International Journal of Information Technology and Knowledge Management
January June 2009, Volume 2, No. 1, pp. 123-129

EFFECTIVE MECHANISMS CONSIDERING LOAD IMBALANCE OF SERVICE REPLICATION FOR DISTRIBUTED MOBILE AGENT SYSTEMS

Jinho Ahn*

To enhance scalability of replicated services a large number of mobile agents attempt to access in mobile agent systems, we present a new strategy to apply an appropriate passive replication mechanism to each replicated service according to its execution behavior because deterministic services require weaker constraints to ensure their consistency than non-deterministic ones. For this goal, two passive replication mechanisms are proposed for non-deterministic services and for deterministic services respectively. They both allow visiting mobile agents to be forwarded to and execute on any node performing a service agent, not necessarily the primary agent. Especially, in case of the mechanism for deterministic services, after a backup service agent has received a mobile agent request and obtained the delivery sequence number of the request from the primary service agent, the backup agent, not the primary one, is responsible for processing the request and coordinating with the other replica service agents.

Keywords: Distributed systems, mobile agent, service replication, scalability, reliability, load balancing

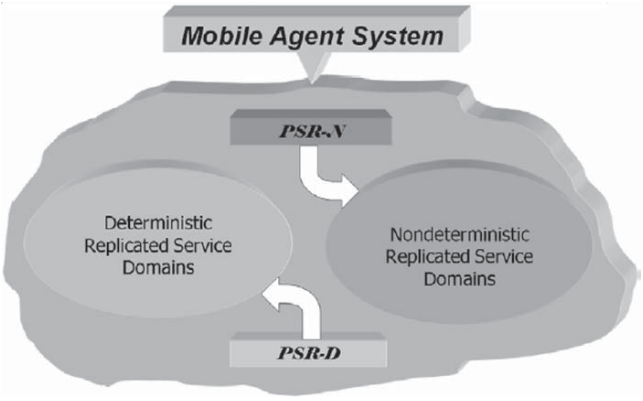
1. Introduction

Mobile agent paradigm is considered as a promising vehicle for developing distributed computing systems such as RFID-based systems, ubiquitous sensor networks, network monitoring and grid and autonomic computing because it provides a number of advantages such as reduction of network traffic and asynchronous interaction and so on unlike the traditional client server paradigm [5,6,9,15,16]. As the mobile agent system is gaining popularity and the number of mobile agent's users rapidly increases, a large number of mobile agents may concurrently be transferred to a node supporting a particular service. In this case, the service agent on the node can be a performance bottleneck and if the agent fails, the execution of all the transferred mobile agents be blocked. In order to solve these problems, the service agent function should be replicated at multiple nodes. This approach may balance the load caused by the mobile agents and even if some service agents fail, continuously allow the other service agents to provide the mobile agents with the service. There are two approaches used in distributed systems to potentially be applied for satisfying the goal: active and passive replication[17]. If the active replication approach[14] is used in the mobile agent system, every non-faulty replicated service agent receives the requests from mobile agents in the same order, processes and replies them to the agents. This approach requires the operations on the service agent to be deterministic: given the initial state of each service agent and the same sequence of mobile agent requests previously performed by the service agent, it will produce the same result[17]. It provides failure transparency for mobile agents because even if some replicated service agents fail, the other normal ones can handle the requests and send their replies to the mobile agents. Additionally, it requires low response time in case of failures compared with the passive approach. However, the approach has two important drawbacks: high resource consumption because all replicated service agents have to process mobile agent service requests and determinism constraint as mentioned above. In the passive replication approach[3,8], only one among a set of service agents, named primary service agent, receives a request from each mobile agent, processes it and then forwards a message including its updated state to the other service agents, called backup service agents. When receiving the message, each backup agent updates its state using the message and returns an acknowledgement message to the primary agent. After the primary agent receives the messages from all non-faulty backup agents, it sends a response of the request to the mobile agent. When the primary agent fails, this approach suffers from a high reconfiguration cost and cannot provide failure transparency for mobile agents. However, the passive replication approach has three desirable features we focus on. First, the approach enables its consistency to be guaranteed even if replicated service agents are performed in a non-deterministic manner. Thus, it can be applied to every replicated service regardless of the execution behavior of the service. Second, it needs lower processing power during failure-free execution than the active replication one. Third, mobile agents have only to use a unicast primitive, not a multicast one because they send service requests only to the primary service agent. But, the traditional passive replication approach may result in some scalability and performance problems when being applied to the mobile agent system as a fault-tolerant technique for replicated

