International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.5 (2014), pp.37-48
http://dx.doi.org/10.14257/ijmue.2014.9.5.04

Virtual Sender-based Message Logging for Large-scale Ubiquitous Sensor Network Systems

Jinho Ahn

Dept. of Comp. Scie., Kyonggi Univ., Iuidong, Yeongtong, Suwon 443-760 Gyeonggi, Republic of Korea
jhahn@kgu.ac.kr

Abstract

Sender-based message logging can considerably lower high failure-free overhead of receiver-based message logging resulting from synchronously logging each message into stable storage by using volatile memory of its sender as storage for logging. This beneficial feature can be obtained at the expense of extra communication costs required for allowing message senders to get receive sequence numbers of the messages from their receivers and confirm them with the receivers and slowness and complexity of recovery of each failed process coming from its obtaining message log from the corresponding senders. However, attempting to apply sender-based message logging into large-scale ubiquitous sensor networks requires reducing the number of messages passing on core networks during its message logging and recovery procedures. This paper presents a scalable virtual sender-based message logging algorithm to solve this problem by enabling the broker elected in a group of nodes as virtual sender to localize both of the logging and recovery procedures to a maximum. We show how this algorithm can guarantee the system consistency in case of sequential failures and perform better than the conventional one in terms of message overhead.

Keywords: *Sensor Networks, Scalability, Fault-tolerance, Message Logging, Recovery*

1. Introduction

Among rollback-recovery techniques [7], sender-based message logging [1, 8, 20] with check pointing [2, 3, 6, 11, 14] is one of the most lightweight fault-tolerance techniques to be capable of being applied in those fields. It may considerably lower high failure-free overhead of receiver-based message logging [15, 21] resulting from synchronously logging each message into stable storage, which can be realized by using volatile memory of its sender as storage for logging [1, 7, 8, 10, 20]. This beneficial feature can be obtained at the expense of extra communication costs required for allowing message senders to get receive sequence numbers(RSNs) of the messages from their receivers and confirm them with the receivers and slowness and complexity of recovery of each failed process coming from its obtaining message log from the corresponding senders. As architectural aspects of current and future distributed computing systems are changing to geographically group-based and peer-to- peer based, many of these systems, especially sensor networks, are adopting broker-based architectures to accommodate these topological features well. Thus, this change is making several issues about their fundamental building blocks that should be reconsidered to work well for these newly fashioned systems in highly effective manners. Existing sender-based message logging protocols abbreviated by SBML [1, 8, 20] should also be examined properly

ISSN: 1975-0080 IJMUE

before their application to accommodate this architectural change, which we focus on in this paper. However, although they can be applied to small-scale peer-to-peer flat style architectures well, all the protocols oblivious to the underlying network may not fundamentally provide any breakthrough for ensuring high scalability required in geographically dispersed sensor networks systems composed of a large number of sensor nodes [4, 6, 12, 13, 16, 18]. In this point of view, the two drawbacks mentioned earlier all of the conventional sender-based message logging algorithms have may be amplified and highlighted greatly if they would be applied into large-scale sensor networks, being capable of significantly diminishing the practical value coming from their common advantageous features, even becoming unpractical. In this paper, we present a scalable virtual sender-based message logging algorithm to address the critical problems by employing the large-scale ubiquitous sensor network systems' architectural features stated earlier. In order to satisfy these requirements, this algorithm enables the broker elected in a cluster or group of nodes to localize both of the logging and recovery procedures to a maximum. This feature may considerably reduce the number of both control and data messages passing on core networks incurred during fully message logging and recovery procedures of sender-based message logging.

The rest of the paper is organized as follows. In Section 2, we describe the distributed system model assumed and in Section 3, present a virtual sender-based message logging algorithm and prove its correctness. Sections 4 and 5 show numerical evaluation results and conclude this paper.

2. System Model

A distributed computation consists of a set P of $n(n > 0)$ sequential processes executed on sensor nodes in the system and there is a distributed stable storage that every process can always access that persists beyond processor failures, thereby supporting recovery from failure of an arbitrary number of processors [7]. Processes have no global memory and global clock. The system is asynchronous: each process is executed at its own speed and communicates with each other only through messages at finite but arbitrary transmission delays. Exchanging messages may temporarily be lost but, eventually delivered in FIFO order. We assume that the communication network is immune to partitioning and sensor nodes fail according to the fail stop model where every crashed process on them halts its computation with losing all contents of its volatile memory [17]. Events of processes occurring in a failure-free execution are ordered using Lamport's happened before relation [9]. The execution of each process is piecewise deterministic [5, 19]: at any point during the execution, a state interval of the process is determined by a non-deterministic event, which is delivering a received message to the appropriate application. The k-th state interval of process p, denoted by $si^k_p(k > 0)$, is started by the delivery event of the k-th message m of p, denoted by $dev^k_p(m)$. Therefore, given p's initial state, si^0_p, and the non-deterministic events, $[dev^1_p, dev^2_p, ..., dev^i_p]$, its corresponding state s^i_p is uniquely determined. Let p's state, $s^i_p = [si^0_p, si^1_p, ..., si^i_p]$, represent the sequence of all state intervals up to si^i_p. s^i_p and $s^j_q(p \neq q)$ are mutually consistent if all messages from q that p has delivered to the application in s^i_p were sent to p by q in s^j_q, and vice versa [3]. A set of states, which consists of only one state for every process in the system, is a globally consistent state if any pair of the states is mutually consistent.

