient Fault Tolerance in Mobile Agent Based Computing

Goutam Kumar Saha

Scientist-F, CDAC, Kolkata, India

Mailing Address: CA – 2 / 4 B, Baguiati,Deshbandhu Nagar, Kolkata 700059, West Bengal, India

gksaha@rediffmail.com

Abstract

Agent technology is emerging as a new paradigm in the areas of distributed and mobile computing. Agent is a computational entity capable of relocating code, data and execution- state to another host. Mobile agents' code often experience transient faults resulting in a partial or complete loss during execution at a host machine. Protocol for fault – tolerant agent prevents a partial or complete loss of a mobile agent at a host. This article describes how to detect and recover random transient bit-errors at an agent before starting its execution at a host after its arrival at a host, in order to maintain availability of an agent by comparing an agent's states by using time and space redundancy. In this proposed self-repair approach, a software fix for fault – tolerance exists along with an agent. This generalized scheme is useful for recovering any kind of distributed agents against hardware transient faults (at a host). This paper presents a fault-tolerance mechanism for mobile agents that attempts to detect and correct any bit errors that may occur at a host after agents' mobility on a Web Agent-based Service Providing (WASP) platform. Though in modern distributed systems, the communication stack handles any bit errors and error correction is used on multiple layers (for example, in transport layer), the proposed approach is intended to be a supplement one to the conventional error detecting and correcting codes.

Keywords: Mobile Agent, transient fault detection, recovery, and tolerance.

Received June 21, 2005 / Accepted September 15, 2005

1. Introduction

It is not uncommon to observe that an agent like a mobile agent often experiences transient faults (resulting in partial or complete loss) during its execution at a host. The proposed work is intended to detect and recover such errors in an agent code in order to increase its availability and thus, to complement the intrinsic error detection mechanisms of a mobile agent system. There is significant attention within the mobile agent fault-tolerance community concerning the loss of mobile agents at remote agent servers that fail by crashing in [10,16,17,18,19,20]. The proposed generalized approach establishes test conditions within a single version of the software program, rather than requiring multiple and distinct versions of the same program running concurrently on many machines.

1.1. Agent Overview

Agent based computing is a new software paradigm in Information Technology today. Agents are autonomous programs situated within an environment, which sense the environment and acts upon it using its knowledge base to achieve its goals. A software agent is a piece of software that is autonomous and owned by some party. Mobile agents travel through a network of heterogeneous machines. They have certain special properties, which make them different from the standard

programs such as mandatory and orthogonal properties. Autonomy, reactive, proactive and temporally continuous is the mandatory properties of the agents. The orthogonal properties are communicative, mobile, learning and believable in [3,8]. Agents can be classified based on properties they posses: local or user interface agents, networked agents, distributed artificial intelligence (AI) agents and mobile agents. Among these agents, the networked agents and user interface agents are single agent systems whereas the other two agents are multi- agent systems. The agents of single agent systems are assumed to never cooperate or communicate with each other whereas the distributed AI and mobile agents are multi-agent systems.

1.2. Mobile Agent

Distributed systems and on a wider scale, the Internet, are inherently complex in the presence of asynchrony, concurrency and distribution. Mobile agent introduces new level of complexity, operating within an environment that is autonomous, open to security attacks (directed by malicious agents and hosts), agent server crashes, and failure to locate resources in [17]. Mobile agent is an itinerant agent dispatched from source computer. It contains program, data and execution state information and it migrates from one host to another host in the heterogeneous network and executes at remote host to accomplish its task in [21]. Mobile agents provide a significant performance benefit compared to the conventional client-server paradigm. Conventionally static clients and servers interact using message passing over a communication channel. On receiving a request from a client, the server processes the request and returns the results over the communication channel. Network bandwidth remains under - utilized for large data sets. Mobile agents are software programs that move from one host to another. The mobile agent approach is based on sending the computation to the data, computing results and accessing local resources. Subsequently results are returned to the originating host. It is capable of relocating code, data, and Execution State to other host.