* Dept. of Computer Science, Kyonggi University, Suwon Gyeonggido, South Korea, *E-mail: jhahn@kyonggi.ac.kr*

services. In other words, to the best of our knowledge, previous works[3,8] uniformly applied the traditional passive replication approach to each replicated service regardless of whether it is deterministic or non-deterministic. But, in this approach, every mobile agent request should be sent only to the primary service agent, which processes the request and coordinates with the other live replicas and then returns a response of the request to the mobile agent. This special role of the primary is necessarily required to ensure the consistency for non-deterministic services. Moreover, the traditional passive replication approach forces all visiting mobile agents to be transferred to and execute their works in order only on the node running the primary service agent of each domain. These inherent features may cause the extreme load condition to occur on the primary service agent when a large number of mobile agents are forwarded to the service domain and access its resources. Thus, this previous strategy may not achieve high scalability and performance.

This paper presents a scalable strategy to apply an appropriate passive replication mechanism to each service according to its execution behavior because deterministic services require weaker constraints to ensure their consistency than non-deterministic ones like figure 1. For this goal, two passive replication mechanisms are designed in this paper. The first mechanism for non-deterministic services is named PSR-N and the second mechanism for deterministic services, PSR-D. They both allow visiting mobile agents to be forwarded to and execute on any node performing a service agent, not necessarily the primary agent. Especially, in case of the second mechanism PSR-D, after a backup service agent has received a mobile agent request and obtained the delivery sequence number of the request from the primary service agent, the backup agent, not the primary one, is responsible for processing the request and coordinating with the other replica service agents. Due to this feature, PSR-D is more lightweight than PSR-N tolerating non-deterministic servers such as multi-threaded servers.



Figue 1: The Strategy for Enhancing Scalability of Replicated Services for Mobile Agents

2. System Model

We consider an asynchronous distributed system where there is no global memory and clock, and no bound on message delay. This system is augmented with a unreliable failure detector[4] in order to solve the impossibility problem on distributed consensus[7]. The system provides mobile agents with a set of $n(n > 0)$ services $S = [s_0, s_1,\ldots, s_{n-1}]$. A service $s_i(0 \le i \le n-1)$ is accessed by the mobile agents and implemented by a set of $k(k > 0)$ service agents $u_i = [u_i^0, u_i^1,\ldots, u_i^{k-1}]$ as a group, where u_i^w executes on a node $node_i^w$ $(0 \le w \le k-1)$. We consider every service agent has its local state and atomically changes the state through agent interactions. In order to perform an assigned task on behalf of its user, a mobile agent a_j executes on a sequence of $l(l > 0)$ places $p_j = [p_{j0}, p_{j1},\ldots, p_{j(l-1)}]$ according to its itinerary, which may be statically determined before the mobile agent is launched at the agent source or dynamically while progressing its execution. Executing a_j at a place $p_{jm}(0 \le m \le l-1)$ is called a stage $Stage_j^m$ of the agent execution. The resulting state of $a_j(j \ge 0)$ executing all the stage operations on the place $p_{jr}(0 \le r \le m)$ is denoted by a_j^{m+1}, which will be forwarded from p_{jm} to $p_{j(m+1)}$ and executed on place $p_{j(m+1)}$.

Agents have crash-failure semantics, in which they lose contents in their volatile memories and stop their executions. We assume that the communication channels are immune to partitioning, reliable and FIFO.

Linearizability is a strong correctness criterion for replicated services that most passive replication mechanisms for real-world distributed systems in this literature ensure[8,17]. It is based on real-time dependency. In other words, it implies that agents appear to be interleaved at the granularity of complete operations, and that the order of non-overlapping operations is preserved[10]. We use it as the basic consistency criterion for the passive replication mechanisms shown in this paper. Any replication mechanism attempting to ensure this criterion should satisfy the following specification of the Generic Replication Problem[8].

