A Scalable Scheme for Causal Message Ordering

Li-Hsing Yen
Dept. of CSIE
Chung Hua Univ.
lhyen@chu.edu.tw

Kuang-Hwei Chi
Dept. of Electrical Engr.
Nat'l Yunlin Univ. of Sci. and Tech.
chikh@yuntech.edu.tw

Ting-Lu Huang
Dept. of CSIE
Nat'l Chiao Tung Univ.
tlhuang@csie.nctu.edu.tw

***Abstract*—This paper presents a scalable scheme for ensuring causal ordering of messages passing among processes in large-scale distributed systems. Previously proposed approaches, categorized as centralized or fully distributed, either place the entire processing loads on a single process or incur quadratic message overheads in the number of participating processes. These solutions perform limitedly in large-scale systems. Our scheme organizes the entire system as hierarchical clusters in which any of the previously proposed approaches can be employed. Message causality is maintained by enforcing the rules by which messages are propagated from origins to destinations. This approach incurs much less processing load on hot-spot sites than the centralized approach, or, alternatively, requires a message space overhead much less than that in the fully distributed approach. We shall show that, by setting cluster size appropriately, the message space overhead can be only a linear or even a logarithmic function of the number of processes involved. Our approach is suitable for large non-proprietary networks.**

Key words: causal ordering, distributed algorithms, distributed systems, message delivery

I. INTRODUCTION

The nondeterministic nature of distributed systems, *i.e.*, asynchronous process execution speeds and unpredictable communication delays, is the major factor that complicates the design, verification, and analysis of distributed systems. *Causal message ordering*, henceforth referred to as CMO, is an ordering imposed on message deliveries to reduce system nondeterminism while retaining concurrency. In systems preserving CMO, messages directed to the same destination are delivered in an order consistent with their potential causality. The causality under consideration is determined by the *happens-before* relation [25] but is restricted to message sending and receiving events. Specifically, if a message-sending event happens before the sending of another message, the former message is considered to have the potential for affecting the latter in a causal way, and therefore must be received before the latter to retain their cause-effect relationship, if they are destined for the same process. In asynchronous distributed systems, it may be difficult to ensure CMO since processes continue their computation and communication activities after they issue messages, and message delays are arbitrary.

CMO is considered important to reliable distributed systems [6], [8], [10], [27]. It can be used to maintain the consistency of replicated data located at different sites [23], [13], observe behaviors of a distributed systems [29], [33], simplify the design of distributed algorithms [1], [4], [6], and preserve semantic causality in news or teleconference applications. Many implementations and extensions of causally ordered communication have been done in distributed shared memory systems [3], multimedia systems [2], [7], and mobile computing systems [5], [18], [28], [35]. Stoller and Schneider [32] formulated a Hoare-style proof system for the verification of algorithms that exploit CMO as their communication primitives. Yen [34] analyzed the probability of breaking CMO by assuming some random distributions of message delays.

Conventional CMO solutions are either centralized or fully distributed. In centralized approaches [12], [24], [30], a dedicated coordinator process serializes all messages exchanged in the system, effectively imposing a total order on delivery that is consistent with the causal order. In distributed approaches [8], [9], [29], [31], each process can send messages directly to any others without the intervention of a coordinator. The distributed approach may use piggybacking technique [8], where each message carries a history copy of all causally prior messages. Thus when a message m is delivered to a process P, copies of all messages addressed to P that causally precede m also arrive with m or have arrived earlier. Alternatively, the distributed approach may exploit vector clocks [20], [26] or similar mechanisms [19] so that, instead of the contents of all causally preceding messages, only vector clock timestamps of causally preceding messages need to be carried in each message [9], [29], [31]. The basic idea is to deliver message m to process P only if all messages that causally preceded m and were destined for P have already been delivered. Otherwise, message m should be buffered until the delivery condition stated above is satisfied.