In the remainder of this paper, the messages applications generate are called application messages and the messages used for the message logging and recovery procedures, control messages.

3. The Virtual Sender-based Message Logging Algorithm

3.1. Normal Operation Procedure

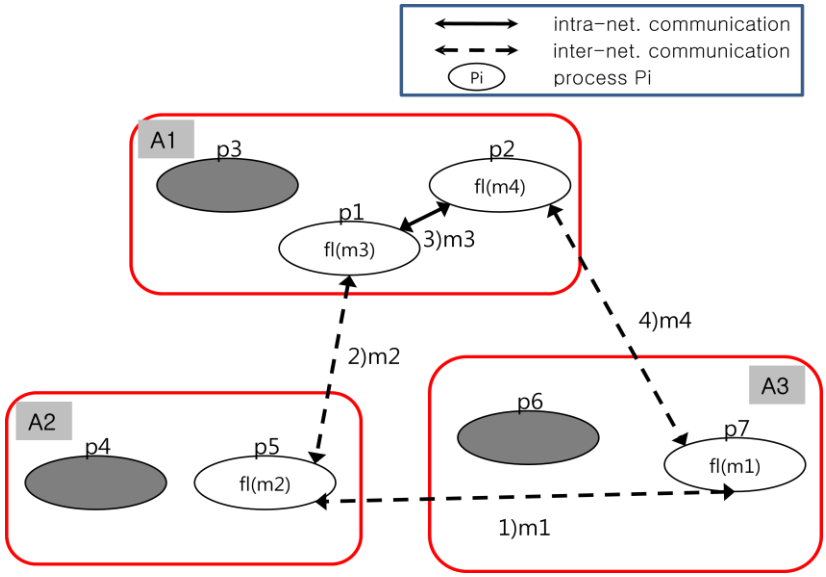

Figure 1. Message interaction-A failure-Free Execution of Existing SBML Algorithms