[Termination Property] If a non-faulty client sends a request req, it eventually receives a response $response_{req}$.

[Agreed Order Property] If there is an event update(req), the modification of the state of a replica as the result of processing req, such that a server replica performs update(req) as its *i*-th update event, then all server replicas that performs the *i*-th update event execute update(req) as their *i*-th update event.

[Update Integrity Property] For any request req, every replica performs update(req) at most once, and only if an event send(req), the emission of request req, was previously executed by a client.

[Response Integrity Property] For any event receive($response_{req}$), the reception of response $response_{req}$,

performed by a client, the event update(req) is executed by some non-faulty replica.

3. THE PROPOSED SERVICE REPLICATION MECHANISMS

When mobile agents are concurrently transferred to a non-deterministic replicated service, the consistency is ensured by applying the traditional passive replication mechanism[3]. When mobile agent a_j attempts to use the service of a non-deterministic replicated service domain u_i, a_j should be transferred from its current node to the node where the primary service agent of u_i, u_i^{prim}, executes. Then, the following phases are performed:

1st-Phase: When u_i^{prim} receives a request message from mobile agent a_j, u_i^{prim} processes the message. After the execution, it generates (response, psn, next_state), where response is the response of the message, psn identifies the processed request message and next_state is the state of u_i^{prim} updated by processing the request.

2nd-Phase: u_i^{prim} sends all backup service agents the update message (response, psn, next_state, *j*, reqid) using View Synchronous Multicast(**VSCAST**)[13] respectively, where *j* identifies a_j and reqid is the send sequence number of the request message. In here, **VSCAST** is the multicast communication primitive used to ensure correctness of the passive replication approach. When each backup service agent receives the update message, it updates its state using next_state, maintains (response, *j*, reqid) in its buffer and then sends an acknowledgement message to u_i^{prim}. In this case, (response, *j*, reqid) is needed to ensure the exactly once semantics despite u_i^{prim}'s failure.

3rd-Phase: After receiving an acknowledgement message from every live backup service agent, u_i^{prim} sends response to a_j.

Therefore, this mechanism forces all visiting mobile agents to be transferred to and execute their works in order only on the node running the primary service agent of each domain. This behavior may cause the extreme load condition to occur when a large number of mobile agents are forwarded to the service domain to use the non-deterministic service. Thus, we introduce a passive replication mechanism for non-deterministic service, PSR-N, to solve the problem by allowing mobile agents to be forwarded to and execute on each a node performing a service agent, not necessarily the primary agent. The mechanism forces each service agent *p* to forward every request received from mobile agents to the primary service agent. Afterwards, the primary agent performs the phases 1 through 2 previously mentioned to satisfy the consistency condition for non-deterministic service. After receiving an acknowledgement from every normal backup service agent, the primary agent sends the response of the request to the service agent *p*, which forwards it to the corresponding mobile agent.

Figure 2 illustrates how the mechanism PSR-N executes. In this figure, mobile agents a_t^1, a_s^m and a_r^n are transferred to and execute on the nodes $node_i^2$, $node_i^1$ and $node_i^0$ for accessing service u_i. Then, the mobile agents perform their tasks via u_i^2, u_i^1 and u_i^0 respectively. For example, if u_i^1 receives a request from a_s^m, it forwards the request to the primary service agent u_i^2. Afterwards, u_i^2 processes the request and coordinates with the other service agents u_i^1 and u_i^0 using **VSCAST**. When receiving acknowledgements from the two backup service agents, it sends the response of the request to u_i^1, which forwards the response to a_s^m. From this example, we can see that PSR-N improves scalability of replicated services in case a large number of mobile agents attempt to use a particular service simultaneously. If a mobile agent accesses a service via a backup service agent in this mechanism, two more messages are required per request compared with the traditional mechanism. However, the additional message cost is not significant because each service domain is generally configured on a local area network like Ethernet.

