Causally Ordered Message Delivery in Mobile Systems*

Sridhar Alagar and S. Venkatesan
Department of Computer Science
University of Texas at Dallas, Richardson, TX 75083
{sridhar,venky}@utdallas.edu

abstract

There is a growing trend in using mobile computing environment for several applications, and it is important that the mobile systems are provided adequate support both at the systems level and at the communication level. Causal ordering *is a useful property, particularly in applications that involve human interactions. In this paper, we present three algorithms for causal ordering in mobile systems. The first algorithm handles the resource constraints of the mobile hosts. But the system is not easily scalable and is not graceful to host disconnections and connections. The second algorithm eliminates the above disadvantages at the cost of inhibiting some messages. The third algorithm is a trade-off between the first two algorithms.*

1 Introduction

Many of the distributed algorithms designed for static hosts cannot be directly used for mobile systems due to the change in physical connectivity, resource constraints of mobile hosts and limited bandwidth of the wireless links [1]. This has spawned considerable amount of research in mobile computing: designing communication protocols [3, 9], file system operations [4], and providing fault tolerance [5]. In this paper, we consider the problem of providing a particular kind of communication support, namely, *causally ordered* message delivery to mobile hosts. Consider two messages m and m' sent to the same destination such that sending of m "happened before" sending of m'. Causal ordering is obeyed if m is received before m' is received.

Causal ordering is useful in several applications like management of replicated data, resource allocation, monitoring a distributed system, USENET etc., [7, 10]. Causal ordering is best suited for applications that involve human interactions from several locations [10]; such applications are typical in mobile systems. Some of the major applications of distributed mobile systems in which causal ordering is useful are teleconferencing, stock trading, collaborative applications, etc.

1.1 Motivation

While designing algorithms for mobile systems, the following factors should be taken into account.

F1. The amount of computation performed by a mobile host should be low.

F2. The communication overhead in the wireless medium should be minimal.

F3. Algorithms should be scalable with respect to the number of mobile hosts.

F4. Algorithms should be able to easily handle the effect of hosts disconnections and connections.

If mobile hosts are made to execute the traditional causal ordering algorithms (by storing the relevant data structures in the mobile hosts), none of the above factors (F1–F4) can be satisfied. We present three algorithms for causal ordering in mobile systems.

Our first algorithm stores the data structures of mobile hosts (MHs) relevant to causal ordering in the mobile support stations (MSSs), and the algorithm is executed by the MSSs on behalf of the MHs. However, the message overhead is proportional to the square of the number of mobile hosts. Thus, factor F3 is not satisfied. Also, the algorithm is graceful to hosts disconnections and connections.

Algorithm 2 eliminates the problems in Algorithm 1. The size of the message header is proportional to the square of the number of MSSs. Since the size

*This research was supported in part by NSF under Grant No. CCR-9110177, by the Texas Advanced Technology Program under Grant No. 9741-036, and by grants from Alcatel Network Systems.

of the header does not vary with the number of mobile hosts, the algorithm is scalable (with respect to the number of the mobile hosts) and host disconnections/connections do not pose any problem. But there may be some "inhibition" in delivering the messages to the mobile hosts. Our experimental results suggest that delay due to inhibition is less than the delay involved in transmitting and processing the long header (of each message) used in Algorithm 1. Also, the load placed on the MSSs is less compared to Algorithm 1.

Algorithm 3 is a hybrid algorithm and is a trade–off between algorithm 1 and algorithm 2. Every MSS is partioned into k logical MSSs to reduce the delay due to "inhibition" in delivering the messages to MHs. However, k cannot be large as this will increase the size of the header and hence the message overhead. A summary of our results is shown in Table 1.

2 Model and Definitions

A distributed mobile system consists of a set of *mobile hosts* and *static hosts*. A *mobile host* (MH) is a host whose geographical location can change with time while retaining its connectivity to the network [3]. A *static host* is a host whose location does not change during the computation. A static host can also be a *mobile support station* (MSS). An MSS has the necessary infrastructures to support the mobile hosts. For simplicity, we assume that the system consists of only MSSs and MHs. A static host can be considered as an MH that does not move.