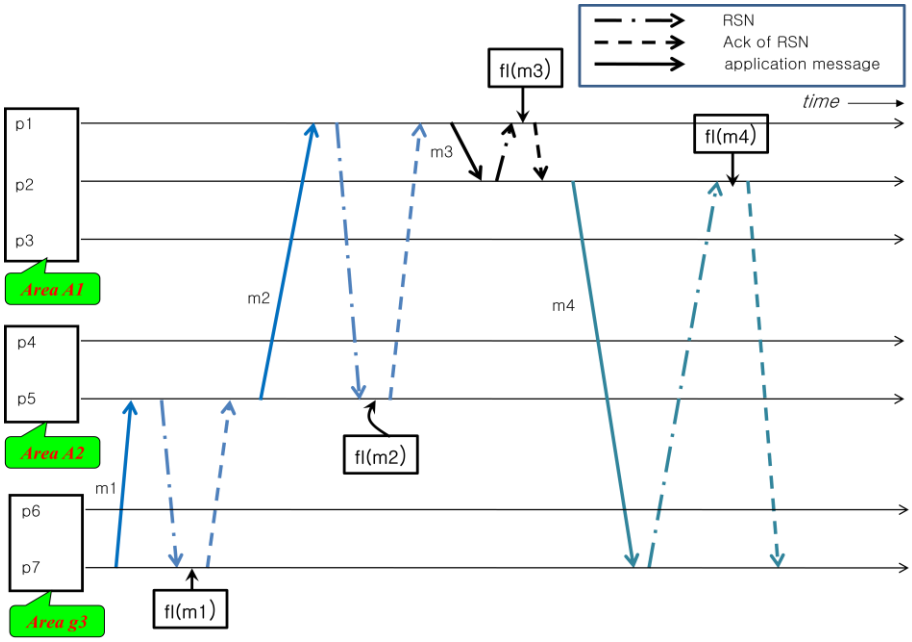

Figure 2. Process Execution-A Failure-free execution of Existing SBML Algorithms

In this paper, we assume that the entire sensor network system consists of a finite set of areas. Each area has a group of nodes among which one having generally, but not mandatorily, the highest capacity of resources such as CPU, memory, storage, network, etc., is elected as broker of their area. Figures 1, 2, 3 and 4 show the same instance of this system model assumed having three areas with 7 processes. In these figures, a gray-shaded ellipse on each area indicates its broker. When a sender process sends a message to its corresponding receiver, SBML requires the following three steps; partially logging the message with its SSN (Send Sequence Number), IDs of its sender and receivers into its volatile storage, saving its RSN returned from its receiver into its log element and informing the receiver of the success of fully logging the message on the sender's volatile storage. However, should they be applied into broker-based sensor networks, the existing SBML algorithms may incur high failure-free overheads in terms of communication cost. This disadvantage occurs if inter-process communications frequently perform across multiple broker controlling areas. For example, in Figures 1 and 2, process p7 sends message m1 to process p5, sending m2 to p1. Then, p1 sends m3 to p2, sending m4 to p7. In this case, the existing SBML algorithms force their three steps with all three messages m1, m2 and m4 to be executed using inter-network communication functions. Unlike the existing SBML algorithms, our algorithm allows the broker to manage the volatile storage for fully logging all the messages sent to every area member, not including itself, from outside its administrative area. This feature enables the second and the third steps to perform locally with each message receiver's broker. For example, Figures 3 and 4 shows how our proposed algorithm operates in the same scenario as Figures 1 and 2. In this case, every sender of each message sent to outside its area like p7, p5 or p2 has only to maintain the partial log information of the message denoted by pl(mi). Instead, each broker like p4, p3 or p6 keeps fully logged messages fl(m1), fl(m2) and fl(m4) on its volatile storage respectively. In Figure 4, we can see that the failure-free steps excluding the first-step with messages m1, m2 and m4 are executed only inside each local area of their receivers respectively.

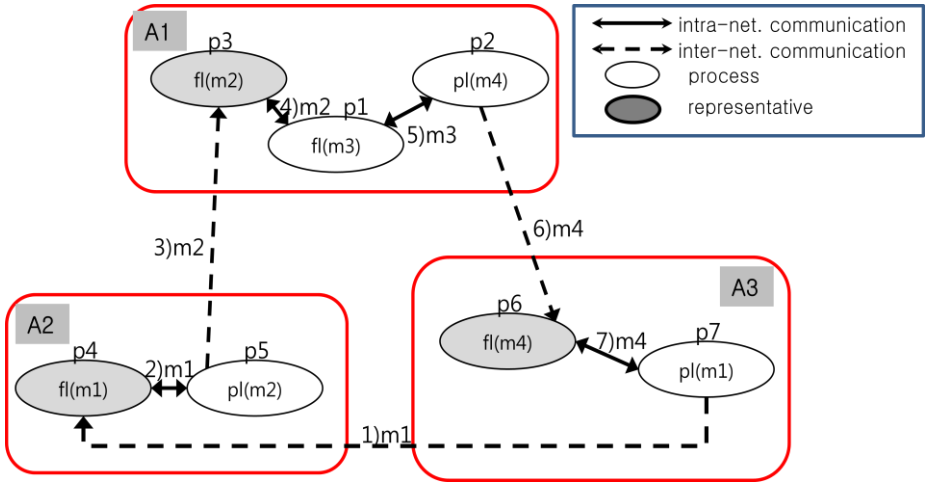

Figure 3. Message Interaction-A Failure-free Execution of our SBML Algorithm

Figure 5 shows the algorithmic description of our proposed SBML algorithm executing during failure-free operation mentioned earlier. All modules for this algorithm are briefly explained as follows. First, **Module** MSG-SEND(m) OF SENDER P_{sndr} is a procedure performed by the sender of the message m when sending and partially logging it on the sender's volatile storage. The second **Module** MSG-RECV(m) OF BROKER BR_{Ai} is

executed by the broker of the area Ai receiving message m, if m's receiver isn't itself, enabling BR_{Ai} to perform the three steps for m. In **Module** MSG-RECV(m) OF RECEIVER P_{rcvr}, message m's receiver assigns a RSN to m and return it to its sender or the broker of the receiver accordingly. **Module** RSN-RCVR(RSN OF m) OF PROCESS P is a procedure for inserting message m's RSN into m's log element on its sender. The last **Module** RSN-CONFIRM(m) OF RECEIVER P_{rcvr} is executed by message m's receiver to know about fully logging m on its sender, releasing messages expected to be sent after the receipt of m.