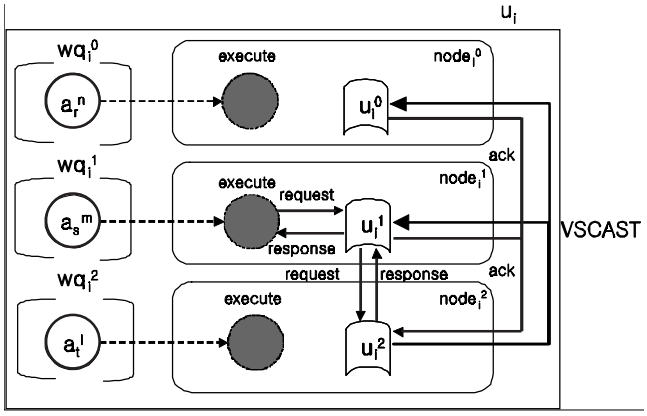


Figure 2: An Example for Mechanism PSR-N

A deterministic replicated service requires weaker constraints to ensure the consistency than a non-deterministic one. In other words, after the primary service agent has determined the processing order of every request from mobile agents, it is not necessary that only the primary agent handles all requests, and coordinates with the other replica service agents like in the mechanism PSR-N. With this observation, we attempt to use a lightweight passive replication mechanism, PSR-D, for each deterministic service domain. The proposed mechanism has the following features.

+ Each mobile agent can use a service via the primary service agent or a backup one.
+ Only the primary service agent determines the processing order of every request from mobile agents.
+ After a backup service agent has received a mobile agent request and obtained the order of the request

from the primary service agent, the backup agent, not the primary one, processes the request and coordinates with the other replica service agents including the primary agent.

Due to these desirable features, this mechanism enables each visiting mobile agent to be forwarded to and execute on a node running any among replicated service agents in the service domain. If mobile agent a_j is transferred to the node where a backup service agent u_i^{backup} executes, PSR-D executes the following phases. Otherwise, the three phases of PSR-N are performed.

1st Phase: When u_i^{backup} receives a request message from a_j, u_i^{backup} asks the primary service agent u_i^{prim} the psn of the request message. In this case, after u_i^{prim} determines the psn of the message, it notifies u_i^{backup} of the psn. Then, u_i^{prim} processes the request message and saves (response, psn, next_state, *j*, reqid) of the message in its buffer. When receiving the psn, u_i^{backup} processes the corresponding request and generates (response, psn, next_state) of the request.

2nd Phase: u_i^{backup} sends the other service agents the update message (response, psn, next_state, j, reqid) using **VSCAST** respectively. When each service agent except for u_i^{prim} receives the update message, it updates its state using next_state, maintains (response, *j*, reqid) in its buffer and then sends an acknowledgement message to u_i^{backup}. If u_i^{prim} receives the update message from u_i^{backup}, it just removes the element (response, psn, next_state, *j*, reqid) for the message from its buffer, saves (response, *j*, reqid) in the buffer and sends an acknowledgement message to u_i^{backup}.

3rd Phase: Once u_i^{backup} receives an acknowledgement message from every other live service agent, it sends response to a_j.

Figure 3 shows an execution of three mobile agents a_t^1, a_s^m and a_r^n attempting to use service u_i via u_i^2, u_i^1 and u_i^0 in the mechanism PSR-D. In this figure, only the procedure is pictured to execute in PSR-D when a_s^m sends a request to u_i^1. In this case, u_i^1 asks the primary agent u_i^2 the psn of the request. u_i^2 determines the psn and sends it to u_i^1. Then, u_i^2 processes the request and saves (response, psn, next_state, *j*+1, reqid) of the request in its buffer. Meanwhile, after obtaining the psn of the request from the primary agent, u_i^1 processes the request and coordinates with the other service agents u_i^2 and u_i^0 using **VSCAST** to satisfy the consistency condition for deterministic service. When u_i^1 receives an acknowledgement from every live agent, it sends the response to a_s^m. From this figure, we can see that the mechanism PSR-D may significantly improve scalability of replicated services by enabling each visiting mobile agent to use a particular service via any among replicated service agents on the service domain and the request processing and coordination load to be distributed between a set of service agents.

