International Journal of Control and Automation
Vol. 11, No. 4 (2018), pp.1-10
http//dx.doi.org/10.14257/ijca.2018.11.4.01

# Large-scale Deadline-Constrained Causal Order Broadcast Algorithm for Enhancing Message Delivery Success Rate Performance

Jinho Ahn[*]

*Dept. of Computer Science, Kyonggi University, Suwon-si, Gyeonggi-do, Korea*
*Tel.: +82 31 249-9674, FAX: +82 31 249-9673*
*jhahn@kgu.ac.kr*

***Abstract***

*Most of existing deadline constrained causal order broadcast algorithms force any group member to drop late messages received before the expiration of their deadlines, but not respecting causal order condition. However, their users want to get as many messages as possible in their cause-effect order within the earliest deadline among them. In this paper, we propose an efficient real-time constrained causal order broadcast algorithm to highly improve responsiveness and minimize the number of late messages discarded. In order to satisfy these features, the algorithm should maintain each message broadcast to a group into the volatile storage of every group member. Therefore, it enables in-transit predecessors of each received message to be obtained from other group members as fast before its deadline as possible. The simulation results show the proposed algorithm performs better than the traditional algorithm in terms of the message delivery success ratio.*

***Keywords:*** *Distributed System, Real-time Constraint, Group Communication, Broadcast, Message Delivery Order*

## 1. Introduction

Causal order delivery to a broadcast group is a very important issue in the fields of sensor networks, video conferencing, stock trading, auction sales and so on [6, 7, 11]. This message ordering condition can be satisfied if any two message sending events have cause-effect relation and the same destination, their corresponding delivery events should occur on the destination in their sending order. In order to ensure this ordering constraint, two approaches may generally be used as follows. First, if a group member receives a message capable of violating the constraint, the message delivery to the application is forced to wait for releasing the restriction caused by its predecessors [1, 4, 5, 9, 12]. Second, if deadline-constrained causal order requirement should be guaranteed, late messages, whose deadlines have passed or whose successors already received have exceeded their deadlines, are discarded. In the latter case, their users attempt to see as many messages as possible in their cause-effect order within the earliest deadline among them. Many researches on developing application-level broadcast communication algorithms have been performed to try to satisfy one of the three properties such as causal message ordering, message deadline assurance and high responsiveness. However, to the best of our knowledge, there exists no algorithm to satisfy all the three properties so far [2, 3, 8, 9, 12]. In this paper, we propose an efficient real-time constrained causal order broadcast algorithm to highly improve responsiveness and minimize the number of late messages discarded like in Figure 1. This feature may help each group member locally

Received (October 5, 2017), Review Result (January 10, 2018), Accepted (January 16, 2018)
[*] Corresponding Author

make correct decisions with consistent information obtained from the messages even in soft real-time constrained environments.

The remainder of the paper is organized as follows. Section 2 describes the distributed system model based on broadcasting links and, Section 3, the problems of the previous causal order broadcasting algorithms in detail. In Section 4, a novel deadline-constrained causal order broadcast algorithm is introduced and, in Section 5, the experimental results are presented to show the superiority over the previous deadline-constrained one. Lastly, Section 6 summaries this paper.

Desirable Properties
Broadcast Comm.
Causal Order
Deadline
Responsiveness
Our Goal
Highly Responsive Broadcast Algorithm for Ensuring Deadline-Constrained Causal Order

**Figure 1. The Three Properties of the Proposed Broadcast Algorithm**

## 2. System Model

We assume an asynchronous distributed system with no global memory, consisting of a set of processors, processes and communication channels. For simplicity, we assume one process per processor. Processes can communicate with each other by exchanging messages through real-time unreliable channels, meaning messages may be dropped or duplicated on these channels. But, it is assumed that the channels are immune to partitioning. Finally, we assume that processes may fail based on the crash-failure model, in which they lose contents in their volatile memories and stop their executions [14]. This system is augmented with an unreliable failure detector [6] in order to solve the impossibility problem on distributed consensus [8].

All events produced when each process performs until completing its own task can be classified into internal events and communication events. Communication events are divided into send and receive events. All these events of processes occurring in a failure-free execution are ordered using Lamport's happened before relation[10] as follows:

**Causal Order Broadcast Relation**: it is generally defined based on Lamport's happened before relation. Like in Figure 2, message $m_1$ may have potentially caused another message $m_2$ (denoted as $m_1 \rightarrow^b m_2$) if:

- process p broadcasts message $m_1$ and then $m_2$ (case ❶).
- process p broadcasts message $m_1$ and then another process q, $m_2$ (case ❷).