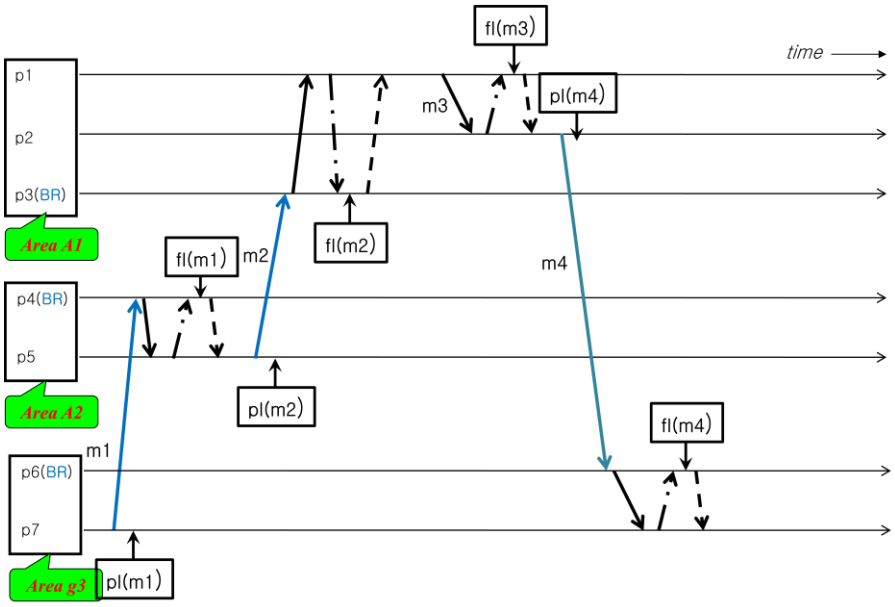

Figure 4. Process Execution-A failure-free Execution of our SBML Algorithm

3.2. Recovery Procedure

If a process fails, SBML has it restore a check pointed state from the stable storage, obtain all the messages received from others before failure and replay them in their RSNs and then FIFO order. For example, in Figures 6 and 7, process p1 has received 12 messages having their RSNs from 4 to 15 among which 8 messages came from outside areas A2 and A3. Although the existing algorithms suffer from their high communication overhead occurring during recovery like in Figure 6, our algorithm can localize p1's recovery procedure like in Figure 7 as follows; First, if a process, not broker, fails according to the crash failure model, it can be recreated on an available node, restore a pre-failure state from its latest checkpoint and broadcast a recovery request only inside its local area. Whenever it receives a reply including logged messages sent to itself from another process, it puts fully logged messages into its replay buffer in RSN order and then partially logged ones in FIFO order. After it has gotten replies from all the other nodes, it replays all the messages sent before its failure after its latest checkpoint in the sorting order. Figure 8 shows the formal expression of our non-broker recovery procedure named **Module** RECOVERY() OF PROCESS P, NOT BROKER. Broker's recovery procedure of our algorithm is more complicated like in figure 9 and may require longer completion time. But, most of broker-based systems assume failures of brokers occur very infrequently.

Module MSG-SEND(m) OF SENDER P_{sndr}
assign its current SSN **to** m ;
increment its current SSN **by** one ;
if(m is a message destined to a process P_{rcvr} in another area not playing the role of area broker)
then
send m **to** the broker of P_{rcvr} ;
else
send m **to** P_{rcvr} ;
log partially m **into** its volatile memory ;

Module MSG-RECV(m) OF BROKER BR_{Ai}
if(m is a message destined to another process P_{rcvr} in its managing area) **then**
forward a message m **to** P_{rcvr} ;
log partially m **into** its volatile memory ;
else
assign its current RSN **to** m ;
increment its current RSN **by** one ;
return RSN of m **to** sender of m ;
disallow all the messages generated for being sent after having received m **to** be sent ;
deliver m **to** its corresponding application ;

Module MSG-RECV(m) OF RECEIVER P_{rcvr}
assign its current RSN **to** m ;
increment its current RSN **by** one ;
if(m is a message sent directly from its original sender P_{sndr}) **then**
return RSN of m to P_{sndr} ;
else
return RSN of m to the broker of P_{rcvr} ;
disallow all the messages generated for being sent after having received m **to** be sent ;
deliver m **to** its corresponding application ;

Module RSN-RCVR(RSN OF m) OF PROCESS P
retrieve m's partial log information, em, **from** its volatile log ;
update RSN of m **into** em ;
confirm fully logging m **to** receiver of m ;

Module RSN-CONFIRM(m) OF RECEIVER P_{rcvr}
allow all the messages delayed sending by m's fully logging procedure until now **to** be sent ;