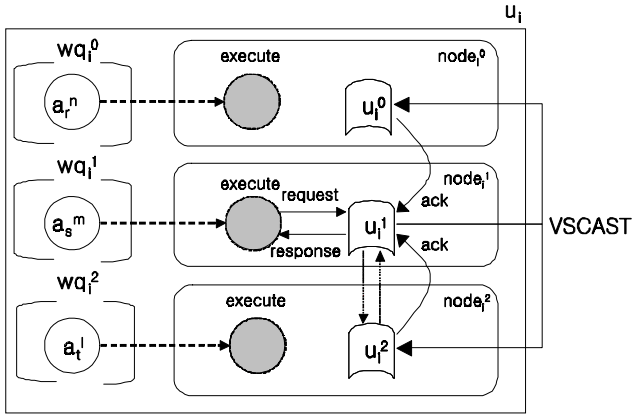


Figure 3: An Example for Mechanism PSR-D

4. CORRECTNESS

In this section, we prove the correctness of the two proposed mechanisms PSR-N and PSR-D using theorems 1 and 2. Especially, lemmas 1 and 2 are given to prove two properties, liveness and ordering, of the second mechanism PSR-D. Also, the atomicity of the mechanism is proved by lemmas 3 and 4.

Theorem 1: *The proposed passive replication mechanism for non-deterministic services PSR-N ensures linearizability.*

Proof: All phases of the mechanism PSR-N are the same as those of the traditional passive replication one[3] except that mobile agents are forwarded to and execute on each a node performing a service agent, not necessarily the primary agent, which forwards every request received from the mobile agents to the primary service agent and its corresponding response from the latter to the first. Linearizability of the traditional one is already proved in the previous work[3]. Therefore, the mechanism PSR-N also ensures the consistency condition for non-deterministic services.

Lemma 1: If a correct mobile agent *k* sends a request $request_k^j$ to replicated server *x* in mechanism *PSR-D*, it eventually receives $response_k^j$ from *x*.

Proof: We prove this lemma by contradiction. Assume that the mobile agent *k* sending the request never receives the response from *x*. As *k* is correct, one correct replica x^l of *x* eventually receives the request from the mobile agent. By the assumption that *k* never obtains the response even if the request is repeatedly resent, x^l has failed before performing step 3. There are two cases to consider as follows:

Case 1: x^l is the primary of *x*.

In this case, a new primary $x^m(l <> m)$ is selected among all backups and *k* perceives x^l's failure. After having known the identity of x^m, the correct mobile agent resends the request to x^m. As it is assumed that at least a replica x^n(l < >

n) is alive in any failure case, x^n eventually receives the request and executes steps 1 through 4. Thus, k receives the response from x^n.

Case 2: x^l is a backup.

In this case, there are two sub-cases to consider:

Case 2.1: The primary is alive.

In our mechanism, the primary previously processed the request after the primary had determined its psn and notified replica x^l of the number psn. Thus, when the primary detects the failure of x^l, it obtains the update information of the request (response$_k^j$, psn, next-state, k, j), from its buffer and sends the information to the remaining live replicas. Receiving the update message, each live backup updates its state using the information and saves response$_k^j$, k, j) in its buffer. If k detects x^l's crash, it resends the request to x. As it is assumed that at least a replica $x^m(l <> m)$ is alive in any failure case, x^m eventually receives the request. Then, it immediately sends the response in its buffer to the mobile agent without processing the request. Thus, k receives the response from x^m.

Case 2.2: The primary crashes.

In this case, the remaining live backups elect a new primary among them. If k perceives x^l's failure, it resends the request to x. As it is assumed that at least a replica x^m ($l <> m$) is alive in any failure case, x^m eventually receives the request and performs steps 1 through 4. Thus, k receives the response from x^m.

Thus, the correct mobile agent receives the response from x in all the cases. This contradicts the hypothesis.

Lemma 2. If one replica x^l of replicated server x has executed a mobile agent's request request$_k^j$ as its i-th update event in mechanism *PSR-D*, all replicas of x that has executed the i-th update event also has executed request$_k^j$ as their i-th event.

Proof: Suppose that the set of all replicas of x that has executed the i-th update event is denoted by SREPS(x). The proof proceeds by induction on the number of all the replicas in SREPS(x), denoted by NUM_OF(SREPS(x)).

[Base case] As NUM_OF(SREPS(x))=1, there is only one replica x^1. Thus, this case is trivially correct.

[Induction hypothesis] We assume that the lemma is true for x in case that NUM_OF(SREPS(x)) = k.