• there exists some message $m_\alpha$ such that $m_1 \rightarrow^b m_\alpha$ and $m_\alpha \rightarrow^b m_2$ (case ❸)

This relation is very useful for making the entire events produced by a group of processes partially ordered. When a message is received by a process, it can be delivered after its delivery condition or other constraint, *e.g.*, deadline constraint, has been satisfied.

Deadline-constrained message delivery requires existence of a global clock where all processes can get their clock values and sequence causally dependent events by assigning the values to them. For this purpose, several global clock synchronization protocols whose clock drift values range from 5 to 10 msec. have been developed. The deadline of a message m is measured as the global clock value before which message m must be delivered at least, including its local clock drift time. Therefore, if the deadlines of all messages scheduled to be sent are reasonably assigned, most of them can be satisfied for delivering their corresponding messages to the applications. But, message lost or congestion may incur message transfer delay, resulting in exceeding the deadline(s) of itself or its successors, which should drop the message in the soft real-time environments.

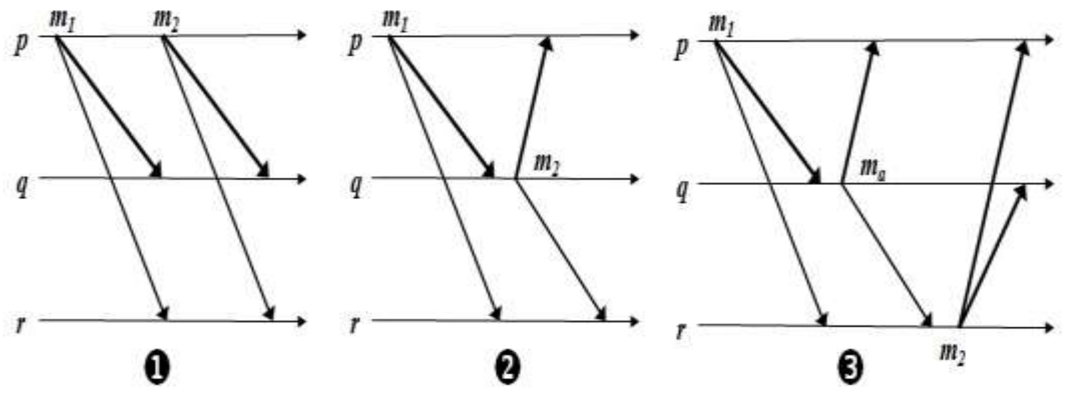

**Figure 2. Three Cases of Causal Order Broadcast Relation**

## 3. Previous Causal Order Delivery Algorithms

First of all, in order to ensure the first property called causal message ordering, when a group member receives a message capable of violating the constraint, the delivery of the message to the corresponding application is forced to wait for releasing the restriction caused by its predecessors. The previous algorithms can be divided into the following two approaches. The first approach is designed for optimizing the amount of the control information piggybacked on each sent message. The second one is to try to reduce the delivery time of the message to the corresponding application by piggybacking all of its predecessors on each sent message. But, the first one is generally preferred. Let me show you an example in Figure 3 for this approach. In this example, there is a broadcast group consisting of four processes p, q, r and s. The three processes p, q and s broadcast three messages $m_1$, $m_2$ and $m_3$ to their own group in order. But, in this case, as the process r receives the third message $m_3$ from s, r hasn't received $m_3$'s predecessors, $m_1$ and $m_2$, yet. Therefore, $m_3$'s delivery should be delayed until $m_1$ and $m_2$ are received and delivered at this point. Like in this example, the casual order delivery algorithm with vector timestamp piggybacked can ensure the first constraint.

Secondly, the deadline constrained approach has been introduced to satisfy the two properties, causal message ordering and message deadline, together. This approach discards late messages whose deadlines have passed or whose successors already received have been delivered. Figure 4 shows how each group member should deliver messages broadcast to its own group according to its encountering situation in this approach. In the figure, the same execution scenario played out like in Figure 3. However, different deadlines of the three messages are set to meet if they are delivered. In this case, processes p, q and s, can fortunately receive all the three messages before the earliest deadline, $deadline_{m3}$, to be met. On the other hand, process r first receives the third message, $m_3$, but, its predecessors, $m_1$ and $m_2$, haven't arrived at r until $m_3$'s deadline is reached. In order to ensure the real-time and causal ordering constraints at the same time, r should deliver $m_3$ to the corresponding application before $deadline_{m3}$. Afterwards, even if $m_1$ and $m_2$ arrive at r before their respective deadlines, they must be dropped to ensure the causal order relation like in the figure.