The geographical area within which an MSS supports MHs is called a *cell*. Communication between MHs and MSSs is through a wireless channel. An MH can communicate directly with an MSS only if the MH is located in the cell of the MSS. A mobile host may belong to at most one cell at any time. Mobile hosts communicate with other hosts through their MSSs[†]. MSSs are connected among themselves using wired channels. The MSSs and the wired channels constitute the static network. We assume that a *logical* channel exists between every pair of MSSs. These logical channels need not be FIFO channels; whereas the wireless channels are FIFO. Both wired and wireless channels are reliable and take an arbitrary but finite amount of time to deliver messages.

A mobile host can migrate from one cell to another cell at any time. Every MSS periodically broadcasts a *beacon* [3]. Let MH h_1 move from the cell of MSS s_1 to the cell of s_2. h_1 discovers that it is in the cell of s_2 after receiving the *beacon* broadcast by s_2. MH h_1 informs MSS s_2 of h_1's *id* and the *id* of its previous MSS s_1. A *handoff* procedure is then executed between s_2 and s_1. s_2 informs s_1 about h_1's migration and gets the relevant information associated with h_1 from MSS s_1.

A mobile host can disconnect itself from the network by sending a *disconnect* message to its current MSS and can reconnect at a later time by sending a *connect* message. If an MSSs receives a message for any of the disconnected mobile hosts, the message can be stored and delivered to mobile host after it reconnects, or the message can be dropped depending on the application.

An event in a host may be a send event (sending a message to another host), a receive event (receiving a message from a host), or an internal event which does not involve sending or receiving a message. Let $send(m)$ be the event that corresponds to the sending of message m and $recv(m)$ be the event that corresponds to the receipt of m. Events in a mobile system are ordered based on the "happened before" relation, $\rightarrow$, introduced by Lamport [6]. For any two events e and e', $e \rightarrow e'$ is true if (i) e and e' are two events in the same host and e occurs before e' or (ii) e corresponds to sending a message m and e' corresponds to the receipt of m or (iii) there exists an event e'' such that $e \rightarrow e''$ and $e'' \rightarrow e'$. *Causal ordering* of message delivery is obeyed if, for any two messages m and m' that have the same destination, $send(m) \rightarrow send(m')$ implies that $recv(m) \rightarrow recv(m')$.

3 Preliminaries

Causal ordering was first proposed for the ISIS system [2]. There are several algorithms that implement causal ordering for distributed systems with static hosts [2, 7, 8]. The algorithm by Birman and Joseph [2] appends, to every message, the history of the communications that happened before the sending of the message. The size of the appended information can become unbounded. However, the channels need not be reliable. The algorithm by Raynal, Schiper and Toueg, referred henceforth as RST algorithm, is based on message counting and assumes the channels to be reliable [7]. The RST algorithm, which we will discuss subsequently, appends N^2 integers to every message, where N is the number of hosts in the system. The algorithm by Schiper et al. [8] uses vector clocks and is somewhat similar to the RST algorithm. In this paper we extend the RST algorithm to mobile systems.

[†]The MSS of an MH is the MSS in whose cell the MH is located.

Algorithm	Size of message header	"handoff" Complexity	"inhibition"
Algorithm 1	$O(n_h^2)$ integers	$O(1)$ messages	No
Algorithm 2	$O(n_s^2)$ integers	$O(n_s)$ messages	Yes
Algorithm 3	$O(k^2 * n_s^2)$ integers	$O(k * n_s)$ messages	decreases with k

Table 1: n_h is the number of MHs, n_s is the number of MSSs, and k is the number of logical MSSs per MSS. $n_h \gg n_s$.

The RST algorithm for causal ordering maintains two arrays, $\text{DELIV}_i[N]$ and $\text{SENT}_i[N, N]$, for each host P_i. $\text{DELIV}_i[j]$ denotes the total number of messages received by P_i from P_j. $\text{SENT}_i[k, j]$ indicates P_i's knowledge about the number of messages P_k has sent to P_j. The following steps are executed at P_i to ensure causal ordering.

When P_i sends message m to P_j, P_i appends its current value of SENT_i with m. (P_i sends (m, SENT_i) to P_j.) P_i then increments $\text{SENT}_i[i, j]$ by 1.