Figure 5. Message Logging Procedures during Failure-Free Operation

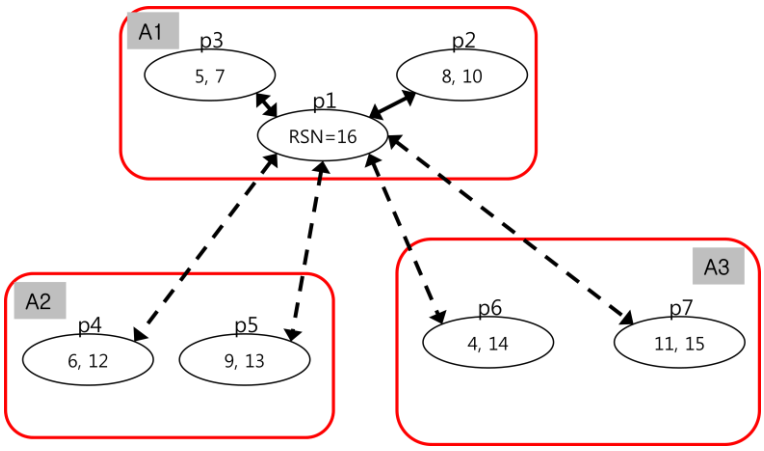

Figure 6. An Example of Recovery Procedure Of Existing SBML Algorithms

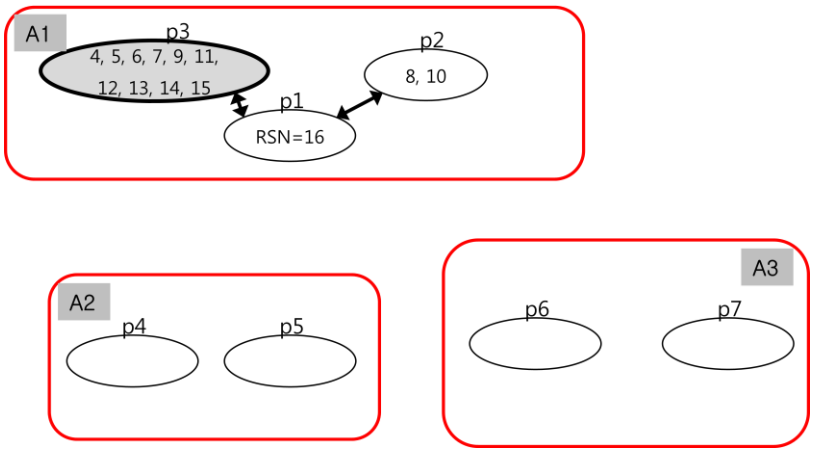

Figure 7. An Example of Recovery Procedure of our SBML Algorithm

Module RECOVERY() OF PROCESS P, NOT BROKER
restore a state **from** its latest checkpoint on stable storage ;
broadcast each a recovery request **to** its area ;
while recovery replies aren't received from all processes in this area **do**
  put fully logged messages for P piggybacked on each reply into $flog_p$ in RSN order ;
  put partially logged messages for P piggybacked on each reply into $plog_p$ in FIFO order ;
for all e $\in$ $flog_p$ st (e.RSN = RSN_p) **do**
  increment RSN_p **by** one ;
  deliver e.m **to** its corresponding application ;
  $flog_p \leftarrow flog_p$ - {e} ;
while $plog_p$ is a non-empty set **do**
  randomly select $\exists$ e in $plog_p$ st (e.m.SSN = $LSSN_p$[e.m.SID]+1) ;
  call Module Msg-Recv(e.m) of Receiver P_{rcvr} ;
  $plog_p \leftarrow plog_p$ - e ;

Figure 8. Recovery Procedures of each Non-Broker Process

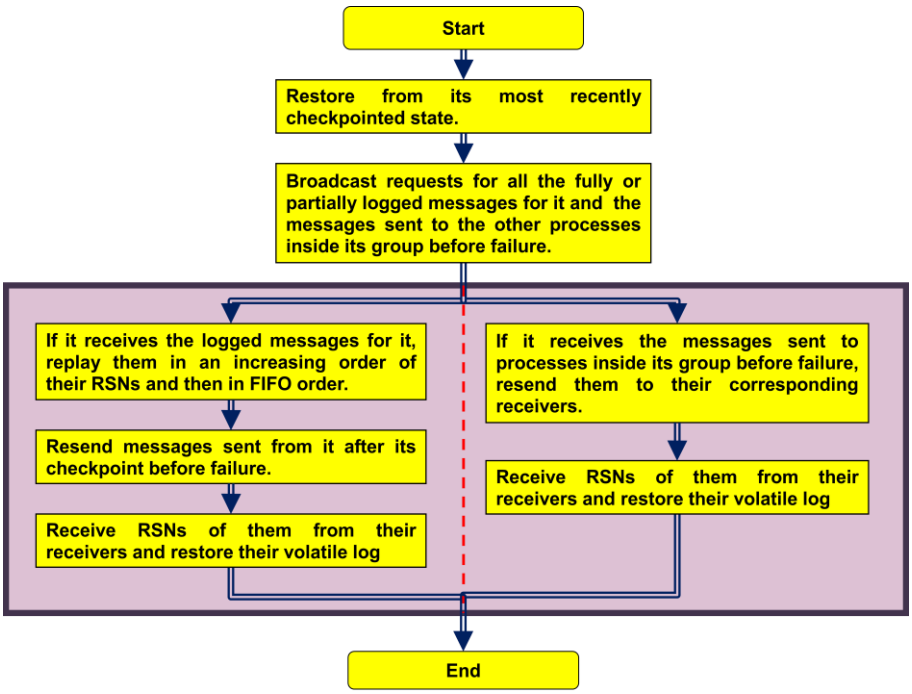

Figure 9. Recovery Procedure of Broker in our SBML Algorithm

3.3. Correctness Proof

Theorem 1 *Our proposed SBML algorithm enables the system to recover to be a globally consistent state in case of sequential failures.*

Proof. We prove this theorem by contradiction. Assume that consistent recovery is impossible even after the recovery procedure of this algorithm has been performed. There are two process failure cases to consider as follows:

Case 1: The failed process *p* isn't a broker.

In this case, the recovering process *p* first restores a pre-failure state from its latest checkpoint on stable storage like in figure 8. Then, it broadcasts a recovery request message to every other member *r* only on its local area including its area broker. In this case, there are two sub-cases to consider:

Case 1.1: *r* isn't a broker.

In this case, *r* sends *p* all the logged messages kept on its volatile memory which were originally sent to *p* by *r* before *p*'s failure.

Case 1.2: *r* is a broker.

In this case, *r* sends *p* all the logged messages kept on its volatile memory which were originally sent to *p* by *r* as well as every other member residing on another area before *p*'s failure.

Therefore, in both subcases, *p* can obtain all the messages received from its communicating members on its local area and on the outside areas before its failure and replay them in their RSN and FIFO order.