Fault tolerance is fundamental to further development of mobile agent based applications. Fault tolerance is the ability of a system to perform its function correctly even in the presence of internal faults. Based on duration, fault can be classified as transient or permanent. A transient fault is not reproducible because it will eventually disappear, whereas a permanent one will remain unless it is removed by some external agency. A particularly problematic type of transient fault is the intermittent fault that recurs, often unpredictably. General fault tolerance procedure includes error detection and error recovery. Error detection is the process of identifying that the system is in an invalid state.

Mobile agent systems offer various useful provisions to the user community for accessing the network resources placed anywhere from anywhere. This new paradigm of distributed in [7] systems has the unique property of making it possible for partial execution of agent' s code on more than one platform. This extension of code distribution that was not present in the traditional distributed systems makes the mobile systems very useful. In mobile agent systems, the runtime codes of mobile agents along with the data are distributed across the network. However, the transient nature of the mobile agent system makes the design policies more complex than that of traditional systems in [2,13]. During the migration of the program, the agent' s code along with its execution state is transported to other hosts, which

may be located on heterogeneous platform, and there the agent resumes its execution. Mobile agent systems experience the problems because of their mobility, portability and wireless communications. The problems that arise with the mobility dimension are how do these systems incorporate the address, process, and object migration from host to host. The problem due to portability is that the mobile code must be able to do its job with the use of limited resources without any loss of efficiency. The resources can be of any form like power, storage etc. The main hurdles in wireless type of communications are the heterogeneity, frequent disconnection in [15] and constant changes in the environment parameters in [14]. However, mobile agent systems may experience transient faults often during its transit or when an agent server experiences electrical transients. And then it is very difficult to maintain the availability of mobile agent because the developers do not have control over the remote agent servers. Though the basis of a mobile agent system is the wireless communication, it can also execute on hosts linked to a fixed network. A mobile agent is lost when an agent server experiences transient faults during an agent' s execution. The approach in [20] proposes that a mobile agent inject a replica into stable storage upon arriving at an agent server. However, in the event of an agent server crash, the replica remains unavailable for an unknown time period.

1.3. Objective & Scope of the work

The objective of this article is to detect and recover an agent' s random transient-bit-errors that might have occurred after its arrival at an agent server or before its execution starts or before it is able to store its three images on a host' s stable storage In this approach, the protocol that provides fault tolerance travels with the agent. This has the important advantage to allow fault-tolerant mobile agent execution without the need to modify the underlying mobile agent platform (in our case on a Web Agent-based Service Provider (WASP)). A mobile agent injects two replicas into stable storage upon its arrival at each agent server. It is intended that the proposed approach will be executing on a Web Agent-based Service Provider (WASP) platform. The WASP architecture is shown in figure 1. The WASP includes a WWW server together with an agent-specific part called Server Agent Environment (SAE). It supports Java as the implementation language since most WWW servers support Java' s Servlet interface (attached to SAE) and because of Java' s ubiquitous availability-especially in Web browsers, which agents use to communicate with the user. WWW server redirects all agent-related request (e.g., agent start, agent migration) to WASP' s SAE. Agents may be started by an HTTP request to a URL designating a particular agent type on some server. The actual start of an agent is done by the server' s SAE. After being loaded and initialized by the SAE, the agent may send its Java-based GUI to the user' s browser. The transfer of migrating agents is realized with an HTTP post request to a SAE specific URL, at target WWW server. The WASP platform provides the necessary services and tasks for example, mobility support, resources management, execution support, communication support and security. The WASP platform supports these tasks by relying on established Java distributed computing in [4] concepts and, more importantly, by integrating agent environments into WWW-servers in [5,11] with the help of server extension modules. As an additional benefit of relying on the well-established WWW