On receiving (m, ST) from P_j, the causal ordering algorithm at P_i first checks if $\text{DELIV}_i[k] \geq ST[k, i]$ for all k. If so, the message m is delivered to the application, $\text{DELIV}_i[j]$ is incremented by 1, $\text{SENT}_i[j, i]$ is set to $ST[j, i] + 1$, and finally $\text{SENT}_i[j, k]$ is set to $max(ST[j, k], \text{SENT}_i[j, k])$ for all j, k. If not, m is queued till $\text{DELIV}_i[k] \geq ST[k, i]$ for all k.

4 Algorithm 1

Algorithm 1 consists of two modules: static module and handoff module. The static module is executed when an MH is in a particular cell. The handoff module is executed when an MH moves from one cell to another.

4.1 Static Module

For each MH h_i, we maintain two arrays– $\text{MH\_DELIV}_i[n_h]$ and $\text{MH\_SENT}_i[n_h, n_h]$, where n_h is the number of mobile hosts. $\text{MH\_DELIV}_i[j]$ denotes the total number of messages received by h_i from h_j. $\text{MH\_SENT}_i[k, j]$ indicates h_i's knowledge about the number of messages h_k has sent to h_j. Assume that MH h_i is in the cell of MSS s_k. To reduce the communication and computation overhead of MH h_i, these arrays are stored in MSS s_k. Since the messages from (to) h_i go through MSS s_k, the causal ordering algorithm is executed by MSS s_k.

Initially, all the entries in the arrays MH_DELIV_i and MH_SENT_i are set to 0. To send a message m to another MH h_j, h_i first sends the message m to its MSS s_k. s_k sends $(m, \text{MH\_SENT}_i)$ to the MSS of h_j and increments $\text{MH\_SENT}_i[i, j]$. There are several protocols [3, 9] that ensure reliable message delivery to mobile hosts. Any of these protocols can be used.

MSS s_k, on receiving a message (m, ST) meant for h_i from MH h_j, first checks whether m is *deliverable*. m is deliverable if $\text{MH\_DELIV}_i[k] \geq ST[k, i]$ for all k. If so, s_k transmits m to h_i, increments $\text{MH\_DELIV}_i[j]$, and $\text{MH\_SENT}_i[j, k]$ is set to $max(ST[j, k], \text{MH\_SENT}_i[j, k])$ for all j, k. m is also temporarily stored by s_k in PEND_ACK_i. Message m will be deleted from PEND_ACK_i after receiving an ack for m from MH h_i. If m is not deliverable, m is stored in MH_PENDING_i till m becomes deliverable. Whenever a message is delivered to h_i, s_k checks MH_PENDING_i for any message that becomes deliverable.

4.2 Handoff Module

Let h_i move from the cell of MSS s_k to the cell of MSS s_t. The handoff module is then executed by s_k and s_t. After entering the cell of s_t, MH h_i sends the message $register(h_i, s_k)$ to s_t. Also, h_i retransmits the messages (to s_t) for which it did not receive ack from its previous MSS s_k. MSS s_t then informs s_k that h_i has switched from MSS s_k to MSS s_t by sending a $handoff\_begin(h_i)$ message to s_k. After receiving $handoff\_begin(h_i)$, s_k transfers MH_DELIV_i, MH_SENT_i, MH_PENDING_i, and PEND_ACK_i to MSS s_t and finally sends message $handoff\_over(h_i)$ to s_t.

On receiving these data structures, s_t first transmits all messages in PEND_ACK_i. Also, s_t forwards the messages (to their destinations) retransmitted by h_i. The handoff procedure is then terminated at s_t. If MH h_i switches to some other cell before the handoff is completed, the current handoff is completed before a new handoff begins.

4.3 Analysis

For every message sent by MH h_i, the MSS (in whose cell h_i resides) sends MH_SENT_i with the mes-

sage. Hence, the size of the header for every message sent over the static network is $O(n_h^2)$ integers. The handoff module uses $O(1)$ messages of size $O(n_h^2)$ numbers when MH h_i switches its cell.