[Induction step] If the (k+1)-th replica has only to have executed the request as its i-th event because k replicas of x have done so by induction hypothesis, the lemma is true for x in case that NUM_OF(SREPS(x)) = k+1. We assume that the the (k+1)-th replica is x^m. There are two cases.

Case 1: Replica x^m received request$_k^j$ directly from mobile agent k.

In this case, x^m has obtained psn of the request from the primary, processed it and sent the k replicas the update message (response$_k^j$, psn, next-state, k, j) respectively. When the replicas receive each the update message, they executes the request as their psn-th event using the update message. By induction hypothesis, psn is equal to i. Thus, the replica x^m has executed the request as its i-th event.

Case 2: Another replica $x^n(n <> m)$ received request$_k^j$ directly from mobile agent k.

In this case, replica x^n previously processed the request as its i-th event and sent the update message to x^m by induction hypothesis. Thus, x^m executes the request as its i-th event by the update message.

By induction, this lemma is correct.

Lemma 3: For any request request$_k^j$, every replica of replicated server x executes the request at most once, and only if mobile agent k previously sent the request to x in mechanism *PSR-D.*

Proof: If a replica x^i executes request$_k^j$ in our mechanism, some replica x^l has processed the request in step 2. Thus, request$_k^j$ was sent to x by the mobile agent k because the request has been received by the replica x^1 in step 1. Also, when x^i executes request$_k^j$, it updates its state using the update message from x^l and saves response$_k^j$ into its buffer. If the replica x^l has failed before response$_k^j$ is sent to the mobile agent, and request$_k^j$ is resent to a live replica x^m, the replica x^m immediately gives response$_k^j$ in the buffer to the mobile agent without processing the request again. Therefore, every replica of x executes request$_k^j$ at most once.

Lemma 4: If mobile agent k has received response$_k^j$ from replicated server x in mechanism *PSR-D,* request$_k^j$ was executed by some correct replica of x.

Proof: We prove this lemma by contradiction. Assume that no replica of x executed the request in this case. As response$_k^j$ is received from x, some replica x^i has previously sent the response to the mobile agent in step 4. In our mechanism, step 4 cannot be started until every replica executes the request. As it is assumed that at least a replica x^l is alive in any failure case, x^l executed request$_k^j$. This contradicts the hypothesis.

Theorem 2: The proposed mechanism *PSR-D* ensures linearizability.

Proof: This theorem is trivially proved by lemmas 1 thru 4 (termination, agreed order, update integrity, response integrity).

5. PERFORMANCE EVALUATION

In this section, extensive simulations are performed to compare our proposed two passive replication mechanisms PSR-N and PSR-D with the traditional Centralized Passive Replication one (PSR-C) using PARSEC discrete-event simulation language[1]. In this simulation, a performance index is used for comparison; average response time per mobile agent request message named $\text{Resp}_{\text{time}}$. Also, suppose a network domain only supports a particular replicated service, which is performed by a group of service agents. The network modeled is a multi-access LAN (100Mbps Ethernet). Nodes connected to the network are identical and uniformly distributed along the physical medium. Each node has at most one process executed on it. Every process communicates with each other using UDP/IP protocol. The message transmission delay between two processes consists of: (1) T_s, the time to execute a send operation on the sending node, (2) T_t, the time to transfer a message from the network interface of the sending node to that of the receiving node, (3) T_r, the time to execute a receive operation on the receiving node. If a process attempts to send or receive a message while its CPU is busy, the message will have to wait until the CPU becomes idle. The network resource is allocated locally based on a FIFO policy, and randomly between nodes. In the simulation, T_s = 0.269 milliseconds (ms), T_s = 0.120 ms and T_r = 0.292 ms for each message frame (1500 bytes). The time required to process a mobile agent request is 1 ms and the time needed for updating the state of a service agent is 80 us. The target service agent selection of each mobile agent in a replicated service domain is a uniformly distributed random variable among its service agents. There are two important simulation parameters. The first is the number of replicated service agents allocated to a service domain called N_{replicas}. The other is the average number of mobile agents transmitted to a service domain per second named N_{magents}. Mobile agents are sent to the network with an interval following an exponential distribution with a mean N_{magents}. The range of the number of request messages generated by a mobile agent transferred to a service domain is from 1 to 20. All experimental results shown in this simulation are all averages over a number of trials.

