IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 3, May 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Fault-tolerant Mobile Agent-based Monitoring Mechanism for Highly Dynamic Distributed Networks

Jinho Ahn[1]

[1] **Dept. of Computer Science, College of Natural Science, Kyonggi University**
Suwon, Gyeonggi-do 443-760, Republic of Korea

Abstract

Thanks to asynchronous and dynamic natures of mobile agents, a certain number of mobile agent-based monitoring mechanisms have actively been developed to monitor large-scale and dynamic distributed networked systems adaptively and efficiently. Among them, some mechanisms attempt to adapt to dynamic changes in various aspects such as network traffic patterns, resource addition and deletion, network topology and so on. However, failures of some domain managers are very critical to providing correct, real-time and efficient monitoring functionality in a large-scale mobile agent-based distributed monitoring system. In this paper, we present a novel fault-tolerance mechanism to have the following advantageous features appropriate for large-scale and dynamic hierarchical mobile agent-based monitoring organizations. It supports fast failure detection functionality with low failure-free overhead by each domain manager transmitting heart-beat messages to its immediate higher-level manager. Also, it minimizes the number of non-faulty monitoring managers affected by failures of domain managers. Moreover, it allows consistent failure detection actions to be performed continuously in case of agent creation, migration and termination, and is able to execute consistent takeover actions even in concurrent failures of domain managers.

Keywords: *Distributed Network, Fault-tolerance, Mobile Agent, Scalability, Takeover.*

1. Introduction

Recently, as the number of users of distributed systems and networks considerably increases with the increasing complexity of their services and policies, system administrators attempt to ensure high quality of services each user requires by maximizing utilization of system resources [5]. To achieve this goal, correct, real-time and efficient management and monitoring mechanisms are essential for the systems. But, as the infrastructures of the systems rapidly scale up, a huge amount of monitoring information is produced by a larger number of managed nodes and resources and so the complexity of network monitoring function becomes extremely high [1]. Also, there are heterogeneous and various network environments within the systems needed to be monitored and the nature of managed resources becomes almost dynamic, not static, which forces traditional static centralized and distributed monitoring mechanisms to be unsuitable for the systems [10]. Thus, mobile agent-based monitoring mechanisms have actively been developed to monitor these large scale and dynamic distributed networked systems adaptively and efficiently.

Mobile agent is an autonomous and independent software program to satisfy the corresponding user's goal on behalf of the user while visiting various target nodes through a network [3]. This mobile agent technology has several advantages such as reduction of network traffic, overcoming of network delay, enabling asynchronous execution and enhancement of dynamic adaptability. Thanks to these desirable features, this technology is very widely used in distributed systems, especially for network management. In a network management system, each mobile agent is generally designed to move to one or more agent-executable nodes in a network, sense temporally and permanently other nodes and resources, and filter and deliver the received management information to the appropriate network management nodes [10].

The previous mobile agent-based monitoring mechanisms are classified as follows: centralized and hierarchical distributed monitoring mechanisms. Most of them are based on the centralized monitoring model and divided into two categories, single mobile agent-based and segment-based mechanisms. In the first [11], a single management station creates a mobile agent and allows the agent to sequentially visit the required nodes in a particular order. This mechanism is simple to implement, but causes the task completion time of a mobile agent to become too long in large-scale distributed systems because the number of visiting nodes significantly increases and the size of the agent may grow considerably. In particular, if the visiting nodes are interconnected through low-bandwidth links, the round-trip delay may extremely increase. Secondly, the segment-based mechanism [2] partitions a network into several sub-networks or domains, and creates and transfers a mobile agent to each domain respectively. Therefore, the collection and filtering of the

management information for monitored nodes can be performed in parallel per domain, which addresses the scalability problem of the first mechanism to a certain extent. However, in this mechanism, the single manager should execute all the monitoring function and may become the performance bottleneck of the entire system. In addition, if the agent migration network includes expensive low bandwidth links, it is very difficult to perform the procedure to obtain and filter the monitoring information in real-time.