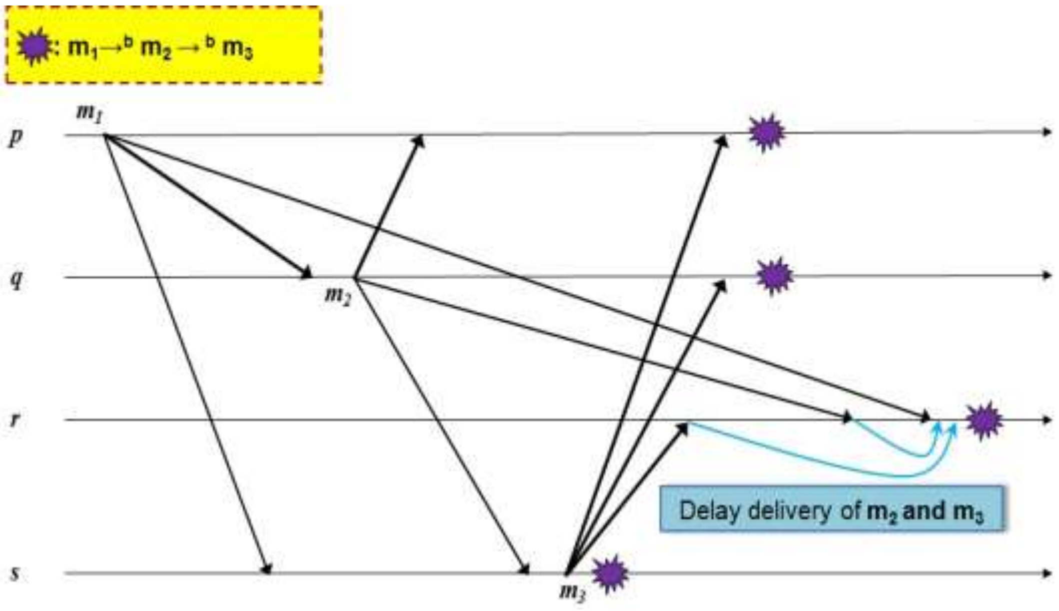

**Figure 3. An Example Showing how to Ensure Inter-Message Causality Relation in the Message Delivery Delaying Approach**

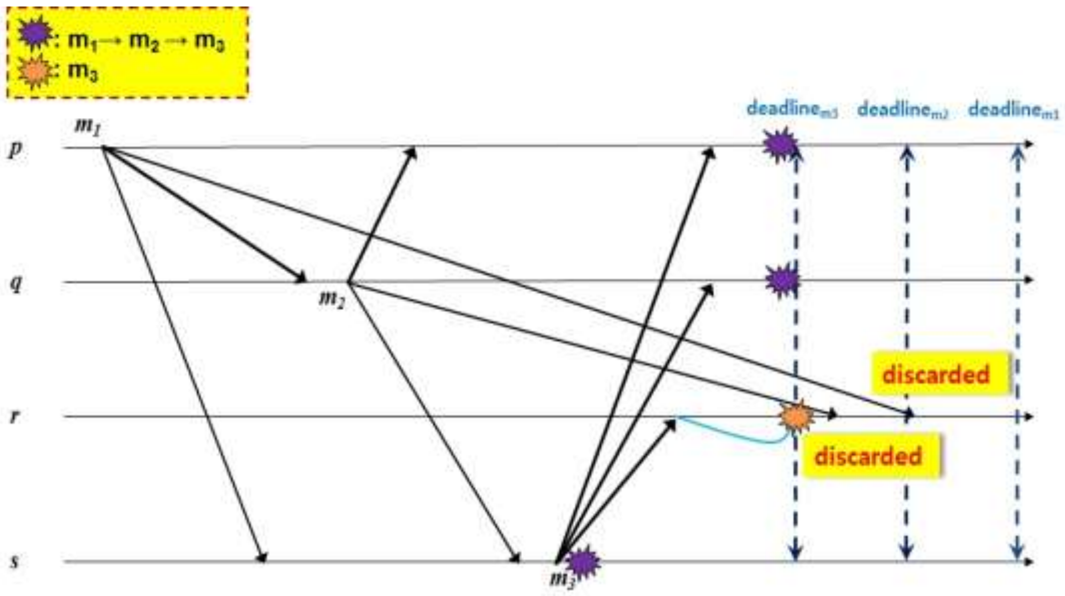

**Figure 4. An Example Showing how to Ensure Inter-Message Causality Relation in the Deadline-Constrained Approach**

## 4. The Proposed Broadcast Algorithm

Generally, the users want to get as many messages as possible in their cause-effect order within the earliest deadline among them. Thus, our proposed algorithm intends to be designed having the following features.

- Minimize the number of late messages discarded.
- Improve responsiveness highly.