Case 2: The failed process *p* is a broker.

In this case, the recovering process *p* first restores a pre-failure state from its latest checkpoint on stable storage like in figure 9. Then, it broadcasts a recovery request message to every other member *r* both on its local area and on the outside areas. In this case, there are two sub-cases to consider:

Case 2.1: *r* has the logged messages whose original receivers were *p* before *p*'s failure.

In this case, *r* sends *p* the full or partial log information of the messages kept on its volatile memory. Then, *p* can replay all the received messages in their RSN and FIFO order.

Case 2.2: *r* has the logged messages whose original receivers were some others on *p*'s local area before *p*'s failure.

In this case, *r* sends *p* the partial log information of the messages kept on its volatile memory. Then, *p* obtains RSNs of the messages from their receivers by retransmitting the messages to them. Hereafter, *p* can become a virtual sender of the other members on its local area.

Therefore, consistent recovery is possible in all the cases. This contradicts the hypothesis.

4. Comparisons

In this section, we present some numerical evaluation for showing superiority of our algorithm(*OSBML*) over the existing one(*ESBML*)[8] in terms of total message cost. For this purpose, several parameters used are defined as follows:

- N_{group}: the number of groups or areas physically or logically partitioned in the system.
- N_{memb}: the average number of processes or nodes located in each group.
- $N(= N_{group} \times N_{memb})$: the total number of processes or nodes in the system.
- N_{msg}: the total number of messages generated in the system.
- C_{core}: the cost of sending a message passing on core networks to a destination in another group.
- C_{local}: the cost of sending a message to a destination within the same group.
- ρ: the probability with which any given message is generated to be sent to the outside area.

Although many WAN-based or Internet-scale broadcast protocols implemented in network layer or application layer were proposed, they require special deployment of IP multicast or control of overlay networks with various membership management protocols. These diversity and complexity make it difficult to measure uniformly the message cost of the broadcast protocols. Due to this reason, we analyze performance of the two message logging algorithms only using uncast functionality. In this evaluation, we assume a geographically dispersed sensor network system model consisting of N_{group} groups having each N_{memb} processes or nodes with two different one-way message costs, C_{core} and C_{local}. With this assumed model, the total message overheads of *ESBML* and *OSBML* occurring during failure-free operations, denoted by $ESBML_{failure-free}$ and $OSBML_{failure-free}$, can be expressed as Eq.1 and Eq. 2 respectively.

$$ESBML_{failure-free} = 2*C_{core}*N_{msg}*\rho+2*C_{local}*(1-\rho)*N_{msg} \quad (1)$$

$$OSBML_{failure-free} = 3*C_{local}*N_{msg}*\rho+2*C_{local}*(1-\rho)*N_{msg} \quad (2)$$

The amount of reduced failure-free message overhead of *OSBML* against *ESBML*, $\Delta MO_{failure-free}$(= Eq. 1 − Eq. 2), is Eq. 3.

$$\Delta MO_{failure-free} = (2*C_{core}-3*C_{local})*N_{msg}*\rho \quad (3)$$

From Eq. 3, we can see that $\Delta MO_{failure-free}$ may significantly become bigger as the number of messages generated, the probability of ρ and the difference between C_{core} and C_{local} grow.

Let us clarify how *OSBML* may improve the scalability of the entire system during failure-free operation using figure 10 compared with ESBML. Figure 10 shows the variation of the ratio of *ESBML*'s failure-free message overhead $ESBML_{failure-free}$ against *OSBML*'s one $OSBML_{failure-free}$ in various aspects. In this figure, the x-axis is the ratio of C_{core} to C_{local} (C_{core}/ C_{local}), denoted by $Ratio_{comm}$, and the y-axis, the ratio of $ESBML_{failure-free}$ to $OSBML_{failure-free}$ ($ESBML_{failure-free}$ / $OSBML_{failure-free}$), denoted by $Ratio_{failure-free}$. The values of $Ratio_{comm}$ range from 10 through 100 incrementing by 10. The probability of ρ in this figure is 0.05(5%),

0.1(10%), 0.2(20%) and 0.3(30%) respectively. As the average ratio $Ratio_{comm}$ increases in the figure, the corresponding value of $Ratio_{failure-free}$ also becomes higher. In particular, although the difference between their average output ratios is not high in the small value of ρ, it is much bigger as the probability of ρ is larger. From this figure, we can see that the scalability of the localized logging nature of *OSBML* is considerably manifested as ρ and the gap between the inter- to intra-bandwidths increase.

Second, the total message overheads of *ESBML* and *OSBML* resulting from a single non-broker failure, denoted by $ESBML_{recovery}$ and $OSBML_{recovery}$, can be expressed as Eq.4 and Eq. 5 respectively.

$$ESBML_{recovery} = 2*((N_{group}-1)*N_{memb}*C_{core} + (N_{memb}-1)*C_{local}) \quad (4)$$

$$OSBML_{recovery} = 2*(N_{memb}-1)*C_{local} \quad (5)$$

The amount of reduced recovery message overhead of *OSBML* against *ESBML*, $\Delta MO_{recovery}$(= Eq. 4 − Eq. 5), is Eq. 6.

$$\Delta MO_{recovery} = 2*(N_{group}-1)*N_{memb}*C_{core} \quad (6)$$

Eq. 6 indicates that as the number of groups, the group size and C_{core} become larger, $\Delta MO_{recovery}$ may enormously increase.