To solve the scalability problem, mobile agent-based mechanisms using hierarchical monitoring structure [6, 7] were proposed. They allow a network to be partitioned into a set of domains organized hierarchically and deploy a new monitoring agent to each domain. In this hierarchy, a main manager is at the top-level (level 1) and delegates monitoring tasks with monitoring agents to the lower level domain managers. Each manager clones and dispatches a monitoring agent to the appropriate domain manager node considering load redistribution of monitoring tasks. In this case, each domain manager collects the management information from the lower-level managers and filters and delivers the processed information to its higher-level manager. The original hierarchical monitoring mechanisms were almost based on a static manager organization model. In other words, each network administrator configures a tree of network domains according to its initial monitoring policy and then the main manager at the root domain creates and migrates monitoring manager agents to other domains. However, if any dynamic changes in various aspects such as network traffic patterns, resource addition and deletion, network topology and so on occur, this mechanism cannot adapt to these changes and will degrade significantly the entire management performance. There were presented some adaptive mobile agent-based mechanisms [8] to address this important issue. In these mechanisms, if each domain manager at level i estimates the need for some additional monitoring capability at run-time, it creates and installs a new manager agent to an appropriate node at level i+1 or migrates to another node for keeping location optimality of its network monitoring.

However, failures of some domain managers even assuming the main manager can be reliable using replication-based fault-tolerance mechanisms are very critical to providing correct, real-time and efficient monitoring functionality in a large-scale mobile agent-based distributed monitoring system. To the best of our knowledge, the fault-tolerance mechanism proposed in [13] is the only one to address this issue. But, in this mechanism, every agent should periodically send heart-beat messages to global failure detection agents (GFDAs). If the GFDA receives no heart-beat message from an agent for a predefined number of consecutive timeout intervals, it generates and delivers an AgentFailure message to a global recovery agent (GRA). Afterwards, the GRA recreates a new agent based on its most recent configuration information and redeploys it to the appropriate target host. However, this behavior results in high failure-free overhead due to the centralization of failure detection functionality in a single point within a large-scale hierarchical monitoring organization. Additionally, the takeover procedure performed by GRAs is much unsuitable for maintaining a tree-like manager structure efficiently. Also, this mechanism includes no concrete method to detect failures of manager agents correctly in case of agent creation, migration and termination triggered by dynamic changes in a network. This paper proposes a novel fault-tolerance mechanism to have the following desirable features appropriate for large-scale and dynamic hierarchical mobile agent-based monitoring organizations:

•Support fast failure detection functionality with low failure-free overhead by each domain manager periodically transmitting heart-beat messages to its immediate higher-level manager.
•Minimize the number of non-faulty monitoring managers affected by failures of domain managers.
•Enable consistent failure detection actions to be performed continuously in case of agent creation, migration and termination.
•Can execute consistent takeover actions even in concurrent failures of domain managers.

The remainder of this paper is organized as follows. In sections 2 and 3, we describe our proposed mechanism in both conceptual and algorithmic ways, and show its correctness proof. Section 4 compares the proposed mechanism with the existing ones in detail and section 5 concludes this paper.

2. The Proposed Mechanism

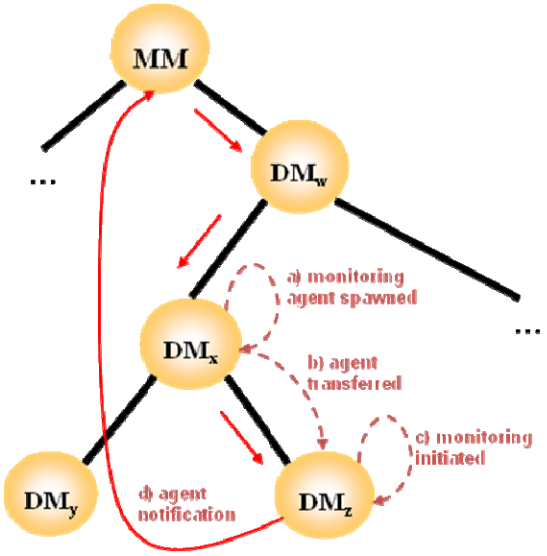
In the following subsections, data structures and algorithms of the proposed mechanism are described informally.

