CIC: An Integrated Approach to Checkpointing in Mobile Agent Systems

Jin Yang, Jiannong Cao, Weigang Wu
Internet and Mobile Computing Lab
Department of Computing, Hong Kong Polytechnic University
Hung Hom, Kowloon Hong Kong
{csyangj, csjcao, cswgwu}@comp.polyu.edu.hk

Abstract

As a widely used fault tolerance technique, checkpointing has evolved into several schemes: independent, coordinated, and communication-induced (CIC). Independent and coordinated checkpointing have been adopted in many works on fault tolerant mobile agent (MA) systems. However, CIC, a flexible, efficient, and scalable checkpointing scheme, has not been applied to MA systems. Based on the analysis of the behavior of mobile agent, we argue that CIC is a well suited checkpointing scheme for MA systems. CIC not only establishes the consistent recovery lines efficiently but also integrates well with the independent checkpointing for reliable MA migration. In this paper, we propose an important improvement to CIC, referred to as the deferred message processing based CIC algorithm (DM-CIC), which achieves higher efficiency by exempting the CIC algorithm from making the forced checkpoints in MA systems. Through simulation, we find out that DM-CIC is stable and better suited to large scale MA systems.

1. Introduction

A mobile agent (MA) is a program that can migrate from host to host in a network of heterogeneous computers to execute the tasks specified by its owner. Characteristics of MAs include mobility, autonomy, asynchrony, encapsulation of protocols, adaptability, and support for mobile computing [3]. These characteristics make MA a new computing model and have a great potential to be used for structuring and coordinating distributed applications, such as e-commerce, information searching, network management, and Grid computing [2, 10, 21]. Most of these applications require high degree of reliability and consistency. Therefore, fault tolerance is a key issue in designing an MA system.

Checkpointing is one of the widely used fault tolerance techniques. A checkpoint is the copy of a process' code and state stored on stable storage. Taking a checkpoint of a MA is readily facilitated in a MA system: serializing an MA for the migration to the next host effectively constructs a checkpoint. Nearly all existing MA platforms are Java based, so it is easy to utilize the serialization technique provided by Java to make checkpoints.

Checkpointing techniques have been classified into three major schemes, namely independent (or uncoordinated), coordinated, or communication-induced [7]. Among the three checkpointing schemes, independent checkpointing is the simplest one. It allows processes to take checkpoints periodically without any coordination with each other. Accordingly, it has been widely used in MA systems to guarantee MAs' reliable migration or to prevent possible agent crashes. Independent checkpointing, however, may suffer from the domino effect [17]. In a group of MAs (or processes) that communicate by messages, the messages can induce inter-MA dependencies during failure-free operation. Therefore, upon a failure of one or more MAs, these dependencies may force some of the MAs that did not fail to rollback, creating what is commonly called rollback propagation. Rollback propagation may extend back to the initial state of the computation, causing the loss of all the work performed before a failure.

In order to avoid the domino effect, coordinated checkpointing is proposed [5], in which processes coordinate to synchronize their checkpointing activities. In this way, a globally consistent set of checkpoints is always maintained in the system. However, coordinated checkpointing involves high message overhead, which makes it unsuitable for mobile/wireless computing systems with low bandwidth communication channels. A further disadvantage of coordinated checkpointing is that

Proceedings of the Second International
Conference on Semantics, Knowledge, and Grid (SKG'06)
0-7695-2673-X/06 $20.00 © 2006 IEEE

process execution may have to be suspended during coordination, which results in the performance degradation. Another checkpointing scheme to over-come the domino effect is communication-induced checkpointing (CIC) [18]. In CIC, processes make two kinds of checkpoints, basic and forced. Basic checkpoint is a kind of independent checkpoint, so no coordination is needed. Forced checkpoint is made to maintain the consistent recovery line. Instead of exchanging coordination messages like in coordinated checkpointing, CIC piggybacks protocol specific information in application messages. Processes use the piggybacked information to decide whether a forced checkpoint should be taken or not.

Compare with independent and coordinated checkpointing, CIC has several advantages. First, it is intuitively believed to be scalable since it does not require the processes to participate in taking a global checkpoint. Second, CIC allows processes themselves to make decision to take checkpoints, so the processes can choose the right time to make the checkpoints. Obviously, this characteristic provides the autonomy and flexibility, which especially match the characteristics of MA system. In addition, CIC can also be integrated with other checkpointing schemes in a MA system.