infrastructure, the WASP platform may easily be deployed in the Internet. HTTP protocol is used for agent transfer or control. In an agent-based computing scenario, hosts must provide a kind of a "docking station" for mobile agents, which act as a local environment or agent platform. We believe that communication stack will control transient bit errors during an agent' s transit. Therefore this proposed scheme does not aim to detect bit errors during an agent' s mobility through a communication channel. The proposed scheme is not meant for wireless environments because of weakest link with limited bandwidth and throughput. An on-line algorithm or a software fix has been designed in order to detect and recover (at the beginning of the execution of an agent at a host) various random bit- errors or byte -errors at an agent code that might occur at its arrival at a host. We assume that transient bit errors at agent code and proposed protocol that may get induced during its transit will be handled by Error Correction mechanisms at the transport layer. We assume that agent' s run-time bit-errors that may get induced after the execution of an agent has started, are detected by the intrinsic Error Detection Mechanisms (EDM) of a system (exceptions, memory protection, checksums etc.). After its arrival, an agent code injects two replicates (i.e., it becomes triplicate) at a host. We execute the triplicate images of an agent at a host sequentially with similar inputs (or by using buffer for inputs like system time etc.,) and at the end of execution of the images of an agent, we compare the images' answers in order to find an answer in majority that is considered to be a correct answer. Such scheme is intended to mask an erroneous answer from the execution of triplicate images at a host on a WASP environment. In such case, our dependency on EDM of a host is raised. However, if no answer in majority is found then such fault-masking scheme crashes. The proposed software protocol is intended to complement the intrinsic EDM of an agent server. To make mobile agents resilient to transient bit-errors, the proposed work needs an agent to be replicated into three images after its arrival at a host. The software fix verifies and corrects byte after byte of the images in memory using space and time redundancy. Thus the proposed work is intended to verify and recover an agent that might have experienced transient faults upon its arrival at a host. Again, in order to detect errors during the execution time of an agent, we rely on the EDM (checksums scheme) at the host.

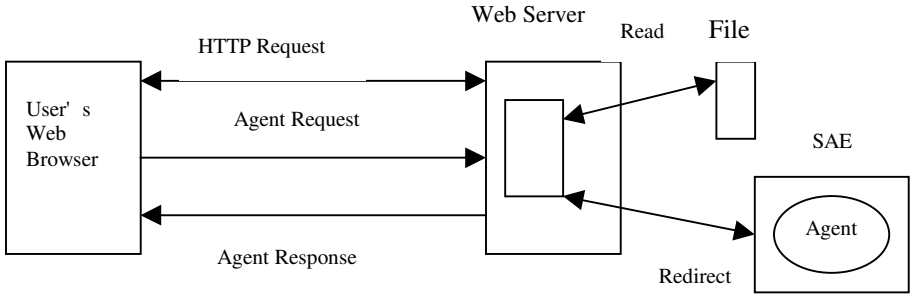

Figure 1. The WASP Platform Architecture

1.4. Mobile Agent Environment

We may also consider the Mobile Agent Code Environment (MACE) [Torsten Stolpmann, 1997], which is an interpretative execution environment for mobile agents and the software fix programmed in C or C++. The code is based upon the model of an abstract machine. The MACE compiler generates byte code derived from the internal representation of Register Transfer Language (RTL) code produced by the GNU C compiler from C or C++ source. This byte code consists of pseudo machine language operations of an abstract microprocessor within a virtual memory image, which is interpreted by the MACE Interpreter. This allows a safe and efficient execution and the simple migration of agent code. However, for the agents on hosts linked to fixed network, we can use compiling - systems. Like any interpretative system MACE is also subject to an inherent performance penalty compared to compiling-systems.