2.1 Data structures

Every domain monitoring manager α has to keep the following three variables.
•AID_α: it is the agent identifier of domain manager α.
•$MMaddr_\alpha$: it is the main manager's identifier needed when domain manager α is created or the organization of its lower-level managers changes.
•$IHMaddr_\alpha$: it is the immediate higher-level manager's identifier of domain manager α.

•ptr_α: it is the root node of a tree for saving the identifier and timer of every lower-level manager of main or domain monitoring manager α. Its node is a tuple (*aid*, *tinterval*, *ptr*). *tinterval* for each lower-level manager *aid* is used so that monitoring manager α detects whether its lower-level manager *aid* is alive or failed, and is initialized to τ . *ptr* for its lower-level manager *aid* is the next-level node maintaining references for all lower-level managers of the domain manager *aid* in a hierarchical manner.

2.2 Informal Description

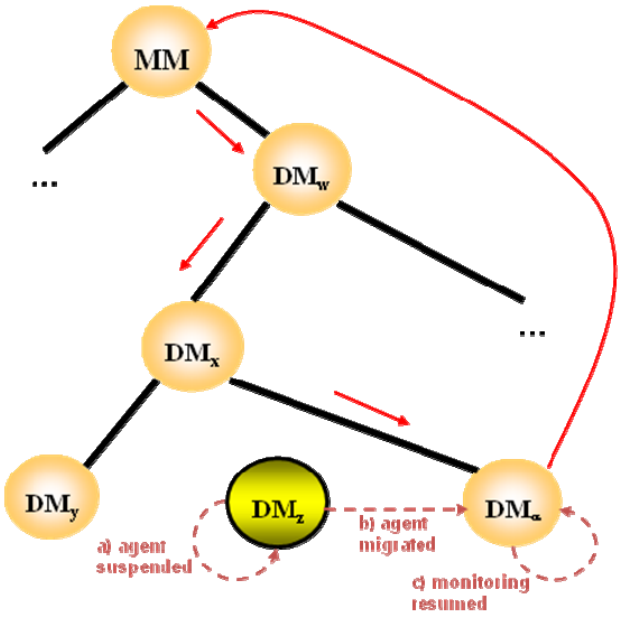


	MM	DM_w	DM_x	DM_y	DM_z
MMadr	id_{ser}	id_{MM}	id_{MM}	id_{MM}	id_{MM}
IHMadr	null	id_{MM}	id_w	id_x	id_x
ptr_{MM}	…. $(id_w.\tau.ptr_w)$				
ptr_w	$(id_x.\tau.ptr_x)$. …	$(id_x.\tau.ptr_x)$. …			
ptr_x	$(id_y.\tau.null)$. $(id_z.\tau.null)$	$(id_y.\tau.null)$. $(id_z.\tau.null)$	$(id_y.\tau.null)$. $(id_z.\tau.null)$		

Fig.1 In case of another DM being required.

Every domain manager α periodically transmits each heartbeat message only to its immediate higher-level manager $IHMaddr_\alpha$. Therefore, each monitoring manager can know which ones fail or are alive among its immediate lower level managers by their periodic notification. In our mechanism, the manager α decrements the timer *tinterval* for its corresponding immediate lower-level manager *aid* in ptr_α by one every certain time interval. If α has not received any heart-beat message from the lower-level manager until the timer expires, it suspects that the lower-level manager crashes. This behavior results in low failure-free overhead incurred by failure detection by utilizing the tree-like organization of monitoring managers effectively.