The rest of the paper is organized as follows. Section 2 briefly describes previous works. Section 3 defines our system model. In Section 4, the basic CIC based checkpointing algorithm for MA systems and its integration with reliable migration are described. Considering the characteristics of a MA system, we propose an important improvement to CIC, called the deferred message processing based CIC algorithm (DM-CIC), which achieves higher efficiency by exempting the CIC algorithm from making the forced checkpoints in MA systems. Results of performance evaluation through simulations are presented in Section 5. We conclude this paper in Section 6.

2. Related works

Existing works on checkpointing schemes for MA system have adopted two checkpointing schemes: independent checkpointing and coordinated check-pointing. Due to the domino effect, independent checkpointing is normally applied to single MA scenarios, while coordinated checkpointing is used to provide checkpointing for a group of MAs.

Independent checkpointing is usually used to guarantee the persistence of state associated with MAs. There are two main usages. The first is to guarantee reliable migration by checkpointing the agent to stable storage before dispatching it to a new host. This copy is kept until the agent arrives at the new host. The second usage is to tolerate agent crash on a host by checkpointing a replica MA into stable storage upon the agent's arrival at each host or during its execution on the host. These two approaches have been adopted in many works [4,9,12,15] as well as in some MA systems. Concordia [19] utilizes proxy objects and a persistent object store to insulate applications from system or network failures. However, the task of checkpointing and recovery of MAs is left to the programmer. Ara [16] offers a similar means for an MA system to create a checkpoint, which is stored on some persistent media (e.g. disk).

Domino effect greatly constrains the deployment of independent checkpointing in the environments where an MA has interactions with others, i.e., the multiple MAs cooperation applications. In order to remove the domino effect, message logging has been used in tandem with independent checkpointing. The authors of [14] used receiver based logging to log messages so as to ensure that the messages could be regenerated during the re-execution phase. The advantage claimed for receiver based logging against sender based logging is faster recovery. Another claimed advantage is that recovery and pruning of the message log can be done autonomously, without interaction with other user agents. But how to implement the recovery was not described. In [11], authors adopted the similar idea. The focus of these two works is on the replication scheme, with checkpointing being of an assistant role. Another independent checkpointing strategy assisted by message logging is found in [13], although the authors call it a communication pairs independent checkpointing strategy.

Another way to restrict the influence of failures on other agents in the same communication group is to use coordinated checkpointing or CIC. In coordinated checkpointing, a coordinator initiates all the group members to start the checkpointing process. This coordinates message passing so as to produce a consistent system snapshot. In [6], a checkpoint manager (CM) monitors all the agents inside a cluster of machines. The CM, which is assumed to be very reliable, is responsible for keeping track of the agents and for restarting the agents when there is a node failure. The main problem with this approach is the fact that the CM is a single point of failure, and the message cost is such a centralized way could also be very high. In [8], authors present a coordinated checkpointing procedure in which one particular checkpoint server acts as a checkpoint coordinator. Same as [6], it suffers from the problem of single point failure of the checkpoint server.

Proceedings of the Second International
Conference on Semantics, Knowledge, and Grid (SKG'06)
0-7695-2673-X/06 $20.00 © 2006 IEEE

3. System model

We consider an MA system consisting of a group of cooperating MAs which form an MA group. In this MA group, each group member (a single MA) has a globally unique group ID and MA ID. Each group member executes and migrates along a predefined or self-initiated itinerary. With a predefined itinerary, the agent knows all the hosts that it will visit, while with a self-initiated itinerary, the agent only knows the first host it will visit and the following hosts are determined by the execution results on the previous host. The itinerary consists of N hosts. Group members communicate by message passing. Messages can be delayed, replicated or lost. Group members may crash during its execution and migration. The host and the MA platform (MAP) may also crash.

To guarantee MA's reliable migration or prevent the possible agent crash, we assume that each MA can take an independent checkpoint before its migration. After the agent finishes its execution on a host, it will migrate to the next host according to the itinerary. This process will continue until all the hosts have been visited.

Errors in message transmission will be handled by the transport layer of an MA system. For simplicity, we assume that all the operations executed by the MAs are idempotent, so the exactly once execution property needs not to be considered in this paper. For non-idempotent operations, MA transaction support is needed to maintain the system consistency during recovery [20].