Therefore, in order to satisfy these features, the algorithm should maintain each message broadcast to a group into the volatile storage of every group member. Also, it allows in-transit or lost predecessors of each received message to be obtained from another group member as fast before its deadline as possible.

Let us show how our algorithm executes to achieve this goal using an example. In Figures 5 and 6, there is a broadcast group consisting of 4 processes, p, q, r and s, sending 3 messages, m1, m2 and m3, to all members in order (by executing Module B-SEND(m, $\text{deadline}_m$) in Figure 8), whose deadlines are $\text{deadline}_{m1}$, $\text{deadline}_{m2}$ and $\text{deadline}_{m3}$, respectively. In the previous deadline constrained algorithms [2, 3, 8], r cannot receive $m_1$ and $m_2$ except for delivering $m_3$ in Figure 4. In order to receive as many messages as possible before their earliest deadline like $\text{deadline}_{m3}$, our proposed algorithm allows each member like p and q to keep received messages in its buffer, $\text{DLVD\_Q}_{rcvr}$ (by executing Module B-RECV(m, $\text{deadline}_m$, $\text{MVector}_{sndr}$) in Figure 8) like in Figure 5. If a member, r, receives a message, $m_3$, from s in this figure, it requests that m3's sender, s, gives m3's predecessors, m1 and m2, to itself by sending a solicitation message with m3's dependency vectors, $\text{MVector}_r$ and $\text{MVector}_s$, like in Figure 6 (by executing Module SOLICIT-RECV($\text{MVector}_{rcvr}$, $\text{MVector}_{upper}$) in Figure 8). After having obtained m1 and m2 from s, r can deliver all three messages to their corresponding applications in order (by executing Module RPY-RECV(*MSG_Q*) in Figure 8). To satisfy the deadline-constrained causal order requirement, our algorithm makes each member check deadline violation every time interval (by executing Module CHECK-MSGS() in Figure 8). In addition, even if some messages may be lost like in Figure 7, the group members not having received them can obtain from other members broadcasting their successors, and deliver them to the corresponding applications within the earliest deadline.

In order to secure the empty message buffer space, $\text{DLVD\_Q}_p$, as much as possible in our algorithm, every process p periodically performs the deadline-based garbage collection procedure for locally removing all the useless messages whose deadlines have passed from its buffer (by executing Module GARBAGE-COLLECT() in Figure 8). The procedure is scalable as it requires no extra synchronization with any other group members and no additional information piggybacked on each broadcast message. Although more complex garbage collection procedures may be designed using the dependency vector, $\text{MVector}_p$, they may require much larger piggybacked information having matrix-like structure to determine whether each message in the buffer has already been delivered to all the group members, which may considerably degrade scalability.