Figures 4 thru 6 show the average response time $\text{Resp}_{\text{time}}$ of PSR-C, PSR-N and PSR-D for the specified range of the number of replicated service agents allocated to a service domain N_{replicas} in a network. In this figure, their N_{magents}s are 30, 40 and 50 agents per second respectively. In these sub-figures, as N_{replicas}s of the three mechanisms increase, their $\text{Resp}_{\text{time}}$s also increase. The reason is that as the degree of replication scales up, they incur higher synchronization overhead of the service agent group. However, $\text{Resp}_{\text{time}}$s of PSR-N and PSR-D is much lower than that of PSR-C. Also, PSR-D performs better than PSR-N. In particular, as N_{replicas} is higher, the gap between their increasing rates is larger. Additionally, comparing the three sub-figures, we can see

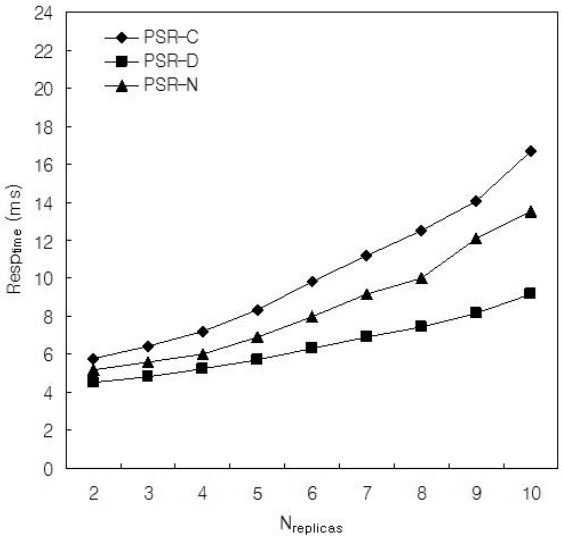


Figure 4: N_{magents} = 30 Agents/Sec

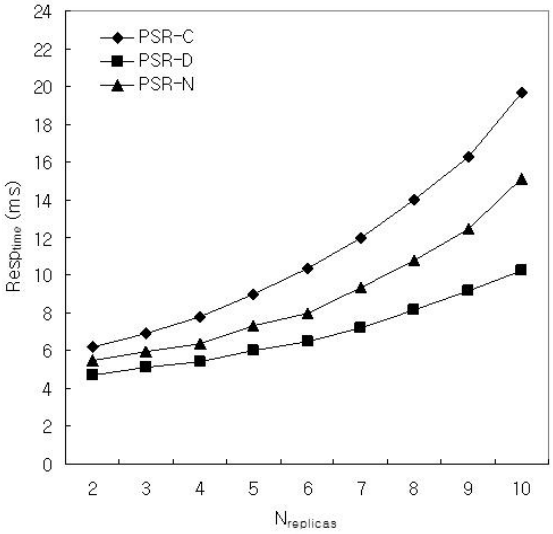


Figure 5: N_{magents} = 40 Agents/Sec

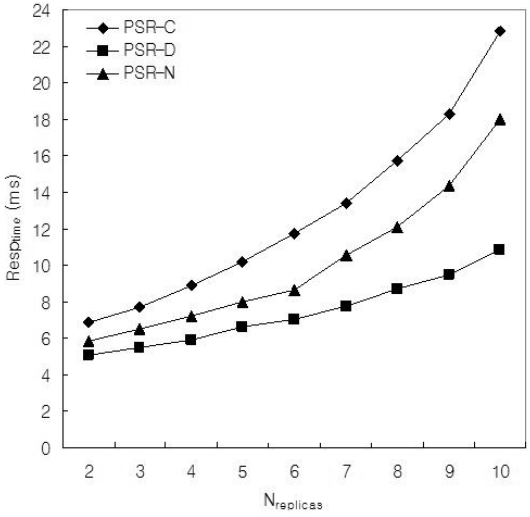


Figure 6: N_{magents} = 50 Agents/Sec

that as $N_{magents}$ increases, the gap between their increasing rates becomes much larger.