4. CIC based Algorithms for MA Systems

As we mentioned, CIC has been proven to be a flexible and efficient checkpointing scheme to shield the domino effect, while allows each process to decide when to make checkpoint by itself. This motivates us to introduce CIC for the execution of a group of MAs. In [1], a typical index-based CIC algorithm has been proposed. The algorithm assumes that each process p_i maintains a logical clock lc_i which functions as p_i's checkpoint timestamp. The timestamp is an integer variable with initial value 0 and is incremented as followings:

1. lc_i increases by 1 whenever p_i takes a basic checkpoint.
2. On every message m it sends, p_i piggybacks a copy of the current value of lc_i. We denote the piggybacked value as $m.lc$.
3. Whenever p_i receives a message m, it compares lc_i with $m.lc$. If $m.lc > lc_i$, then p_i sets lc_i to the value of $m.lc$ and takes a forced checkpoint before processing the message.

The set of checkpoints having the same timestamps in different processes is guaranteed to be a consistent state. We adopt this traditional CIC algorithm and integrate it with the existing checkpointing algorithms in MA systems, such as independent checkpointing used to guarantee the reliable migration for MAs. We also improve the proposed CIC algorithm by adopting the deferred message processing. The new algorithm, called DM-CIC, can avoid the forced checkpoint so as to get a better trade-off on the system performance.

4.1 Basic-CIC Algorithm for MA Systems

Algorithm 1: *Basic-CIC*

1. Each MA_i maintains a logical clock: C_i. C_i is initiated to be *0*.

2. Basic checkpoint is taken by MA platform (MAP_i) according to a predefined frequency (defined in each MA) and transparent to MA. Before taking a basic checkpoint, C_i is increased by 1 and the basic checkpoint is tagged with C_i.

3. The value of C_i is piggybacked in every message M_i sent out by MA_i. We denote the piggybacked C_i as MC_i.

4. Suppose a message M_i is sent to MA_j. When MA_j receives a message M_i, it compares C_j with MC_i. If $MC_i > C_j$, then sets C_j to be the value of MC_i and takes a forced checkpoint tagged with C_j before processing M_i.

The Basic-CIC algorithm, as described in the above box of Algorithm 1, directly applies the traditional CIC algorithm to MA systems except that the MA platform will assist to make the checkpoints. Basic-CIC algorithm is.

Failure detector (FD) helps detect the failures of MAs. If an MA failed, it will be detected by the FD installed in the MA systems. Then, the FD will trigger the recovery process which will inform all the MAs in this group to rollback to the set of checkpoints having the same timestamps.

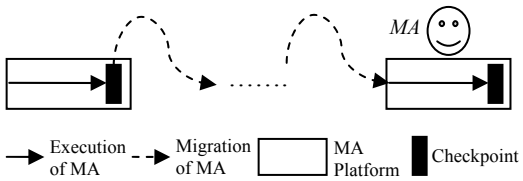

Figure 1 Reliable Migration of MA

According to Step 2 of the Basic-CIC algorithm, the basic checkpoint is taken by the MA platform according to a predefined frequency. As we mentioned,

Proceedings of the Second International
Conference on Semantics, Knowledge, and Grid (SKG'06)
0-7695-2673-X/06 $20.00 © 2006 IEEE

an advantage of CIC is that it can prevent the domino effect while allowing processes considerable autonomy in deciding when to take basic checkpoints. An efficient implementation of CIC must therefore adopt a checkpointing policy that exploits this autonomy and translates it into a benefit. In general, a good understanding of the application and of the execution environment is required. Specifically, for MA systems, we must exploit their characteristics to find out how to benefit from the autonomy of CIC.

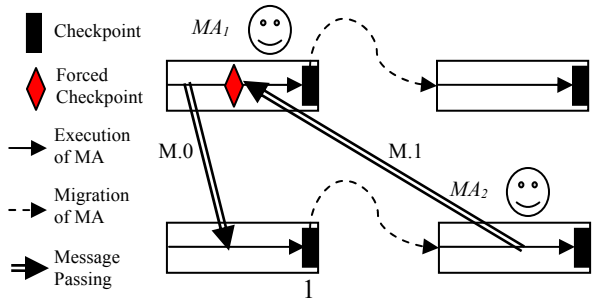

Figure 2 Basic-CIC Integrated with Reliable Migration

Figure 1 shows the scenario of the reliable migration algorithm with the use of independent checkpointing. If we can integrate the Basic-CIC algorithm with the reliable migration algorithm to let the independent checkpoints act as the basic checkpoints in Basic-CIC, we can save the operations to make basic checkpoints for Basic-CIC, so as to make the whole MA system more efficient. Figure 2 illustrates the integration of these two algorithms, where the checkpoints for reliable migration act as the basic checkpoints in Basic-CIC.