1.5. Contribution

Like any other fault tolerance approach, the proposed generalized scheme also inherits an overhead (of the order of three) in both time and space. In order to tolerate potential transient faults we cannot avoid such time and space redundancy. Though a self-stabilizing distributed algorithmic technique running on each node on the network, is useful for bringing a system to a legal configuration in a finite number of steps from an arbitrary illegal system configuration caused by message corruption or sensor malfunction, it suffers from high time redundancy. Other fault tolerant schemes like, Algorithm Based Fault Tolerance in [6], Assertions in [12] etc., suffers from the lack of generality and wide applicability. Other costly schemes like, N-Version Programming (NVP) in [1], Triple Modular Redundancy (TMR) relies upon multiple versions of software and hardware. However, the proposed scheme is a low-cost solution because this relies on only one version of agent software, which is enhanced with the protective software fix. This approach does not lack in generality and applicability. This is intended to complement the intrinsic EDM in agent server systems towards tolerating random bit-errors. In other words, the proposed work is based on an enhanced single version-programming (ESVP) scheme using the extra protective code. The proposed technique needs three replicas or images of the agent code. It is assumed that an agent platform will allow its replication or reception. It does not need multiple independently developed versions of the agent-code. Rather it uses only an enhanced single – version (ESV) agent-code. It tolerates (detect & recover) one byte -fault in every three bytes at the same offset or displacement inside the three images of an agent. However, it detects all the three bytes-errors because the probability that all three corrupted bytes at an offset in the images will represent the same value is negligibly small (2^{-24}), because the chance of all three similar bytes, at various locations, getting altered to some other similar value is ($1/2^8 * 1/2^8 * 1/2^8$), as transient caused errors are random and independent in nature. Only one image of an agent executes on an agent - server while the other two images of the same agent code are used for the purpose of fault detection (through state comparison) and recovery thereof. This paper does not address the synchronization issues (e.g. lock-step execution) because all the three images of an agent-code are not executed simultaneously; rather only one image or copy of an agent is executed here on a host.

2. Work Description

The proposed approach is capable of detecting transient faults or soft errors in the mobile agent's code and data. The objective of this software is to verify and repair the random byte-errors (induced upon its arrival at a host) in a mobile agent before its execution on a host starts. This is necessary to prevent error propagation. This software technique is a useful tool for fixing fault tolerance in mobile agent systems. We assume that byte-errors that may occur after a mobile agent starts execution at a host will be detected by a host' s intrinsic EDM in the case when we do not execute all the three copies of an agent in a sequence in order to vote upon their answers for a majority one. Alternately, on using extra time redundancy (if allowed) we compute run-time checksums of the image in execution and compare it with the previously computed (and saved) checksums, for detection of run-time errors that might occur once the execution of a mobile agent has started. In such case, we have the option to restart the mobile agent' s execution in order to gain a fail-safe kind of fault tolerance. Thus, ambiguities in computed answers are also avoided. We cannot rely entirely on the various conventional error- correcting codes or EDM (implemented by hardware circuits at a host) though they are capable of both detecting and correcting certain types of errors. They are not free from limitations. For example, adding six extra bits per sixteen-bit word, single or double can correct bit errors. However, random multiple- bit errors are very common type of errors that are caused by potential transients. Conventional off-line error -corrective fixes can detect but cannot correct all random multiple- bit errors.

The pseudocode of the proposed protocol for detecting and correcting bit errors in a mobile agent upon its arrival is described by a procedure namely, Mobile_FLT_*DET_REP* (as shown in Table-1), for fault check and repair.

ALGORITHM- ***Mobile_ FLT_DET_REP***

The following steps (pseudocode) describe how the *Mobile_FAULT_DET_REP* algorithm verifies for the presence of multiple number of byte - errors in the mobile agent code and data, and how it corrects the erroneous bytes of the contaminated agent code, by comparing an agent' s state (using three images of an agent that are injected at a host upon its arrival) byte by byte. The symbols "/* */" are to enclose remarks only.

/* There are three images or replicas of an agent that are injected at a host. Byte-wise comparisons and corrections are carried out (using XOR) till the last byte of the images. It verifies the corresponding three bytes at an offset say, *d*, of the three images of the agent code and if any byte error has occurred, then it repairs the corrupted byte. Starting addresses of the three images are known. The notation IG^1_d denotes the *d* th byte (at an offset say, *d*) in the agent' s first image or Image –1. */

Step 1. Set *S* = Size of an image in bytes. /*size of an image in bytes is known*/

Step 2. Set *d* = 0 /* initialize the memory offset say, *d*/

Step 3. $R_{12} = IG^1_d .XOR. IG^2_d$ /*bytes at offset *d* in images IG^1 , IG^2 are -

Step 4. If R_{12} .EQ. 0 , Then: - XORed and result is stored at R_{12} */

No Error. /* goto step 5 i.e., program control goes out of the outer -
EndIf of step-4, for scanning the next byte of the three images */

Else:

$R_{13} = IG^1_d$.*XOR*. IG^3_d

If R_{13} .*EQ*. 0 , Then:

$IG^2_d = IG^2_d$.*XOR*. R_{12} /* Bytes at IG^1_d ,IG^3_d are correct but IG^2_d is bad, so the erroneous byte at IG^2_d is repaired*/

Else:

$R_{23} = IG^2_d$.*XOR*. IG^3_d

If R_{23} .*EQ*. *0*, Then:

$IG^1_d = IG^1_d$.*XOR*. R_{12}

/* Bytes at IG^2_d , IG^3_d are correct but IG^1_d is bad, so the erroneous byte at IG^1_d is repaired*/

Else:

Call *HARD_ERR* /*All the three bytes at the same offset *d* in the three images are corrupted i.e. all the three images are corrupted - indicates a memory device problem or permanent errors, Call *HARD_ERR* routine to restart the agent execution. */

{End of If structure}

{End of If Structure }

{End of If Structure }

Step 5. Set $d = d + 1$ /* offset *d* is incremented by one to scan next byte */

If $d < S$, Then: /* Scan the next byte for error detection & - repair thereof */

GOTO Step 3.

Else:

Return /*After the entire scan & repair, starting from 0^{th} byte through (S-1) th byte in the 1^{st},2^{nd} and 3^{rd} images simultaneously, program control goes back to the primary image of agent and agent execution continues in an application based on mobile agents */

{ End of If structure }

[End of Algorithm – *Mobile_FLT_DET_REP*]

Table 1. Algorithm for Mobile Agent Error Detection and Repair

3. Discussion

Mobile_FLT_DET_REP is invoked for the error detection and recovery of an agent. Algorithm-*Mobile_FLT_DET_REP* shows the steps involved in detection and correction of errors of the agent code. Whenever an agent visits a host, it injects two

replicas at host and we need to store it in the memory of the computing host. The starting addresses of the three images are for example, say, IG^1, IG^2 *and* IG^3 respectively. When the offset *d* is of say 0 (initial value), then the address IG^1_0 denotes the starting address of first image IG^1 only, because IG^1_0 means the value of $(IG^1 + 0)$ i.e., the starting address plus offset. In general, if IG^N were the starting address of the N th image then, the address of the *d* th byte (or at offset say, *d)* is shown by equation (1).

$$IG^N_d = IG^N + d \quad \ldots\ldots (1)$$

Again, if any one byte out of the three corresponding bytes of the three images at an offset say, *d*, is corrupted, then this routine can repair the corrupted byte by *XOR* ing. The wrong byte is detected by comparing three bytes at the same offset, as shown at step 3 and step 4 of the Algorithm. If two corresponding bytes content are same then XORing result is zero i.e., *00000000 or 00H*. Otherwise, result is a nonzero value. In general, if the two byte – contents of p th and q th images at an offset say, *d*, are say, IG^p_d, IG^q_d and if these two values are not corrupted, then the following equation is true.

$$IG^p_d \ .XOR. \ IG^q_d = 0 \quad \ldots\ldots (2a)$$

Otherwise, if the two-byte contents are not same, then the equation (2a) will not be satisfied. In other words, relation (2b) is true.

$$IG^p_d \ .XOR. \ IG^q_d \bullet 0 \ldots\ldots (2b)$$

The possibility of getting inadvertently alteration (by transients) of two similar bytes in two images (located at two distant locations) to mean a some other value resulting in a similar corrupted byte-pattern, is negligibly small. In other words the chances of byte error remaining undetected is:

$$(1 / (2^8)) * (1 / (2^8)) = 1 / (2^{16}) \quad \ldots\ldots (3)$$

This method is capable of detecting even 8 bit errors i.e., even an entirely corrupted byte can be detected. If say, IG^p_d byte is corrupted to IG^{p*}_d. But say, IG^q_d and IG^r_d byte- contents remain same at an offset *d* (i.e., are not affected by transients) in the images IG^q and IG^r and then by comparing three corresponding bytes of the three images, we can detect that IG^p_d byte is corrupted (as shown at step-3 and step-4 of the Algorithm). The corrupted byte is repaired in the following way. This is applicable even for 8 bit errors in a byte. For an example, the original bit-pattern of IG^p_d is *1001 1101* and after the worst case corruption (all eight bit alterations), the byte- pattern of IG^p_d is say, $IG^{p*}_d = 0110\ 0010$ then the result (in R_{qp}) on *XOR* ing these two byte- patterns will be: $\boldsymbol{R_{qp}} = IG^q_d$ *.XOR.* $IG^{p*}_d = \boldsymbol{1111\ 1111}$ Now, the byte -pattern on carrying out (IG^{p*}_d *.XOR.* $\boldsymbol{R_{qp}}$) will be *1001 1101* i.e., corrupted byte pattern IG^{p*}_d is repaired or corrected. If there is no error in an agent program and data code, then the following equation will be satisfied.

$$IG^p_d = IG^q_d = IG^r_d \ldots\ldots (4a)$$

The chance of satisfying the equation (4a) by the corrupted three bytes of the three images at the same offset is negligibly small, because the transients' effects on memory, registers are very random and independent in nature.

$$(1/(2^8)) * (1/(2^8)) * (1/(2^8)) = 1 / (2^{24}) \ldots\ldots (4b)$$

In other words, the chances of three bytes at different locations corresponding to a particular value with similar byte-pattern, getting altered simultaneously to a similar value in order to satisfy equation (4c) is negligibly small. Here, IG^{p*}_d denotes a corrupted byte of the image–*p* at an offset *d.*

$$IG^{p*}_d = IG^{q*}_d = IG^{r*}_d \ldots\ldots (4c)$$

The chance of a value stored at an offset say, *d*, in all the three images getting altered simultaneously into some other three different values, is negligibly small. Such disastrous effect indicates a possibility of memory device hardware or permanent errors and

then *HARD_ERR* routine is invoked for necessary recovery thereof (e.g., restarting the agent code). In other words, the possibility of invoking the error routine namely, *HARD_ERR* (as shown at step-4) is negligibly small (2^{-24}).

The routine *Mobile_FLT_DET_REP* verifies and recovers byte errors in the entire mobile agent code, by increasing the value of offset *d* from 0 to S–1 (the size of an image is of say, *S* bytes). This is very effective for soft errors (induced during its transit or at reception at a host) detection and corrections of agent code prior to its execution on a host. After detecting and repairing the entire agent code, program control goes to start the execution of a mobile agent' s image. Even a totally corrupted agent image can be repaired by this proposed technique of repairing byte by byte.

3.1. Experimental Results

The capability of the proposed scheme for detection and correction of transient bit errors of a mobile agent upon its arrival at a host and before executing an agent at a host is assessed here. It is observed that hardware Error Detection Mechanisms (EDM) at a host could detect 25% errors whereas the software fix could detect 37% errors in a fault injection (i.e., causing random bit errors) experiment based on a compiler SingleStep7.4 of SDS Inc, for a Motorola 68040 processor (using simple C programs as benchmarks). Remaining errors were found to be fail-silent (without changing program behavior). Space redundancy of this proposed technique is about three.

Time and Space redundancy remain at the same order for domains with higher number of agents running in a host. However, because of the lower economic trend on the hardware prices, this much space redundancy can be easily afforded. This higher time redundancy (of the order of three) can be affordable when we use an affordable high-speed processor and memory.

4. Merits & Limitations

The proposed technique is promising enough to detect multiple soft errors and corrections thereof with an affordable redundancy in both time and memory space for gaining higher fault-tolerance. This is also very useful for the system maintenance engineers for the work of faster memory scrubbing i.e., for periodic rewriting and correction of data stored at all memory addresses. The algorithm based on bit operator *XOR* is meant for faster fault detection and recovery. However it does not have the overhead of multiple software versions, hardware redundancy and synchronization, in order to obtain the agent code' s transient fault - immunity. It is a low cost solution. It is suitable for self – stabilizing the agent code against the transient caused random byte errors. The stabilizing time includes the time redundancy for comparing the bytes of the images and for correction thereof. The disastrous event of entire damage of all the three images of an agent is also detected by it without resending agent code. But, for recovery of such disastrous event, an error routine *HARD_ERROR* is invoked for restarting or reloading the agent code execution from stable memory. If one byte among the three bytes (at an offset say *d* in the three images) is corrupted, then this algorithm can detect and recover it. Even, it detects the consequences of two or three corrupted bytes in the images; but for recovery, we invoke the *HARD_ERROR* routine for reloading from stable storage. Even if the three images get corrupted at random locations with random byte- errors causing random values, this technique proves to be useful also (as shown at 4(b)) for detection and recovery