None of these conventional approaches scales well. The centralized approach creates a performance bottleneck at the coordinator, resulting in performance degradation especially when message-exchanges are frequent. This drawback makes cen-

tralized approaches unsuitable for large-scale systems. The piggybacking approach may either suffer from unbounded growth of the information added to messages, or require a complex mechanism to prevent it. Fully distributed approaches exploiting vector clocks, on the other hand, impose a size of $O(n^2)$ message header on every message (n is the number of participating processes), which has been proven necessary for system-wide CMO [2][1] This overhead becomes intolerable when n grows large.

In fact, one of the reasons that some researchers criticized about CMO is that observed solutions do not scale [17]. To cope with this limitation, in this paper we propose a CMO scheme that unifies conventional approaches for large-scale distributed systems. Our scheme organizes the system into hierarchical *clusters*. Within a cluster, any existing CMO methods can be locally adopted. Our approach effectively decomposes system-wide CMO into a number of independent cluster-wide CMOs. The correctness is guaranteed by confining the way in which messages are propagated. The merit is that the heavy work load on the coordinator that the centralized approach imposes is distributed, or, alternatively, the costly message space overhead that a distributed approach imposes is decreased significantly.

The rest of this paper is organized as follows. Section 2 gives some definitions and assumptions concerning the problem. Section 3 describes details of the proposed method and also proves its correctness. In Section 4, we analyze the effects of clustering on reducing message header space. Section 5 concludes the paper.

II. Preliminary

Our scheme assumes an asynchronous distributed system consisting of n processes. A process communicates with others solely by means of message-passing. No shared memory or global clock is available. The communication channel between a pair of processes is assumed to be logically reliable. Message transmission delays are arbitrary but finite.

An event is defined as an atomic operation that changes the state of a process. Three types of events may occur in distributed systems: the *sending* of messages, the *receipt* of messages, and internal events [26]. What constitutes an internal event depends on the context of the system and is irrelevant to the definition of message causality. The happens-before relation (denoted by "$\rightarrow$") on the set of events is the smallest transitive relation satisfying the following conditions [25]:

- if a and b occur in the same process and if a comes before b then $a \rightarrow b$;
- if a is the sending of message m and b is the receipt of m, then $a \rightarrow b$.

Conventional CMO approaches can be abstracted as providing every process with a causal message delivery part (CMD) between the application process and underlying communication network. The receipt of a message is thus differentiated from the delivery of the same message. A message is said to be *received* by a site when it arrives at the CMD of that site, and is said to be *delivered* to a site when it is passed by the CMD to the application process without violating causal order. Let *sent*(m) and *deliv*(m), respectively, denote the events of sending and delivering message m. Conventional schemes as well as ours aim to ensure CMO with respect to sending and delivering events, *i.e.*, *sent*$(m) \rightarrow$ *sent*(m') always implies *deliv*$(m) \rightarrow$ *deliv*(m') for two messages m and m' addressed to the same destination.

III. The Proposed Scheme

The proposed scheme consists of two parts. The first part states how processes are organized. The second part describes how messages are propagated from sources to destinations. We shall first present a primitive two-layer hierarchy and then extend it to a more complex structure.

A. A Primitive Two-Layer Hierarchy

Process Organization

The set of all processes is partitioned into a number of subsets called clusters. Processes in the same cluster is allowed to send messages directly (*i.e.*, without any intermediation of any other process) to each other. Within each cluster a specific process is designated as the *agent* of the cluster, and all others are referred to as *clients*. All messages into or out of a cluster must be queued and served serially at the agent. The set of all agents forms a special cluster, the *agent cluster*, which has no agent for it. How processes are clustered is irrelevant to the correctness of our scheme. A straightforward approach is to assign computationally coherent processes to the same cluster. Alternatively, processes can be clustered to match the underlying communication or administration structure of existing network.