Let us verify how much *OSBML* may decrease recovery message overhead using figure 11 compared with *ESBML*. In figure 11, we show the variation of the ratio of *ESBML*'s recovery message overhead $ESBML_{recovery}$ against *OSBML*'s one $OSBML_{recovery}$ with varying $Ratio_{comm}$ in case N_{group} is 5, 10, 15 and 20 respectively. As $Ratio_{comm}$ becomes bigger in this figure, their $Ratio_{recovery}$s are also increasing. Especially, as N_{group} grows larger, the values of $Ratio_{recovery}$ are stepping up to higher levels. This outcome arises from the reason that the increase of the number of groups and $Ratio_{comm}$ force both *ESBML* to significantly generate a large number of high cost outgoing messages for communicating between processes on different areas or groups during recovery compared with *OSBML*.

Third, the total message overheads of *ESBML* and *OSBML* occuring when a single broker fails can be both expressed as Eq.4 respectively. In this case, $\Delta MO_{recovery}$ is 0.

In conclusion, these results show that our algorithm with the seamless localization using virtual sender concept has a potential of highly decreasing both failure-free and recovery message overheads incurred by the conventional sender-based message logging ones applied to large-scale broker-based sensor networks.

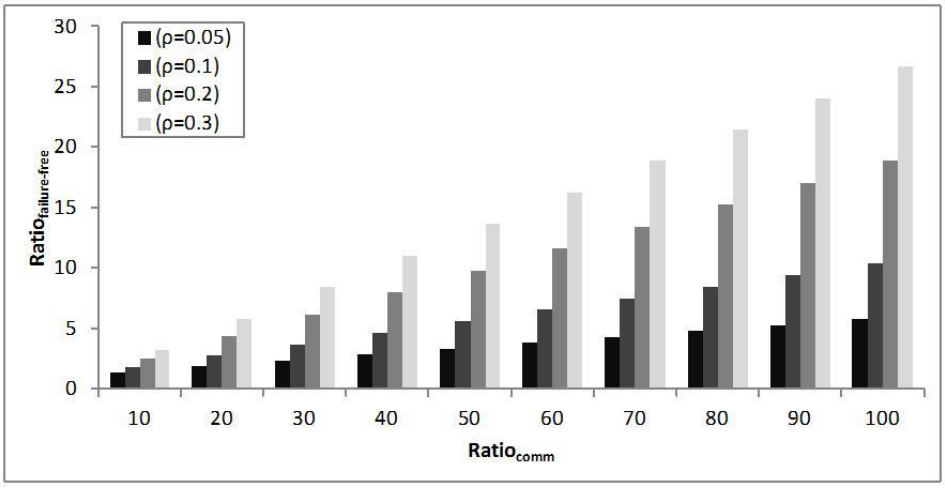

Figure 10. Ratio of $ESBML_{failure-free}$ to $OSBML_{failure-free}$, $Ratio_{failure-free}$, with Varying Values of ρ and (C_{core} / C_{local}), $Ratio_{comm}$

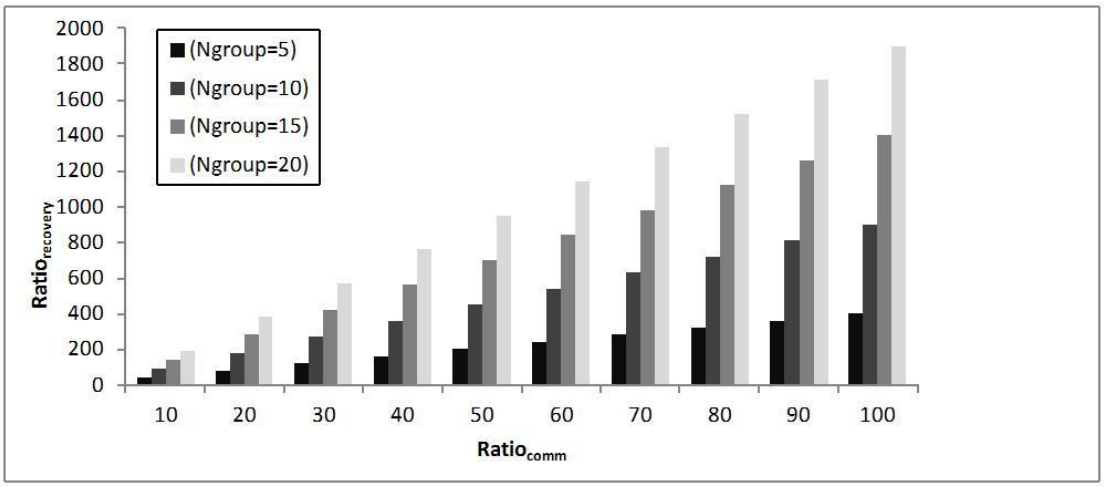

Figure 11. Ratio of $ESBML_{recovery}$ to $OSBML_{recovery}$, $Ratio_{recovery}$, with Varying Values of N_{group} and $Ratio_{comm}$