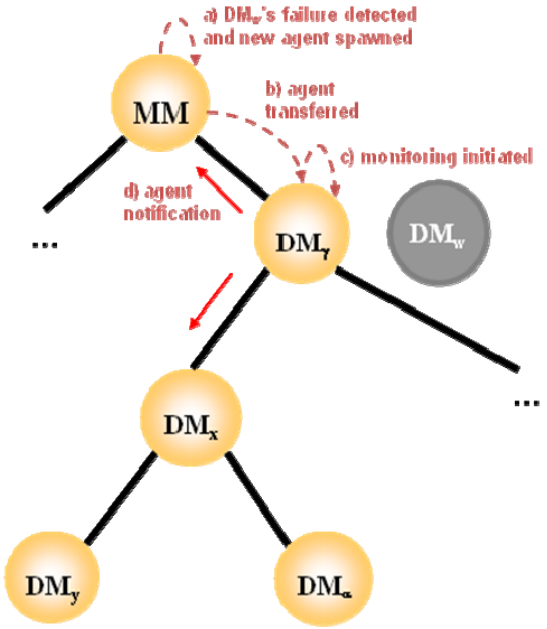
If a monitoring manager determines that a new one is needed as its immediate lower-level manager for effective monitoring, it creates a new mobile agent for this like figure 1. In this figure, manager DM_x has a monitoring agent spawned and transferred to a new node DM_z. The agent initiates its monitoring task and notifies of its location all nodes on the path between the main manager *MM* and itself. When a manager knows that it cannot play its role well and effectively for guaranteeing the monitoring performance required, it is voluntarily replaced by agent migration like in figure 2. In this figure, after manager DM_z has made the same decision mentioned above, it finds an appropriate substitute node DM_α and forces its agent to migrate to the substitute, where the agent resumes its monitoring task. If a manager detects some immediate lower-level managers has failed, it activates our takeover procedure.



	MM	DM_w	DM_x	DM_y	DM_α
MMadr	id_{ser}	id_{MM}	id_{MM}	id_{MM}	id_{MM}
IHMadr	null	id_{MM}	id_w	id_x	id_x
ptr_{MM}	…. $(id_w.\tau.ptr_w)$				
ptr_w	$(id_x.\tau.ptr_x)$. …	$(id_x.\tau.ptr_x)$. …			
ptr_x	$(id_y.\tau.null)$. $(id_\alpha.\tau.null)$	$(id_y.\tau.null)$. $(id_\alpha.\tau.null)$	$(id_y.\tau.null)$. $(id_\alpha.\tau.null)$		

Fig.2 In case of DM replacement by agent migration.

At this point, there can occur among three cases depending on availability and capability of nodes. First, like in figure 3, when *MM* recognizes DM_w's failure and a new node DM_γ is its suitable substitute, the main manager creates and transfers a new monitoring agent with the same role to node DM_γ. Then, it performs the same monitoring function the failed node DM_w executed, and inform its immediate lower-level managers, e.g., DM_x of this replacement. Second, when a manager DM_γ identifies the failure of its next-level manager DM_x in figure 3 and there is no available node for replacing the failed one, it checks whether among DM_x's immediate lower-level managers DM_y and DM_α, there exists a proper one as DM_x's replacement.

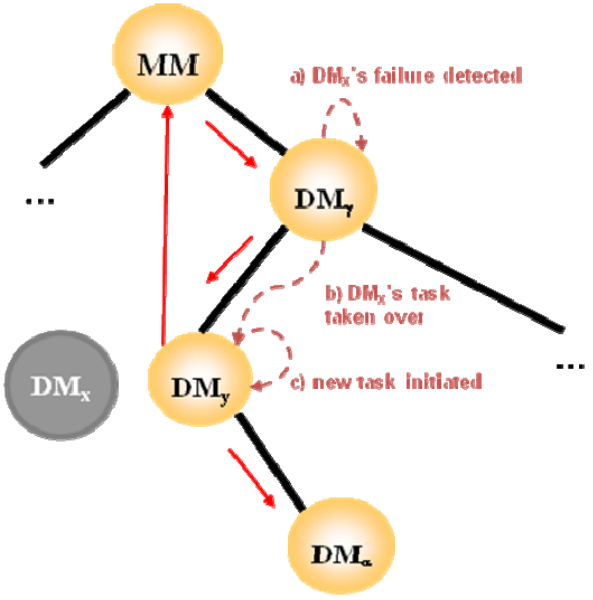


	MM	DM_γ	DM_x	DM_y	DM_α
MMadr	id_{self}	id_{MM}	id_{MM}	id_{MM}	id_{MM}
IHMadr	null	id_{MM}	id_γ	id_x	id_x
ptr_{MM}	.... $(id_\gamma, \tau, ptr_\gamma)$				
ptr_γ	(id_x, τ, ptr_x), ...	(id_x, τ, ptr_x), ...			
ptr_x	$(id_y, \tau, null)$, $(id_\alpha, \tau, null)$	$(id_y, \tau, null)$, $(id_\alpha, \tau, null)$	$(id_y, \tau, null)$, $(id_\alpha, \tau, null)$		