It is easy to implement such integration. We only need to modify the Step 2 of Algorithm 1 and let each MA platform (MAP) take basic checkpoints according to the frequency defined by the algorithm of reliable migration. Through this integration, an independent algorithm for reliable migration is not necessary. Basic-CIC will take the role to guarantee the reliable migration for each MA.

Up to now, we still have one problem. Although we let the checkpoints for reliable migration act as the basic checkpoints in the Basic-CIC algorithm, we still need to take the possible force checkpoints during an MA's execution on a host. As shown in Figure 2, an MA may receive a message during its execution on a host, which will result in a forced checkpoint being taken and the MA's execution interrupted temporarily. Such interruption will affect the performance of the MA application severely if an MA will travel a large number of hosts. At next section, we try to improve the Basic-CIC algorithm by making further exploiting the characteristic of MA systems.

4.2 Deferred Message Processing based CIC (DM-CIC)

In an MA system, due to the mobility of MAs, messages are delivered in a purely asynchronous manner. A typical message delivery scheme is the forwarding pointer. Each MA leaves a pointer on each MAP it visited to point to the next MAP that it migrates to. A message destined to this MA will first be sent to the home of this MA, and then follows the points to chase the MA. Depending on the execution speed of the MA and the networks' QoS, the message can catch up the MA sooner or later. For the MA, usually it is not important to receive the message earlier or later, on current host or on the next host. Considering this characteristic, we can avoid the forced checkpoint by deferring the processing of the message until a basic checkpoint is taken.

Figure 3 illustrates the scenario of the deferred message processing and Algorithm 2 shows the procedures of DM-CIC. When MA_1 receives a message *M.1* and, according to CIC, a forced checkpoint should be made, instead of taking a forced checkpoint, MA_1 only accepts message *M.1* but does not process it. The received message that can result in a forced checkpoint is stored in the MA's associated mailbox (an MA's associated mailbox is a buffer in an MA). The processing of the received message is delayed until MA_1 makes a basic checkpoint and lands on a new host. In this way, we merge the operations of taking the basic checkpoint and the force checkpoint, but the price is that the messages which can result in a forced checkpoint can not be processed immediately when they reach the MA.

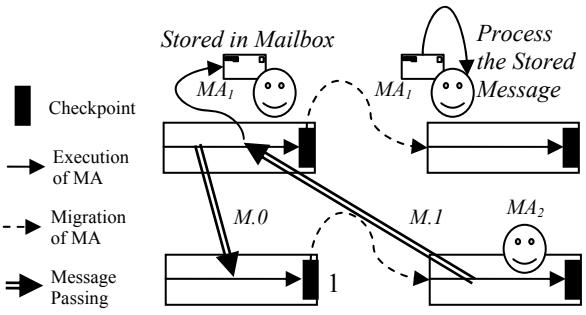

Figure 3 DM-CIC

Proceedings of the Second International
Conference on Semantics, Knowledge, and Grid (SKG'06)
0-7695-2673-X/06 $20.00 © 2006 IEEE

Algorithm 2: *DM-CIC*

1. Each MA_i maintains a logical clock C_i and a mailbox $Mbox_i$. C_i is initiated with 0 and $Mbox_i$ is initiated to be empty.

2. Basic checkpoint is made by MA platform (MAP_i) when the MA is ready to migrate to the next host. The frequency will be set according to the requirement of reliable migration. Before a basic checkpoint is made, MA's associated mailbox will be checked. There are two conditions:

   a) If the mailbox is empty, C_i is increased by 1 and the basic checkpoint is tagged with C_i.

   b) If the mailbox is not empty, C_i is set to the maximum logical clock of the stored messages in the mailbox, and then the basic checkpoint is tagged with C_i.

3. The value of C_i is piggybacked in every message M_i sent out by MA_i. We denote the piggybacked C_i as MC_i.

4. Suppose a message M_i is sent to MA_j. When MA_j receives the message M_i, it compares C_j with MC_i.
   a) If $MC_i \leqslant C_j$, then message M_i will be processed.
   b) If $MC_i > C_j$, M_i will not be processed but to be stored in MA_j's associated mailbox $Mbox_j$.