5. Conclusions

This paper proposed a virtual sender-based message logging algorithm to utilize the large-scale ubiquitous sensor network systems' architectural features, considerably reducing extra communication costs required for allowing message senders to get receive sequence numbers (RSNs) of the messages from their receivers and confirm them with the receivers and slowness and complexity of recovery of each failed process coming from its obtaining message log from the corresponding senders. In order to satisfy these requirements, it enables the broker elected in a cluster or group of nodes to localize both of the logging and recovery procedures to a maximum. This feature may enormously decrease the number of control and data messages passing on core networks incurred during fully message logging and recovery procedures of sender-based message logging. We showed how this algorithm can guarantee the system consistency in case of sequential failures and perform better than the conventional one in terms of message overhead.

Acknowledgements

This research was supported by Kyonggi University Research Grant 2012(Project No.: 2011-027).

References

[1] A. Bouteiller, F. Cappello, T. Herault, G. Krawezik, P. Lemarinier and F. Magniette, "MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging", In Proc. of the Int'l Conf. on High Performance Networking and Computing, **(2003).**

[2] D. Buntinasd, C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello, "Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI Protocols", Future Generation Computer Systems, vol. 24, **(2008)**, pp. 73-84.

[3] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", ACM Transactions on Computer Systems, vol. 3, no. 1, **(1985).**
[4] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer and M. Snir, "Toward Exascale Resilience", International Journal of High Performance Computing Applications, vol. 23, no. 4, **(2009).**
[5] F. Cappello, A. Guermouche, and M. Snir, "On Communication Determinism in Parallel HPC Applications", In 19th International Conference on Computer Communications and Networks (ICCCN 2010), **(2010).**
[6] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi, "Hybrid Checkpointing Using Emerging Nonvolatile Memories for Future Exascale Systems", ACM Transactions on Architecture and Code Optimization, vol. 8, no. 2, **(2011).**
[7] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems, ACM Computing Surveys, vol. 34, no. 3, **(2002).**
[8] D. Johnson and W. Zwaenpoel, "Sender-Based Message Logging, Int'l Symp. on Fault-Tolerant Computing", pp. 14-19, **(1987).**
[9] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM, 21, **(1978).**
[10] T. LeBlanc, R. Anand, E. Gabriel, and J. Subhlok, "VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes, Lecture Notes In Computer Science", vol. 5759, **(2009)**, pp. 124-133.
[11] H. F. Li, Z. Wei, and D. Goswami, "Quasi-atomic recovery for distributed agents", Parallel Computing, vol. 32, **(2009)**, pp. 733-758.
[12] C. Li, H. Zhang, B. Hao and J. Li, "A Survey on Routing Protocols for Large-Scale Wireless Sensor Networks", Sensors, vol. 11, no. 4, **(2011)**, pp. 3498-3526.
[13] K. Liu, Q. Ma, X. Zhao and Y. Liu, "Self-diagnosis for large scale wireless sensor networks", Proc. of IEEE INFOCOM 2011, **(2011)**, pp. 1539-1547.
[14] Y. Luo and D. Manivannan, "FINE: A Fully Informed aNd Efficient communication-induced checkpointing protocol for distributed systems", J. Parallel Distrib. Comput., vol. 69, **(2009)**, pp. 153-167.
[15] M. Powell and D. Presotto, "Publishing: A reliable broadcast communication mechanism", In Proc. of the 9th International Symposium on Operating System Principles, **(1983)**, pp. 100-109.
[16] C. Rosiers, G. Chelius, T. Ducrocq, E. Fleury, A. Fraboulet, A. Gallais, N. Mitton, T. Noel and J. Vandaele, "Using SensLAB as a First Class Scientic Tool for Large Scale Wireless Sensor Network Experiments", In Proc. Of the 10th international IFIP TC 6 conference on Networking, **(2011)**, pp. 147-159.
[17] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: an approach to designing fault-tolerant distributed computing systems", ACM Transactions on Computer Systems, vol. 1, **(1985).**
[18] B. Schroeder and G. A. Gibson, Understanding Failures in Petascale Computers, Journal of Physics: Conference Series, vol. 78, 012022, **(2007)**, pp. 11.
[19] R. E. Strom and S. A. Yemeni, "Optimistic recovery in distributed systems", ACM Transactions on Computer Systems, vol. 3, **(1985)**.
[20] J. Xu, R. B. Netzer, and M. Mackey, "Sender-based message logging for reducing rollback propagation", In Proc. of the 7th International Symposium on Parallel and Distributed Processing, **(1995)**, pp. 602-609.
[21] B. Yao, K. Ssu, and W. Fuchs, "Message Logging in Mobile Computing", In Proc. of the 29th International Symposium on Fault-Tolerant Computing, **(1999),** pp. 14-19.

Author

Jinho Ahn, he 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 a 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.