Fig.3 In case of a new DM taking over failed DM's task.

If DM_γ determines that DM_y is just suitable for the role, it allows DM_y to take over DM_x's task like in figure 4. In this case, DM_y notifies DM_x's other immediate lower-level managers of this substitution and updates its location on all nodes on the path between the main manager *MM* and DM_x. As the last case, when there is neither any new nor lower-level manager capable of being substituted for the failed one DM_x in figure 3, DM_x's immediate higher-level manager DM_γ takes over DM_x's role aside from DM_γ's own task in figure 5. Also, the mechanism performs the consistent takeover procedure even in case of concurrent failures of domain managers.

Algorithmic description of the failure detection and takeover procedures for main or domain manager *Self* in our mechanism are formally given in figures 6 and 7.



	MM	DM_γ	DM_y	DM_α
MMadr	id_{self}	id_{MM}	id_{MM}	id_{MM}
IHMadr	null	id_{MM}	id_γ	id_y
ptr_{MM}	.... (id_y, τ, ptr_y)			
ptr_y	$(id_\gamma, \tau, ptr_\gamma)$, ...	$(id_\gamma, \tau, ptr_\gamma)$, ...		
ptr_γ	$(id_\alpha, \tau, null)$	$(id_\alpha, \tau, null)$	$(id_\alpha, \tau, null)$	

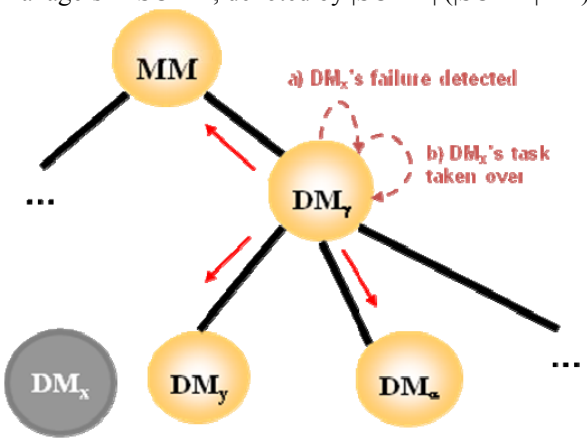
Fig.4 In case of an existing DM taking over failed DMs task.

3. Correctness Proof

This section shows theorems 1 and 2 to prove safety and liveness of our proposed mechanism in order.

Theorem 1. Even if multiple domain managers crash concurrently, our mechanism enables other live managers to monitor all the network elements previously managed by the failed ones.

Proof: Suppose that the entire distributed monitoring system consists of a finite set N of monitoring managers whose size is n and there is the set of all crashed domain managers, denoted by SCDM. The proof proceeds by induction on the number of all the crashed domain managers in SCDM, denoted by |SCDM| (|SCDM| < n).



	MM	DM_y	DM_y	DM_α
MMadr	id_{self}	id_{MM}	id_{MM}	id_{MM}
IHMadr	null	id_{MM}	id_y	id_y
ptr_{MM}	.... (id_y, τ, ptr_y)			
ptr_y	$(id_y, \tau, null)$. $(id_\alpha, \tau, null)$ ...	$(id_y, \tau, null)$. $(id_\alpha, \tau, null)$ ...		