Now, consider the factors F1–F4 discussed in Section 1.1. Since Algorithm 1 is executed at MSSs, factors F1 and F2 are satisfied. The overhead in the wireless medium is kept minimal. But factors F3–F4 are not satisfied. An overhead of $O(n_h^2)$ integers over the static network is costly if n_h is very large. Also, due to disconnections and connections, n_h varies. So during disconnections, some of the entries in the arrays MH_DELIV, and MH_SENT are not needed. The arrays need not be static, but maintaining dynamic arrays can become complicated if the MH disconnections and connections are frequent. In addition, the processing time for updating the matrix MH_SENT will be substantial for large n_h, and the nontrivial processing time increases the delay in delivering a message.

5 Algorithm 2

In Algorithm 1, messages are tagged with complete information to explicitly maintain causal ordering among the mobile hosts. In Algorithm 2, messages are tagged with sufficient information just to maintain causal ordering among the MSSs. Since the wireless channel between an MSS and an MH in its cell is FIFO, maintaining causal ordering at the static network level is sufficient if the MHs do not move. To ensure that causal ordering is not violated after an MH moves, we incorporate some steps into the handoff procedure.

5.1 Static Module

The static module is similar to the static module of Algorithm 1 but for some of the data structures. For each MSS s_i, we maintain $\text{MSS\_DELIV}_i[n_s]$, $\text{MSS\_SENT}_i[n_s, n_s]$, and MSS_PENDING_i. (This is unlike in Algorithm 1 where we maintain these data structures for every mobile host.) Observe that the size of the arrays $\text{MSS\_DELIV}_i[n_s]$ and $\text{MSS\_SENT}_i[n_s, n_s]$ vary with n_s, the number of MSSs. The value of $\text{MSS\_DELIV}_i[j]$ indicates the number of messages (whose destination can be different MHs) received from MSS s_j by MSS s_i. $\text{MSS\_SENT}_i[k, j]$ denotes the number of messages sent by MSS s_k (not necessarily delivered) to MSS s_j that s_i knows of. Every MSS knows (need not be exact) about the location of the MHs. Initially, we assume that the initial locations of MHs are known to all MSSs. We show how this knowledge gets updated in the next section. In other aspects, the static module is similar to the static module of algorithm 1.

5.2 Handoff Module

The handoff module is more involved when compared to the handoff module of algorithm 1. Since causal ordering is explicitly maintained only at the MSSs level, some measures have to be taken during handoff to maintain causal ordering after an MH moves.

Before we describe the handoff module, we illustrate the problem at hand with an example. Consider mobile hosts h_1, h_2, and h_3. Assume that h_1, h_2 and h_3 are in the cells of MSSs s_1, s_2 and s_3 respectively. Let h_3 send a message m_1 to h_1 (m_1 will be sent to MSS s_1) and then send a message m_2 to h_2. Before receiving m_1, let h_1 switch to the cell of s_2. Now, MH h_2, after receiving m_2 from h_3, sends a message m_3 for h_1 to s_2. If s_2 delivers m_3 to h_1, causal ordering will be violated because h_1 has not yet received m_1. Also, s_2 cannot find out from the knowledge it has gained so far whether there are any in–transit messages for h_1 sent to s_1. However, if s_2 delivers m_3 after ascertaining that all the messages for h_1 sent to s_1 have been delivered, causal ordering will not be violated. Now, we describe the handoff module.

Assume that a mobile host h_k switches from the cell of MSS s_i to the cell of MSS s_j. After switching, MH h_k sends $register(h_k, s_i)$ message to s_j. On receiving this message, s_j sends the message $handoff\_begin(h_k)$ to s_j, and then broadcasts the message $notify(h_k, s_i, s_j)$ to all the MSSs. The message $notify(h_k, s_i, s_j)$ signifies that MH h_k has switched from MSS s_i to MSS s_j. An MSS s, on receiving $notify(h_k, s_i, s_j)$ message, updates its local knowledge about the location of MH h_k and sends a $last(h_k)$ message to s_i. After receiving $notify(h_k, s_i, s_j)$, MSS s will send messages meant for MH h_k only to s_j (the new MSS of h_k) and not to s_i (the previous MSS of h_k). MSS s_i, after receiving the message $handoff\_begin(h_k)$ from s_j, sends $enable(h_k, \text{PEND\_ACK}_k)$ message to s_j and waits for $last(h_k)$ messages from all the MSSs. Meanwhile, if any message received by s_i for h_k becomes deliverable to h_k, s_i marks it as *old* and forwards it to s_j.