through the *HARD_ERROR* routine. This is a useful approach towards the transient fault – tolerant mobile agent. However, in the conventional methods (e.g., NVP, TMR), there exists higher redundancy in both time and memory space, and in synchronization. For example, an NVP scheme needs *(n+2)* number of versions (or variants) and machines in order to tolerate *n* number of sequential faults. Whereas, this proposed technique, based on so-called ESVP (Enhanced Single Version Programming), needs *(n +2)* number of images (or copies) of a single-version (SV) mobile agent and one reliable fast machine only for tolerating (detection & recovery) n number of sequential faults. In other words, to tolerate two sequential faults this ESVP technique needs only four replicas of the same version of the application and one machine only. The proposed approach is not intended to tolerate agent software design bugs. The other merits of this approach include its lower -cost, generality and wider applicability. Indeed, if the host the agent is executing on has crashed, the agent is not available. The limitation of the *Mobile_FLT_DET_REP* routine is its inability to detect errors that may occur after the execution of an agent has started. In case a mobile agent' s host crashes, we can use mobile shadow in [9] for better availability by resending a shadow mobile agent from its parent host. In order to address run time errors that are occurring during the execution of an agent, we use a checksum scheme or we may adopt (if more time overhead is permitted) the fault-masking scheme to run all the images of an agent and at the end of execution of all images, we compare the answers in order to find an answer in majority.

5. Conclusion

The proposed generalized approach establishes test conditions within a single version of a mobile agent program, rather than requiring multiple and distinct versions of the same mobile agent program running concurrently on many machines. This scheme is able to detect and repair transient errors in mobile agent code upon its arrival at a host. This is effective at the cost of an affordable redundancy in both time and space, without an increased monetary budget. We believe that in a few years, the computing power, disk capacity and wireless bandwidth will be abundant on small footprint devices (cell phones, etc,) also and then, it will not be difficult for us to operate measurements and thorough experiments on this so called ESVP based protocol for wireless platform also. However, for more robustness, a few images (say, 3) of the *Mobile_FLT_DET_REP* routine can also be used in order to tolerate faults (say, one sequential fault) inside this decider routine itself. This is also applicable to an NVP system that needs three versions of the decider program in order to tolerate a fault inside the decider. It can be used as a very effective tool for the system engineers for designing and maintaining a robust mobile agent for a mobile computing application. It provides reliability in a mobile computing application using a WASP and WWW server, through on-line errors-detection and corrections and thus, it eliminates ambiguities, arising due to agent code–corruptions at random. This software fix has wider applicability including scientific computations and embedded systems. This work is intended to supplement the existing EDM at a mobile host.

6. References

[1] Avizienis, A. The N-Version Approach to Fault Tolerant System, IEEE Trans. Software Engineering, v. SE-11, n.12, p.1491-1501, 1985.

[2] Borselius, N. Mobile Agent Security, Electronics & Communication Engineering Journal, v.14, n.5, IEE, UK, 2002.

[3] Dasgupta, P., Narasimhan, N., Moser, L. and Smith, P.M. MAGNET: Mobile Agents for Networked Electronic Trading, IEEE Transactions on Knowledge and Data Engineering, v.24, n.6, p. 509-525, 1999.

[4] Funfrocken, S. Migration of Java-based Mobile Agents- Capturing and Reestablishing the state of Java Programs, Proc. of the 2nd Intl. Workshop on Mobile Agents, p.26-27, 1998.

[5] Funfrocken, S. How to Integrate Mobile Agents into WEB servers, Proc. of the WETICE' 97 Workshop on Collaborative Agents in Distributed Web Applications, Boston, p.94-99, 1997.