Fig.5 In case of the immediate higher-level DM taking over failed DMs.

```
procedure NOTIF AGENTALIVEMSG()
   send a message Update_AgentTInterval(AID_Self)
   to IHMaddr_Self ;

procedure UPDATE AGENTTINTERVAL(AID)
   for all e in ptr_Self do
      if(e.aid = AID) then e.tinterval ← τ ;
   return ;

procedure MIGRATE AGENTTONEWNODE(nmngr)
   invoke MNGR TAKEOVER(AID_Self) on nmngr ;
   send a message Change_IHigherLevelMngr(IHMaddr_Self)
   to nmngr ;
```

Fig.6 Failure detection and takeover procedures for manager *Self*.

[Base case] As |SCDM|=1, there is only one crashed domain manager DM_x. In this case, the following three cases should be considered.

Case 1: there is a new available domain manager DM_y capable of taking over DM_x.

In this case, after detecting DM_x's failure, the immediate higher level manager of DM_x creates and transfers a new monitoring agent with the same role to node DM_y. Then, DM_y performs the same monitoring function the failed node DM_x executed, and inform DM_x's immediate lower-level managers of this replacement and updates its location on all nodes on the path between the main manager and DM_x.

```
procedure CHECK_AGENTLIVENESS()
   failedMngrs ← invoke DECR_TINTERVAL() on Self ;
   for all fmngr in failedMngrs do
      if(there is a new node nmngr as an appropriate substitute
      for fmngr) then
         invoke MNGR_TAKEOVER(fmngr) on nmngr ;
         send a message Change_IHigherLevelMngr(AID_Self)
         to nmngr ;
      else if(there is a suitable substitute lmngr for fmngr
      among its immediate lower-level managers) then
         invoke MNGR_TAKEOVER(fmngr) on lmngr ;
         send a message Change_IHigherLevelMngr(AID_Self)
         to lmngr ;
      else invoke MNGR_TAKEOVER(fmngr) on Self ;

procedure DECR_TINTERVAL()
   failedMngrs ← Φ ;
   for all e in ptr_Self do
      e.tinterval ← e.tinterval - 1 ;
      if(e.tinterval = 0) then
         failedMngrs ← failedMngrs U {e} ;
   ptr_Self ← ptr_Self - failedMngrs ;
   return failedMngrs ;

procedure MNGR_TAKEOVER(fmngr)
   for all e in fmngr.ptr do
      ptr_Self ← ptr_Self U {(e.aid, e.ptr, τ )} ;
      send a message Change_IHigherLevelMngr(AID_Self)
      to e.aid ;
   send a message Change_TreeTopologyAtMMngr(AID_Self,
    ptr_Self ) to MMaddr_Self ;

procedure CHANGE_TREETOPOLOGYATMMNGR(AID,
ptr)
   find a path mngrs to AID in ptr_Self ;
   find a node e in ptr_Self st (e.aid = AID);   e.ptr ← ptr ;
   NLMngr ← the first element e in mngrs ;
   mngrs ← mngrs - {e} ;
   send a message Change_TreeTopology(mngrs, AID, ptr)
   to NLMngr ;

procedure CHANGE_TREETOPOLOGY(mngrs, AID, ptr)
   find a node e in ptr_Self st (e.aid = AID) ;
   e.ptr ← ptr ;
   if(mngrs = Φ) then
      NLMngr ← the first element e in mngrs ;
      mngrs ← mngrs - {e} ;
      send a message Change_TreeTopology(mngrs, AID, ptr)
      to NLMngr ;

procedure CHANGE_IHIGHERLEVELMNGR(AID)
   IHMaddr_Self ← AID ;
```

Fig.7 Failure detection and takeover procedures for *Self* (continued).

Case 2: among DM_x's immediate lower-level managers, there is a proper one DM_y as DM_x's substitute.
In this case, DM_x's immediate higher level manager allows DM_y to take over DM_x's task and notifies DM_x's other immediate lower-level managers of this substitution and updates its location on all nodes on the path between the main manager and DM_x.
Case 3: there is neither any new nor lower-level manager capable of being substituted for DM_x. In this case, DM_x's immediate higher-level manager DM_y takes over DM_x's role aside from DM_y's own task. The subsequent procedure is the same as in case 2.
[Induction hypothesis] We assume that the theorem is true in case that |SCDM|=k.
[Induction step] Only if (k+1)-th crashed domain manager (k+1 < n) can be taken over by any other live domain managers, the theorem is true in case that |SCDM|=k+1. The following case is the same as the base case mentioned above.
By induction, even after |SCDM| concurrent domain manager failures occur, our mechanism allows their monitoring functions to be taken over other surviving ones.