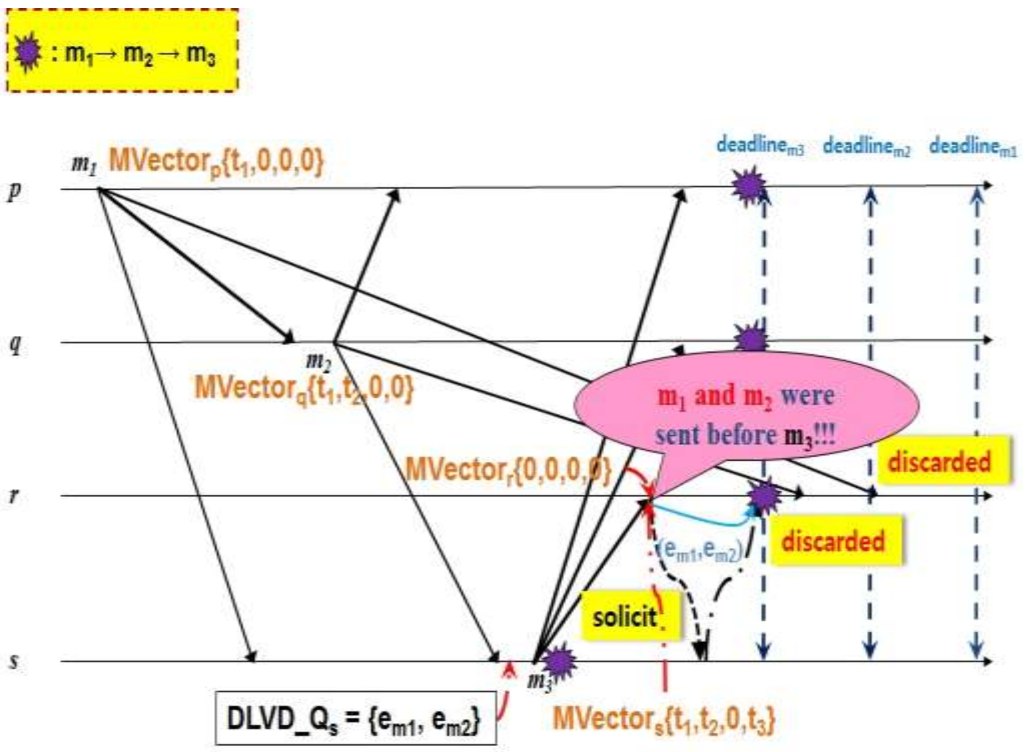

**Figure 5. Example Showing how our Algorithm Reduces Late Message Discarding Rate**

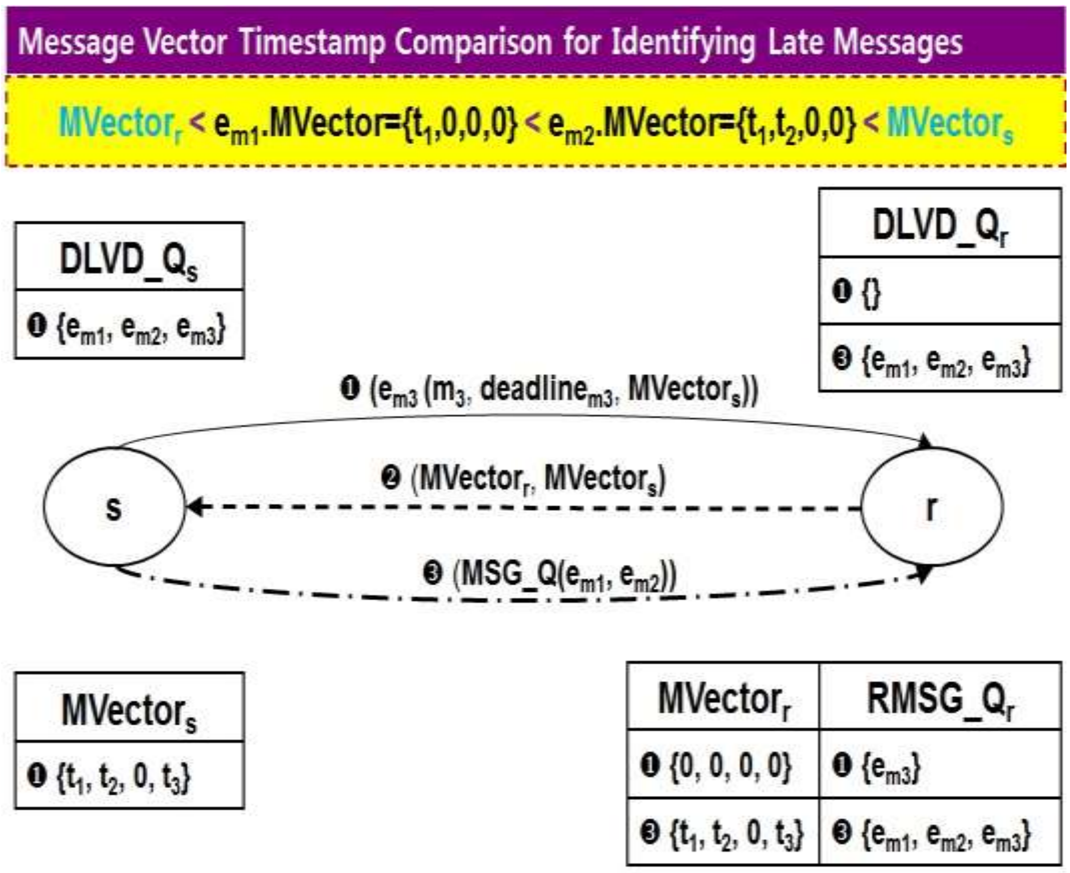

**Figure 6. Interaction Procedure of our Proposed Algorithm**

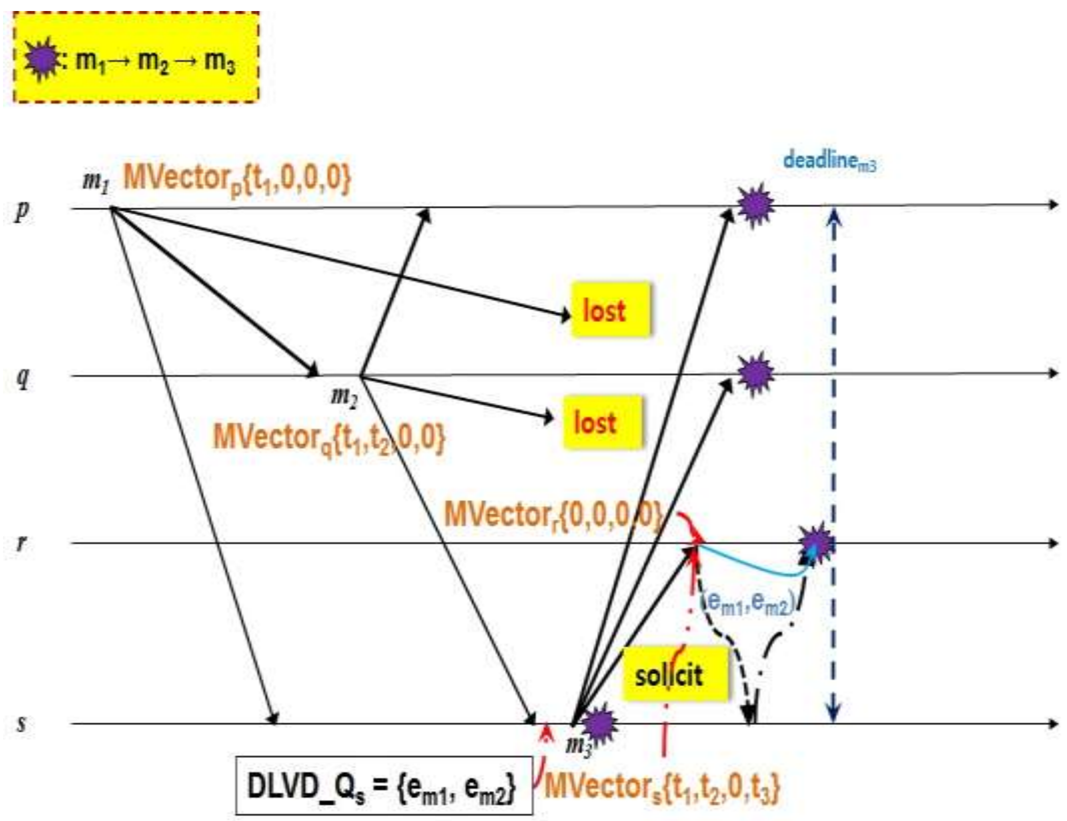

**Figure 7. Example Showing how our Algorithm Supports High Responsiveness even in Case of Message Loss**

```
// On sending a message m to group G at $P_{sndr}$
Module B-SEND(m, $deadline_m$)
  $MVector_{sndr}[sndr] \leftarrow$ the current clock value of $P_{sndr}$ ;
  broadcast a message m with ($deadline_m$, $MVector_{sndr}$) to all the other members $\in$ G;

// On receiving a message m broadcast to group G at $P_{rcvr}$
Module B-RECV(m, $deadline_m$, $MVector_{sndr}$)
  if(($deadline_m <$ the current clock value of $P_{rcvr}$ ) $\vee$ ($MVector_{sndr}[sndr] \leq MVector_{rcvr}[sndr]$))
  then
    discard message m from $P_{rcvr}$ ;
  else if(($MVector_{sndr}[sndr] > MVector_{rcvr}[sndr]$) $\wedge$
       ($\forall i \neq sndr \in G$: $MVector_{sndr}[i] \leq MVector_{rcvr}[i]$)) then
    $\forall i$: $MVector_{rcvr}[i] \leftarrow$ max($MVector_{rcvr}[i]$, $MVector_{sndr}[i]$) ;
    deliver m to its corresponding application ;
    insert (m, $deadline_m$, $MVector_{sndr}$) into $DLVD\_Q_{rcvr}$ in m's sending time order ;
    call CHECK-MSGS() ;
  else
    insert (m, $deadline_m$, $MVector_{sndr}$) into $RMSG\_Q_{rcvr}$ in m's sending time order ;
    send a message solicitation with ($MVector_{rcvr}$, $MVector_{sndr}$) to $P_{sndr}$ ;
```