[6] Huang, K.H. and Abraham, J.A. Algorithm-Based Fault Tolerance for matrix perations, IEEE Transactions on Computers, v.c-33, n.6, p.518-528, 1984.

[7] Marris, J., Satyanarayan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S., and Andrew, F.D. A Distributed Personal Computing Environment, ACM Communication, v.29, 1986.

[8] Nagamuta,V.and Endler, M. Coordinating Mobile Agents through the Broadcast Channel, Proc. SBRC, Florianopolis, 2001.

[9] Pears, S, Xu, J. and Boldyreff, C. Mobile Agent Fault Tolerance for Information Retrieval Applications: An Exception Handling Approach, Proc. of the DSN'2001, 2001.

[10] Pleisch, S.and Schiper, A. Modeling Fault-Tolerant Mobile Agents as a Sequence of Agreement Problems, Proc. of the 19th Symp. on Reliable Distributed Systems (SRDS), Nuremberg, p.11-20, 2000.

[11] Pleisch, S. and Schiper, A. FATOMAS- A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach, Proc. of the Intl. Conf. on Dependable Systems and Networks (DSN 2001), 2001.

[12] Rela, M.Z., Madeira, H. and Silva, J.G. Experimental Evaluation of the Fail-Silent Behavior in Programs with Consistency Checks, Proc.of the FTCS -26, p.394-403, 1996.

[13] Rothermel, K., et.al, Mobile Agent Systems : What is Missing?, Proc. of the DAIS, 1997.

[14] Saha, G.K. Fault Management in Mobile Computing, ACM Ubiquity, v.4, n.32, ACM Press, USA, 2003.

[15] Satyanarayan, M., Kistler, J.J., Mummert, L.B., Ebling, M.R., Kumar, P. and Lu, O. Experience with Disconnected Operation in a Mobile Computing Environment, Proc. of 1993 USENIX Symposium on Mobile & Location Independent Computing, Cambridge, MA, 1993.

[16] Schneider, F. Towards Fault-Tolerant and Secure Agentry, Proc. of the 11th Intl. Workshop on Distributed Algorithms, Sarbrucken, p.1-14, 1997.

[17] Silva, L.M., Batista, V. and Silva, J.G. Fault - Tolerant Execution of Mobile Agents, Proc. of the Intl. Conf. on Dependable Systems and Networks, New York, p.144-153, 2000.

[18] Silva, F.M. and Zeletin, R.P. Mobile Agent - Based Transactions in Open Environments, IEICE Transactions on Communications, E83-B(5), p.973-987, 2000.

[19] Strasser, M., Rothermel, K. and Maihofer, C. Providing Reliable Agents for Electronic Commerce, Proc. of the TREC' 98, LNCS 1402, Springer-Verlag, p.241-253, 1998.

[20] Vogler, H., Hunklemann, T. and Moschgath, M. An Approach for Mobile Agent Security and Fault Tolerance Using Distributed Transactions, Proc. of the Intl. Conf. on Parallel and Distributed Systems (ICPADS' 97), Seol, p.268-274, 1997.

[21] Wong, D., Paciorek, N. and Moore, D. Java based mobile agents, Communications of ACM, v. 42, n.3, p.92-102, 1999.

Author's Biography

In his last seventeen years' research and development experience, he has worked as a Scientist in LRDE, Defence Research & Development Organisation, Bangalore, and at Electronics Research & Development Centre of India, Calcutta. At present, he is with the Centre for Development of Advanced Computing, Kolkata, India, as a Scientist-F. He has authored around one hundred research papers in SAMS Journal, ACM, C&EE J., IEEE, CSI, JISE etc. He is a senior member in IEEE, Computer Society of India, ACM, and Fellow in IETE. He has received various awards, scholarships and grants from both the national and international organizations. He is a reviewer of the CSI Journal (India), AMSE Journal (France & Spain) and of an IEEE Magazine. He is an associate editor of the ACM Ubiquity (ACM Press, USA). His field of interest is on dependable, fault tolerant computing and natural language processing.