Message Ordering and Propagation

We extended the abstraction of conventional approaches by introducing at every site a message relay part (MR) between the application process and the CMD. MR takes charge in relaying messages between clusters. When a message is issued by an application process, it is first passed to the MR at the same site to determine the next stop of the messages. The determination rule is as follows. An intra-cluster message will be sent directly to its destination, whereas an inter-cluster message will be first directed to the origin's agent, then toward the destination's agent, and finally to the destination process. Each relay of this message corresponds to an intra-cluster message propagation between two MRs. Since every cluster employs its

[1]The message overhead of some multicast (broadcast) protocols can be degenerated to $O(n)$. For such examples, we refer the reader to [14], [11]. Observe that a multicast can be achieved by multiple unicasts but not *vice versa*. The reduction of complexity essentially results from abbreviating redundant, repeatedly identical causality information of messages in the unicast-type protocol. This paper is primarily concerned with the point-to-point communication paradigm, a primitive.

own CMO protocol, the relay is accomplished by the locally adopted cluster-wide CMO protocol. An agent, which involves in communications between two clusters, must maintain two independent CMD processes, one for each cluster. At such an agent, an incoming message is first received by the corresponding CMD process. Upon delivery respecting CMO, the message is then passed upward to the MR. If the delivered message is destined for this site, it is passed further to the application process; otherwise, the MR relays the message through the other CMD process. A typical message propagation scenario is illustrated in Figure 1.

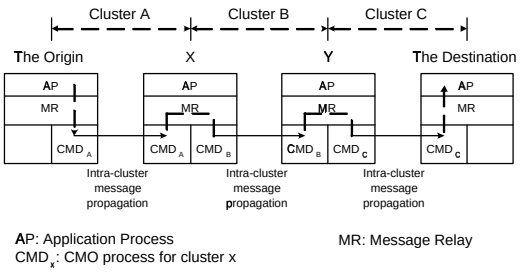

Fig. 1. Message propagation example

Note that the relaying between two CMD processes must be done in FIFO order. Consider Y in Figure 1 as an example. Let m and m' be two messages both from cluster B. If m is delivered by *CMD*$_B$ to Y's MR before m' is, the MR must send m through *CMD*$_C$ before m'. For message relayed in the opposite direction, *i.e.,* from clusters C to B, the FIFO order requirement must be met as well.

B. Hyper-Clustered Hierarchy

The two-layer hierarchy can be extended, considering the fact that the agent cluster forms a smaller subsystem that can be further decomposed by recursively applying the same clustering rule. Such an extension yields a hyper-clustered hierarchy.

Definition 1: Consider system $\mathcal{S}$. Let the primitive set of processes be at the first layer and let the set of processes at the i-th layer be denoted by $\mathcal{S}_i$. $\mathcal{S}$ forms an l-layer hierarchy ($l \geq 2$) if for all i, $1 \leq i < l$, the following conditions hold.

- $\mathcal{S}_i$ is partitioned into m_i clusters, $\mathcal{S}_i^1, \mathcal{S}_i^2, \ldots, \mathcal{S}_i^{m_i}$, where $1 < m_i < |\mathcal{S}_i|$.
- Let A_i^j denote the agent of cluster $\mathcal{S}_i^j$ ($1 \leq j \leq m_i$). $\mathcal{S}_{i+1} = \{A_i^1, A_i^2, \ldots, A_i^{m_i}\}$.

According to the definition, there is only one cluster on the topmost layer which has no agent for itself. Figure 2 shows an example of a three-layer hierarchy. Note that some process participating in two or more clusters may act as an agent on some layer as well as a client on a higher layer. As an example, process R in Figure 2 involves in communications among three clusters, one on each layer, and acts as a client on the third layer.

A site participating in k ($k \geq 2$) clusters now must maintain k independent CMD processes, one

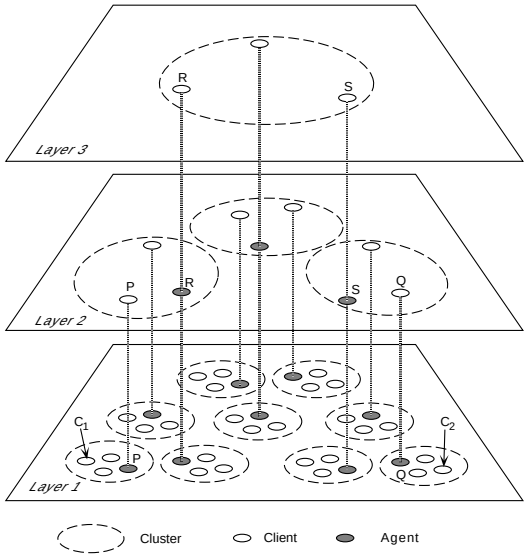

Fig. 2. An example of a three-layer hierarchy

for each cluster. This results in total $k(k-1)$ possible directions along which messages may be relayed from one CMD process to another. It is required that MR must relay all messages along the same direction in FIFO order.