**Figure 8. Modules for our Real-Time Constrained Broadcast Algorithm Ensuring Message Causality**

```
// Every time interval, the procedure is executed at $P_p$.
Module CHECK-MSGS()
  for $\forall e \in RMSG\_Q_p$ in FIFO order do
    if(e.m's sender j: (e.MVector[j]>$MVector_p$[j]) $\wedge$ ($\forall i \neq j$: e.MVector[i]$\leq$$MVector_p$[i])) then
      $\forall$i:$MVector_p$[i]$\leftarrow$ max($MVector_p$[i], e.MVector[i]) ;
      deliver e.m to its corresponding application ;
      insert a copy of e into $DLVD\_Q_p$ in e.m's sending time order ;
      remove e from $RMSG\_Q_p$ ;
    else if(e.deadline = the current clock value of $P_p$) then
      for $\forall c \in RMSG\_Q_p$ in FIFO order st (c.MVector $\leq$ e.MVector) do
        $\forall$i: $MVector_p$[i] $\leftarrow$ max($MVector_p$[i], c.MVector[i]) ;
        deliver c.m to its corresponding application ;
        insert a copy of c into $DLVD\_Q_p$ in c.m's sending time order ;
        remove c from $RMSG\_Q_p$ ;

// On receiving a message solicitation from $P_{rcvr}$ at $P_{sndr}$
Module SOLICIT-RECV($MVector_{rcvr}$, $MVector_{upper}$)
  MSG_Q $\leftarrow \Phi$ ;
  for $\forall e \in DLVD\_Q_{sndr}$ in FIFO order st (($\forall$i: e.MVector[i] < $MVector_{upper}$[i]) $\wedge$
    ($\forall$j: e.MVector[i] > $MVector_{rcvr}$[i])) do
    insert a copy of e into MSG_Q in e.m's sending time order ;
  send a message reply with (MSG_Q) to $P_{rcvr}$ ;

// On receiving a message reply at $P_{rcvr}$.
Module RPY-RECV(MSG_Q)
  $RMSG\_Q_{rcvr} \leftarrow RMSG\_Q_{rcvr} \cup MSG\_Q$ ;
  call CHECK-MSGS() ;

// Every time interval, the procedure is executed at $P_p$.
Module GARBAGE-COLLECT()
  for $\forall e \in DLVD\_Q_p$ in increasing deadline order st
    (e.deadline $\leq$ the current clock value of $P_p$) do
    remove e from $DLVD\_Q_p$ ;
```

**Figure 8. Modules for our Real-Time Constrained Broadcast Algorithm Ensuring Message Causality (continued)**

## 5. Performance Evaluation

In this simulation, we show the effectiveness of the algorithm (ODCA) proposed in this paper compared with the Traditional Deadline-Constrained causal order broadcast Algorithm (TDCA) embracing heterogeneous deadline imposition [13] using a discrete simulation language [1]. One performance indicator is used for comparison; the ratio of the number of delivered messages satisfying causal broadcast order and message deadline constraints together over the total number of sent messages ($Ratio_{delivered}$). A simulated system is composed of N nodes each associated with a coordinate(x, y). They are all interconnected with each other. Any two adjacent nodes are connected with a LAN link having a bandwidth of 100 Mbps with the propagation delay of 1ms. For simplicity of this simulation, we assume that both bandwidth and propagation delay between any pair of nodes are proportional to their distance. In our simulation environment, there are several important simulation parameters as follows. The first parameter is $Group_{size}$, the total number of processes created in the system and joining a process group. The target of each message sent from a process is always the process group. Thus, IP multicast is used for multicasting a message to a group of processes. Every process has a 128MB buffer space for storing messages needed due to transmission delay and message loss. Also, the size of each application message transmitted between any two processes ranges from 1KB to 100KB. In addition, messages to the process group are sent to the network with an interval following an exponential distribution with a mean $T_{ms}$=100ms. The network delay follows normal distribution where one way delay mean is 100ms and its loss rate is 0.05. We model the deadline of a message as a random variable with exponential distribution with a mean $T_{ms}$=500ms. All experimental results shown in this simulation are all averages over a number of trials.

Figure 9 shows $Ratio_{delivered}$ for the two algorithms, TDCA and ODCA, with varying $Group_{size}$, ranging from 16 to 100. As $Group_{size}$ increases in this figure, $Ratio_{delivered}$ values in the two algorithms fall down proportionally because the amount of the broadcasting traffic also becomes exponentially higher. However, the figure indicates that the $Ratio_{delivered}$ values of TDCA are much lower than those of ODCA. Especially when the value of $Group_{size}$ rises up greater than 36, the gap of $Ratio_{delivered}$ between the two algorithms becomes significantly larger. The rise of $Ratio_{delivered}$ of ODCA over TDCA ranges from 12.2% to 37.7%. This outcome arises from the reason that the large broadcast traffic and the message loss caused by the increase of $Group_{size}$ may considerably lower the probability with which group members can receive messages broadcast to them within their earliest deadline in TDCA. However, ODCA may stabilize $Ratio_{delivered}$ by greatly rising up the opportunity for group members to obtain the predecessors of each message they have received, but not delivered yet, from its sender before the deadline.