On receiving the message $enable(h_k, \text{PEND\_ACK}_k)$ MSS s_j starts sending the application messages sent by h_k. Also, s_j delivers all the messages in PEND_ACK_k in the FIFO order to MH h_k. s_j also delivers all the messages for MH h_k that are marked *old* to h_k in the order in which the messages arrived. Any mes-

sages for h_k that are not marked *old* will be queued in MSS_PENDING$_j$.

MSS s_i (the previous MSS of h_k), after receiving *last*(h_k) from all the MSSs sends the message *handoff_over*(h_k) to MSS s_j. Observe that no messages for h_k sent to s_i will be in transition after s_i receives *last*(h_k) from all the MSSs. (Messages sent as part of handoff module are also causally ordered.) The handoff terminates at s_j after *handoff_over*(h_k) is received by s_j. If s_j receives the message *handoff_begin*(h_k) from some other MSS before the current handoff terminates (this can happen if h_k switches its cell), s_j will respond to the message only after the handoff terminates.

5.3 Analysis

The size of MSS_SENT is n_s^2 integers and hence the size of each message header over the wired network is $O(n_s^2)$ integers. The overhead does not depend on n_h, the number of MHs. Clearly, factors F3–F4 are satisfied. MH connections/disconnections do not affect the size of the arrays MSS_DELIV and MSS_SENT. During handoff, a *notify* message has to be sent to all the MSSs, and all the MSSs send *last* messages. Hence, the handoff module uses $O(n_s)$ messages. The storage requirement of Algorithm 2 and the load placed on the MSSs are less than that of Algorithm 1.

Though the handoff module is involved, it does not affect the performance (compared to Algorithm 1) due to the following reasons. (i) MH h_k does not wait for the handoff module to terminate to receive messages. It keeps receiving *old* messages. (ii) Messages sent by h_k for other MHs are sent by s_j (the new MSS of h_k) immediately after s_j receives *enable* message.

The drawback of Algorithm 2 is the possibility of a message being "inhibited" from being delivered to an MH. There is an *inhibition* in delivering a message to an MH if it is queued in MSS_PENDING even though the delivery of the message does not violate causal ordering. Messages may be inhibited because, in Algorithm 2, causal ordering is explicitly implemented among the MSSs. Reception of a message may violate causal ordering from an MSS's point of view; whereas its delivery to an MH may not violate causal ordering from the MH's point of view. However, this delay is less than the delay introduced by Algorithm 1 in transmitting and processing the header of each message. The average delay in delivering a message in Algorithm 2 is considerably less than the delay in Algorithm 1 when n_h increases, as shown in Figure 1. (For the details of our simulation model, see Appendix A.) When $n_h < 30$ the message header in both the algorithms are comparable in size. The message delay in Algorithm 2 is more than that of Algorithm 1 due to the inhibition inherent in Algorithm 2. However, as n_h increases the delay due to processing the message header in Algorithm 1 dominates.

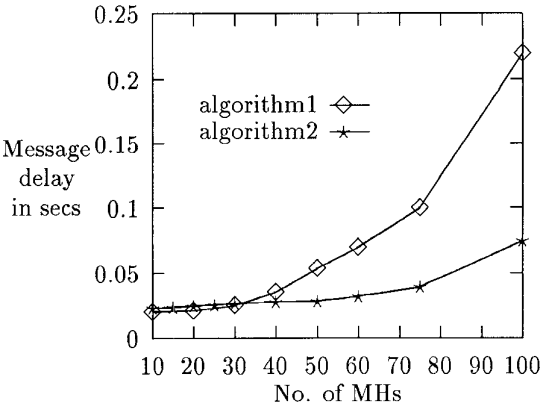

Figure 1: Comparison of Algorithm 1 and Algorithm 2 with respect to message delay. $n_s = 10$

6 Algorithm 3

This algorithm reduces the delay in delivering the messages to MH due to inhibition, the drawback of Algorithm 2, without much increase in the message overhead. The algorithm achieves this by partioning every physical MSS into k logical MSSs.