Theorem 2. Our mechanism terminates within a finite time.

Proof: As no more than |SCDM| (|SCDM| < n) domain manager crashes occur, the proposed mechanism has only to re-execute its takeover procedure at most up to |SCDM| times as explained in theorem 1. Thus, the mechanism terminates within a finite time.

4. Comparisons

Most of monitoring systems using mobile agents were developed based on flat network infra-structure. Single agent-based monitoring system proposed in [11] forces a single agent to be created on the network manager and to perform its task monitoring function according to the itinerary consisting of its target nodes. It is simple to implement, but not scalable because in large distributed networks, the round-trip delay for the agent may become significantly increasing, especially on polling frequently, and its size, considerably large while visiting its target nodes.
Corradi et al. [2] presented a segment-based monitoring mechanism partitioning a network into a set of sub-networks or domains and transferring a mobile agent to each domain. This mechanism can reduce greatly its overall monitoring response time by collecting and filtering its management data per domain in parallel compared with the single agent-based one [11].
Gavalas et al. [4] proposed a broadcast-based monitoring mechanism, where the network manager instantiates and migrates each a mobile agent to all managed nodes. After the agent collects and analyzes the network traffic information from the corresponding node, it returns to the network manager platform with the requested information. Thus, this mechanism maximizes the parallelism of its monitoring process and achieves its short response time. However, as the number of managed network elements or resources extremely increases, a large number of mobile agents are required. This feature may incur high agent movement overhead by broadcasting and so degrade the entire system performance remarkably.
All the mechanisms stated above may not overcome the limitation of scalability fundamentally because of their centralization nature. Also, this problem becomes getting increasingly worse if expensive and low bandwidth links are included in routing paths of agents.
Liotta et al. [7] introduced a scalable multi-level monitoring mechanism based on the concept of Management by Delegation (MbM) [6]. In other words, this mechanism partitions a networked system into several domains composing a hierarchical structure and deploys a mobile agent to each of them.
A distributed java agent-based monitoring system JAMM was proposed for grid computing in [12]. This system enables monitoring sensors to execute by triggering their execution based on actual client usage. Clients can control remote sensors and obtain their requested information from the sensors in the form of events.
In [9], a multi-agent based distributed monitoring system is implemented composed of dynamically controllable agents. The structure of the system is divided into three layers to support independence among communication protocols, message interpretation and monitoring tasks. This independence among the three layers may reduce agent development time and make
it easy to manage distributed systems.
However, these three mechanisms [7, 9, 12] cannot be autonomously adaptable for dynamic changes such as variations of network traffic patterns, resource addition and deletion, changes of network topology and so on because their structure of monitoring managers is static after the initial agent deployment.
In [8], an adaptive and hierarchical mobile agent-based monitoring mechanism was presented to address the above mentioned problems. In this mechanism, each middle-level manager agent is not bound to a particular network node and be able to sense the network, find and move to better locations for seeking monitoring location optimality.
But, among all the previously stated hierarchical mobile agent-based mechanisms, no one addresses the failure

detection and recovery issue for a hierarchy of monitoring managers.

Tripathi et al. [13] presented a mobile agent-based distributed monitoring system supporting autonomic configuration and recovery. In this system, there are several global failure detection agents subscribing to heart-beat events from all monitoring agents. Thus, every monitoring agent periodically sends its heart-beat message to each global failure detection agent. If a monitoring agent fails, one of global recovery agents in this system executes the following recovery procedure: the recovery agent instantiates the monitoring agent based on its most recent configuration information and re-launches it to an available node. Thus, if large-scale networks are assumed, this feature results in high failure-free overhead due to the centralized failure detection procedure. Additionally, the takeover procedure of the global recovery agents in this system is very unsuitable for maintaining a tree-like monitoring manager structure efficiently. Also, this system presented no concrete mechanism to have the hierarchical structure of monitoring managers adaptable for its correct and efficient failure detection in case of agent creation, migration and destruction caused by the dynamic changes within its entire network.