6. CONCLUSION

This paper proposed a new strategy to improve scalability of mobile agent systems by applying an appropriate passive replication mechanism to each replicated service according to its execution behavior, deterministic or non-deterministic. For this purpose, we presented the two passive replication mechanisms, PSR-N and PSR-D, for non-deterministic and deterministic services respectively. While ensuring linearizability, they both allow visiting mobile agents to be forwarded to and execute their tasks on any node performing a service agent, not necessarily the primary agent. Especially, the more lightweight mechanism PSR-D allows any service agent to process each mobile agent request and coordinate with the other replica service agents after receiving the request and obtaining its delivery sequence number from the primary agent. Thus, if PSR-D is well-combined with existing load balancing schemes[2], the request processing and coordination load can be evenly distributed among a set of deterministic and replicated service agents based on the workload of each service agent. The simulation results verified that our proposed strategy with the two mechanisms PSR-N and PSR-D considerably outperforms over the one only using the traditional passive replication mechanism in terms of service response time.

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin and H. Y. Song, Parsec: A Parallel Simulation Environments for Complex Systems, *IEEE Computer*, (1998), 77–85.

[2] H. Bryhni, E. Klovning and O. Kure, A Comparison of Load Balancing Techniques for Scalable Web Servers, *IEEE Network*, **14**, (2000), 58–64.

[3] N. Budhiraja, K. Marzullo, F. B. Schneider and S. Toueg, The Primary-backup Approach, *Distributed Systems* (S. Mullender ed.), Ch. 8, Addison-Wesley, Second ed., (1993), 199–216.

[4] T. D. Chandra and S. Toueg, Unreliable Failure Detectors for Reliable Distributed Systems, *Journal of ACM*, **43**, (1996), 225–267.

[5] J. Cui and H. Chae, Mobile Agent based Load Balancing for RFID Middlewares, In *Proc. of International Conference on Advanced Computer Technology*, (2007), 973–978.

[6] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. Murphy and G. Picco, Mobile Data Collection in Sensor Networks: The TinyLime Middleware, *Journal of Pervasive and Mobile Computing*, **4**, (1), (2005), 446–469.

[7] M. J. Fischer, N. A. Lynch and M. S. Paterson, Impossibility of Distributed Consensus with One Faulty Process, *Journal of ACM*, **32**, (1985), 374–382.

[8] X. Defago and A. Schiper, Semi-passive Replication and Lazy Consensus, *Journal of Parallel and Distributed Computing Systems*, **64**, (12), (2004), 1380–1398.

[9] M. Fukuda, K. Kashiwagi and S. Kobayashi, AgentTeamwork: Coordinating Grid-Computing Jobs with Mobile Agents, In Special Issue on Agent-Based Grid Computing, *International Journal of Applied Intelligence*, (2006).

[10] M. Herlihy and J. Wing, Linearizability: A Correctness Condition for Concurrent Objects, *ACM Transactions on Progr. Languages and Syst*., **12**, (3), (1990), 463–492.

[11] D. Powell, M. Chereque and D. Drackley, Fault-tolerance in Delta-4, *ACM Operating Systems Review*, SIGOPS, **25**, (2), (1991), 122–125.

[12] K. Rothermel and M. Schwehm, Mobile Agents, In A.Kent and J. G.Williams (Eds.): *Encyclopedia for Computer Science and Technology*, **40**, (25), (1999), 155–176.

[13] A. Schiper and A. Sandoz, Uniform Reliable Multicast in a Virtually Synchronous Environment, In *Proc. of the 13rd International Conference on Distributed Computing Systems*, (1993), 561–568.

[14] F. B. Schneider, Implementing Fault-Tolerant Services using the State Machine Approach: *A Tutorial, ACM Computing Surveys*, **22**, (4), (1990), 299–319.

[15] Y. Teranishi, PIAX: Toward a Framework for Sensor Overlay Network, *In Proc. of International IEEE CCNC Workshop on Dependable and Sustainable Peer-to-Peer Systems*, (2009).

[16] 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*, **37**, (2007), 493–522.

[17] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme and G. Alonso, Understanding Replication in Databases and Distributed Systems, In *Proc. of the 21st International Conference on Distributed Computing Systems*, (2000), 464–474.