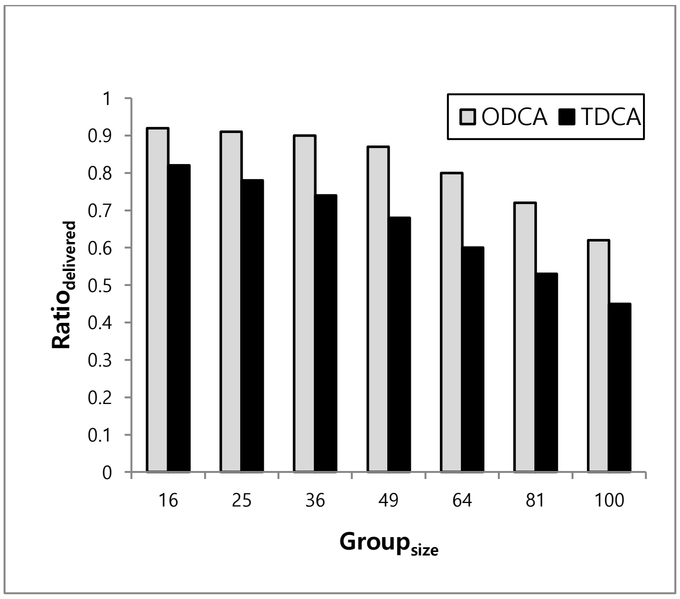

**Figure 9. Comparisons of $Ratio_{delivered}$ of the Two Algorithms with Varying Values of $Group_{size}$**

## 6. Conclusion

This paper presented effective deadline-constrained broadcast algorithm to ensure inter-message causality, highly reduce late message discarding rate and improve responsiveness. The algorithm makes every group member keep each message broadcast to its group on its volatile storage. With this behavior, it enables late messages preceding each received message to be given to its receiver from another group member as quickly before its deadline as possible. The simulation results show the proposed algorithm may be a lightweight solution to guarantee deadline-constrained causal order broadcast condition and greatly enhance the message delivery success ratio.

## Acknowledgment

This work was supported by Kyonggi University Research Grant 2015(2015-037).

## References

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

[2] R. Baldoni, "A Positive Acknowledgment Protocol for Causal Broadcasting", IEEE Transactions on Computers, vol. 47, no. 12, **(1998)**, pp. 1341-1350.

[3] R. Baldoni, A. Mostefaoui and M. Raynal, "Causal Delivery of Messages with Real-Time Data in unreliable Networks", Real-Time Systems Journal, vol. 10, no. 3, **(1996)**, pp. 245-262.

[4] R. Baldoni, R. Prakash, M. Raynal and M. Singhal, "Efficient Delta-Causal Broadcasting", Journal of Computer Systems Science and Engineering, vol. 13, no. 5, **(1998)**, pp. 263-270.

[5] K. Birman and T. Joseph, "Reliable Communication in the Presence of Failures", ACM Transactions on Computer Systems, vol. 5, no. 1, **(1987)**, pp. 47-76.

[6] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," Journal of the ACM vol. 43, **(1996)**, pp. 225-267.

[7] J. Fanchon, K. Drira and S. P. Hernandez, "Abstract channels as connectors for software components in group communication services", Computer Science, ENC 2004. Proceedings of the Fifth Mexican International Conference, **(2004)**, pp. 88-95.

[8] M. Fischer, N. Lynch and M. Merritt, "Easy Impossibility Proofs for Distributed Consensus Problems", PODC 1985, **(1985)**, pp. 59-70.

[9] C. Kim and J. Ahn, "Causal order multicast protocol using minimal message history information", the 12th international conference on Algorithms and Architectures for Parallel Processing, vol. 1, **(2012)**, pp. 546-559.

[10] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM, vol. 21, **(1978)**, pp. 558-565.

[11] C. Plesca, R. Grigoras, P. Queinnec, G. Padiou and J. Fanchon, "A coordination-level middleware for supporting flexible consistency in CSCW", 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, **(2006)**.

[12] S. E. Pomares Hernandez, E. Lopez Dominguez, G. Rodriguez Gomez and J. Fanchon, "An Efficient Δ-Causal Algorithm for Real-Time Distributed Systems", Journal of Applied Sciences, vol. 9, **(2009)**, pp. 1711-1718.

[13] L. Rodrigues, R. Baldoni, E. Anceaume and M. Raynal, "Deadline-constrained causal order", Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 234-241, **(2000)**.

[14] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: an approach to design-ing fault-tolerant distributed computing systems", ACM Transactions on Computer Systems, vol. 1, no. 3, **(1985)**, pp. 222-238.

## Authors

**Jinho Ahn**, he received his B.S., M.S. and Ph.D. degrees in Computer Science and En-gineering from Korea University, Republic of Korea, in 1997, 1999 and 2003, respectively. He has been a full professor in Department of Computer Science, Kyonggi University since 2003. 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.