5. When an MA lands on a new host, it will check its mailbox and process the messages if the mailbox is not empty.

Even if an MA system does not adopt the independent checkpointing based reliable migration algorithm, DM-CIC can still be beneficial. Since an MA must be serialized before its migration, which effectively constructs a checkpoint, the basic checkpoint and forced checkpoint can be made at this point, so as to improve the system's performance.

However, the advantage is achieved on the price of sacrificing the messages' real-time processing. On the other hand, exempting from the force checkpoint saves the system's execution time. In the next section, we evaluate the performance of DM-CIC and Basic-CIC thought simulation.

5. Performance Evaluation

In our simulation, we consider the behavior of an MA which belongs to a group of MAs. The group of MAs communicates by message passing. Two metrics are adopted to evaluate the performance of Basic-CIC and DM-CIC: the delayed message processing and the whole execution time of an MA. We decompose an MA's execution into three procedures: task execution (T_{exe}=50ms), checkpointing (T_{cp}=10ms), and migration (T_{mg}=10ms). An MA can process messages during T_{exe}, while the message processing will be blocked during T_{cp} and T_{mg}. Since only the messages causing the forced checkpoint can impact the performance of the CIC algorithms, we only consider such messages in our simulations. We assume these messages are sent and received randomly (following the exponential distribution) during the running of MAs' execution.

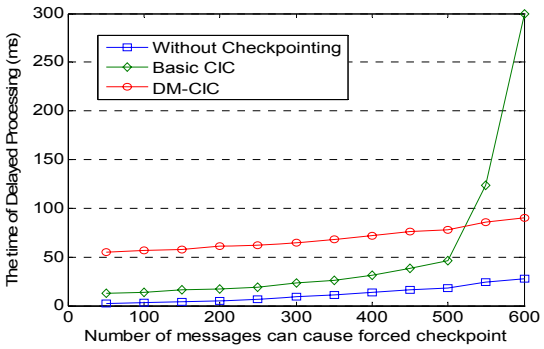

Figure 4 The delayed message processing

Figure 4 illustrates the time of the delayed message processing. In DM-CIC, all the messages' processing will be delayed, but the delay is stable. Basic-CIC performs well when few messages arrive. But its performance degrades sharply when more messages come in, because the large number of operations for forced checkpointing will block the message processing and increase the delay dramatically.

The operations of taking forced checkpoints also prolong the MA's whole execution time, as shown in Figure 5.

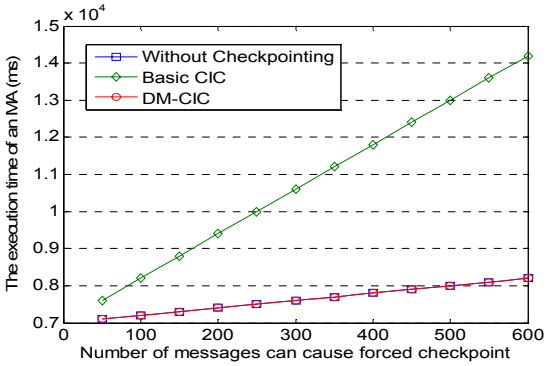

Figure 5 The whole execution time of an MA

6. Conclusions

In this paper, we proposed to apply the communication induced checkpointing (CIC) scheme for MA systems. Basic-CIC and DM-CIC algorithms have been developed to support the recovery of a group of MAs. CIC-based algorithms not only establish the consistent recovery lines efficiently but can also well integrate with the existing independent checkpointing operations used for reliable MA migration in MA systems. Through simulation, we find

that DM-CIC is better suited for large scale MA systems which may involve large amount of message exchange.

Acknowledgement

This work is supported in part by the University Grant Council of Hong Kong under the CERG Grants PolyU 5183/04E, and China National 973 Program under Grant 2002CB312002.

References

[1] D. Briatico, A. Ciuoletti, and L. Simoncini, “A Distributed Domino-Effect Free Recovery Algorithm”, In Proceedings of the IEEE International Symposium on Reliability Distributed Software and Database, pages 207~215, December 1984.

[2] J. Cao, G.H. Chan, W. Jia, and T. Dillon, "Checkpointing and Rollback of Wide-Area Distributed Applications Using Mobile Agents", Proc. IPDPS2001 - IEEE 2001 International Parallel and Distributed Processing, April 2001, San Francisco, USA.