If an MH enters the cell of an MSS, the MH will be allocated to one of the logical MSSs depending on the load in each logical MSS of the MSS. The MHs will communicate with the other MHs through their logical MSSs. Every logical MSS maintains two arrays MSS_DELIV$[k * n_s]$ and MSS_SENT$[k * n_s, k * n_s]$ and a queue MSS_PENDING. The algorithm is the same as Algorithm 2 except for the fact that causal ordering is explicitly maintained among the logical MSSs. The size of the message header is $O(k^2 * n_s^2)$.

Messages to MHs that belong different logical MSSs will not inhibit each other though the MHs may be in the same cell. Thus, as k increases, the unnecessary delay in delivering the message to MH decreases. However, as k increases the size of the message header will increase and, as a result, the time to process the message header will become a dominating factor. In Figure 2, the average message delay initially decreases

when k increases. But when k becomes large the average message delay increases.

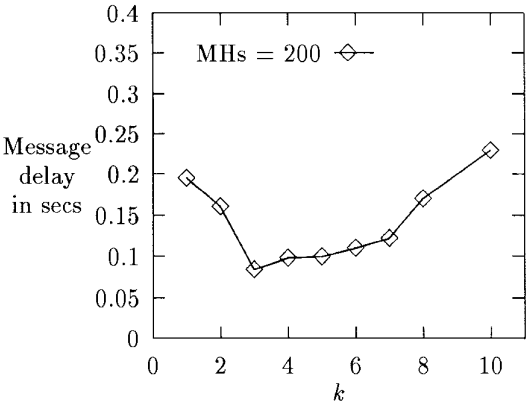

Figure 2: Message delays for various values of k. $n_s = 10$.

References

[1] BADRINATH, B., ACHARYA, A., AND IMIELINSKI, T. Impact of mobility on distributed computations. *ACM Operating Systems Review 27*, 2 (April 1993).

[2] BIRMAN, K., AND JOSEPH, T. Reliable communications in presence of failures. *ACM Trans. Comput. Syst. 5*, 1 (1987), 47–76.

[3] IOANNIDIS, J., DUCHAMP, D., AND MAGUIRE, G. IP-based protocols for mobile internetworking. In *Proceedings of ACM SIGCOMM Symposium on Communication Architecture and Protocols* (1991), pp. 235–245.

[4] KISTLER, J., AND SATYANARAYANA, M. Disconnected operation in coda file system. *ACM Trans. Comput. Syst. 10*, 1 (February 1992).

[5] KRISHNA, P., VAIDYA, N., AND PRADHAN, D. Recovery in distributed mobile environments. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems* (1993), pp. 83–88.

[6] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (1978), 558–565.

[7] RAYNAL, M., SCHIPER, A., AND TOUEG, S. Causal ordering abstraction and a simple way to implement it. *Inf. Process. Lett. 39*, 6 (1991), 343–350.

[8] SCHIPER, A., EGGLI, J., AND SANDOZ, A. A new algorithm to implement causal ordering. In *Proceedings of the 3rd International Workshop on Distributed Algorithms* (1989), Springer Verlag, pp. 219–232.

[9] TERAOKA, F., YOKOTE, Y., AND TOKORO, M. A network architecture providing host migration transparency. In *Proceedings of ACM SIGCOMM* (September 1991).

[10] VAN RENESSE, R. Causal controversy at le mont st.-michel. *ACM Operating Systems Review 27*, 2 (April 1993), 44–53.

A Simulation Details

Our simulation model is similar to that of in [5]. The simulation is event driven and it is run on a Sparc 10 station. The events are send message, receive message, and handoff. The bandwidth of a wired channel is assumed to be 100 Mbits/sec, and the propagation delay in a wired channel is 7 ms. For a wireless channel, the bandwidth and propagation delay are assumed to be 1 Mbits/sec and 500 μs, respectively.

Initially, the cells of the mobile hosts are assigned randomly. The time interval between two send events in a mobile host is an exponentially distributed random variable with a mean of t_s seconds. The time interval between handoff is also an exponentially distributed random variable with a mean of t_h seconds. The values of t_s and t_h are varied (0.1, 1.0, 10 secs) to consider different scenario of communication and mobility. The processing time considered in measuring the message delay is the actual CPU running time in processing the message header. The value of every point in the graph is an average of the results of 1000 experiments performed.