5. Conclusions

This paper presented a novel fault-tolerance mechanism to have the following advantageous features appropriate for large-scale and dynamic hierarchical mobile agent-based monitoring organizations. It supports fast failure detection functionality with low failure-free overhead by each domain manager transmitting heart-beat messages to its immediate higher-level manager. Also, it minimizes the number of non-faulty monitoring managers affected by failures of domain managers. Moreover, it allows consistent failure detection actions to be performed continuously in case of agent creation, migration and termination, and is able to execute consistent takeover actions even in concurrent failures of domain managers.

References

[1] H. Asgari, P. Trimintzios, M. Irons, G. Pavlou, S. Berghe, and R. Egan, "A Scalable Real-time Monitoring System for Supporting Traffic Engineering", in Proc. of the IEEE Workshop on IP Operations and Management, Dallas, USA, 2002.

[2] A. Corradi, C. Stefanelli, and F. Tarantino, "How to Employ Mobile Agents in Systems Management", in Proc. of the 3rd Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'98), 1998, pp. 17-26.

[3] A. Fuggetta, G.P.Picco, and G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, Vol. 24, No. 5, 1998, pp. 342-361.

[4] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony, "Complimentary Polling Modes for Network Performance Management Employing Mobile Agents", in Proc. of the IEEE Global Communications Conference (Globecom'99), 1999, pp. 401-405.

[5] D. Goderis, S. Bosch, and Y. T'Joens, "A Service-Centric IP Quality of Service Architecture for Next Generation Networks", in Proc. of the IEEE/IFIP Network Operations and Management Symposium, 2002, pp. 139-154.

[6] G. Goldszmidt, and Y. Yemini, "Delegated Agents for Network Management", IEEE Communication Magazine, Vol. 36, No. 3, 1998, pp. 66-70.

[7] A. Liotta , G. Knight, and G. Pavlou, "Modelling Network and System Monitoring Over the Internet with Mobile Agents", in Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS'98), 1998, pp. 303-312.

[8] A. Liotta , G. Pavlou, and G. Knight, "Exploiting Agent Mobility for Large-scale Network Monitoring", IEEE Network, 2002, pp. 7-15.

[9] S. Kwon, and J. Choi, "An Agent-based Adaptive Monitoring System", Lecture Notes In Artificial Intelligence, Vol. 4088, 2006, pp. 672-677.

[10] J. Philippe, M. Flatin, and S. Znaty, "Two Taxonomies of Distributed Network and System Management Paradigms", Emerging Trends and Challenges in Network Management, 2000.

[11] G. Susilo, A. Bieszczad, and B. Pagurek, "Infrastructure for Advanced Network Management based on Mobile Code", In Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS'98), 1998, pp. 322-333.

[12] B. Tierney, B. Crowley, D. Gunter, J. Lee, and M. Thompson, "A Monitoring Sensor Management System for Grid Environments", Cluster Computing Journal, Vol. 4, No. 1, 2001, pp. 19–28.

[13] A. Tripathi, D. Kulkarni, H. Talkad, M. Koka, S. Karanth, T. Ahmed, and I. Osipkov, "Autonomic Configuration and Recovery In A Mobile Agent-based Distributed Event Monitoring System", Software Practice and Experience, Vol. 37, 2007, pp. 493–522.

Jinho Ahn received his B.S., M.S. and Ph.D. degrees in Computer Science and Engineering from Korea University, Korea, in 1997, 1999 and 2003, respectively. He has been an associate professor in Department of Computer Science, Kyonggi University. He has published more than 70 papers in refereed journals and conference proceedings and served as program or organizing committee member or session chair in several domestic/international conferences and editor-in-chief of journal of Korean Institute of Information Technology and editorial board member of journal of Korean Society for Internet Information. His research interests include distributed computing, fault-tolerance, sensor networks and mobile agent systems.