[3] D. Chess, C. harrison, and A. Kershenbaum, “Mobile Agents: Are They a Good Idea?”, IBM Research Report, RC 19887 (#88465).

[4] X. Chen and M.R. Lyu. “Performance and Effectiveness Analysis of Checkpointing in Mobile Environments”, Proc. of the Int’l Symp. On Reliable Distributed Systems, pp. 131-140, 2003.

[5] Chandy, M., Lamport, L. “Distributed snapshots: Determining global states of distributed systems”, ACM Trans. Comput. Syst. 31, 1, 63–75. 1985.

[6] M. Dalmeijer, E. Rietjens, D. Hammer, A. Aerts, M. Schoede, “A reliable mobile agents architecture,” in Proc. 1st Int. Symp. Object-Oriented Real-Time Computing, Kyoto, Japan, Apr. 1998.

[7] E. N. Elnozahy, D. B. Johnson, Y. M. Wang. “A Survey of Rollback-Recovery Protocols in Message Passing Systems”, ACM Computer Surveys, Volume 34, Number 3, 2002 pp. 375-408

[8] E. Gendelman, L.F. Bic, and M.B. Dillencourt. “An Application-Transparent, Platform-Independent Approach to Rollback-recovery for Mobile Agent Systems”, Proc. 20th Int’l Conf. on Distributed Computing Syst., 2000.

[9] D. Johansen, K. Marzullo, F.B. Schneider, K. Jacobsen, and D. Zagorodnov, “NAP: Practical Fault-Tolerance for Itinerant Computations,” Proc. 19th Int’l Conf. Distributed Computing Systems (ICDCS ’99), June 1999.

[10] D.B. Lange and M. Oshima, “Seven Good Reasons for Mobile Agents”, Communication of the ACM, Vol. 42, No. 3, March 1999. pp. 88-89.

[11] M.R. Lyu and T.Y. Wong, "A Progressive Fault Tolerant Mechanism in Mobile Agent Systems," in Proceedings 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI2003), Orlando , Florida, July 27-30 2003, pp. 299-306.

[12] A. Mohindra, A. Purakayastha, and P. Thati, “Exploiting nondeterminism for reliability of mobile agent systems,” in Proc. Int. Conf. Dependable Systems Networks, Los Alamitos, CA, 2000.

[13] Osman, T.; Wagealla, W.; Bargiela, A.; "An approach to rollback recovery of collaborating mobile agents", IEEE Transactions on Systems, Man and Cybernetics, Part C, Volume 34, Issue 1, Feb. 2004 Page(s):48 - 57

[14] Holger Pals, Stefan Petri, and Claus Grewe, “FANTOMAS: Fault Tolerance for Mobile Agents in Clusters”, Parallel and Distributed Processing - Proceedings of 15 IPDPS 2000 Workshops, Cancun, Mexiko, May 2000, Cancun, Mexiko.

[15] Taesoon Park; Ilsoo Byun; Hyunjoo Kim; Yeom, H.Y.; “The performance of checkpointing and replication schemes for fault tolerant mobile agent systems” 2002. Proceedings. 21st IEEE Symposium on Reliable Distributed Systems, 13-16 Oct. 2002

[16] H. Peine and T. Stolpmann, “The Architecture of the Ara Platform for Mobile Agents”, Proc. 1st Int’l Workshop on Mobile Agents, 1997, Berlin, Germany.

[17] Randell, B. “System structure for software fault tolerance”, IEEE Trans. Softw. Engin. 1, 2, 220–232. 1975.

[18] Russell, D. L. “State restoration in systems of communicating processes”, IEEE Trans. Softw. Engin. 6, 2, 183–194. 1980.

[19] D. Wang, et al, “Concordia: An infrastructure for collaborating mobile agents”, Proc. 1st Int’l Workshop on Mobile Agents, Lecture Notes in Computer Science, Vol. 1219, 1997. pp.86-97.

[20] Jin Yang, Jiannong Cao, Weigang Wu, Chengzhong Xu, “A Framework for Transactional Mobile Agent Execution”, Proc. Int. Conf. on Grid and Cooperative Computing (GCC’05) 1002~1008 November 30-December 3, 2005, Beijing, China.

[21] H.Zhuge, et al, A Scalable P2P Platform for the Knowledge Grid, IEEE Transactions on Knowledge and Data Engineering, 17 (12) (2005):1721-1736