The message relaying rule is generalized for an l-layer hierarchy ($l \geq 2$). Figure 3 shows a procedure, MRP, that computes the complete sequence of sites through which to propagate a message with known source and destination sites. The relaying path for messages from site S_i to site S_j is obtained by invoking *MRP*(S_i, S_j, 1). The output is a sequence of $\langle$*sender*, *receiver*$\rangle$ tuples, which, in order, specifies each link of that path.

---

procedure *MRP*(S_i, S_j, k)
  if S_i and S_j are in the same cluster on layer k **then**
    output $\langle S_i, S_j \rangle$
  else
    Let A_i be the agent of S_i on layer k
    Let A_j be the agent of S_j on layer k
    if $S_i \not\equiv A_i$ **then**
      output $\langle S_i, A_i \rangle$
    end if
    invoke *MRP*$(A_i, A_j, k+1)$
    /* to generate the path from A_i to A_j */
    if $S_j \not\equiv A_j$ **then**
      output $\langle A_j, S_j \rangle$
    end if
  end if
end procedure

Fig. 3. Procedure Message Routing Path (MRP)

C. CMD Implementations

Adopting Raynal-Schiper-Toueg Algorithm

The first implementation we consider is to adopt in every cluster the algorithm proposed by Raynal, Schiper, and Toueg [29] (henceforth referred to as the RST algorithm). In this algorithm, each process

P_i maintains an $n \times n$ matrix, $SENT_i$, where n is the total number of processes, to record the number of messages, as it has known, sent from each process to each others. Every message transmitted by P_i is tagged with the contents of $SENT_i$. Each process P_i also maintains an n-entry vector, $DELIV_i$, to record the number of messages delivered to P_i from all others. On receiving a message, say m, process P_i can determine whether m can be delivered by comparing $DELIV_i$ with the i-th column of the *SENT* matrix tagging m. If m can be delivered, $SENT_i$ as well as $DELIV_i$ are updated, and the *SENT* matrix tagging m can be discarded.

Adopting the RST algorithm as an implementation of CMD, our scheme can be viewed as a way of clustering the RST algorithm. Each CMD process maintains its own *SENT* matrix and *DELIV* vector. The sizes of the matrix and the vector depend on the size of the cluster which the CMD process belongs to. When a message is delivered by a CMD, the tagged *SENT* matrix will be discarded before the message is passed to the MR. On the other hand, when the MR passes a message through a CMD to send to a cluster, the CMD will tag the message with the *SENT* matrix maintained by that CMD. Consequently, when an inter-cluster message is in propagation, it is only tagged with the *SENT* matrix corresponding to the current cluster. The *SENT* matrix corresponding to the previous cluster has been discarded when the message left that cluster.

Adopting the Centralized Approach

If the centralized approach is adopted in every cluster as an implementation of CMD, a coordinator must be nominated in each cluster. A simple strategy is to incorporate the coordinator's functionality in every agent, and to nominate a process as the coordinator at the top layer since the top layer, by our definition, has no agent for itself. When combining this strategy with the FIFO requirement on message relaying, message propagation can be modeled as a propagation tree similar to those proposed for total-ordering multicast [21], [22]. Figure 4 shows the propagation tree that corresponds to the three-layer hierarchy shown in Figure 2.

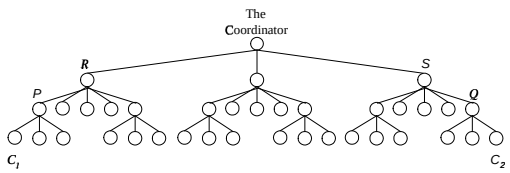

Fig. 4. The propagation tree corresponding to the hierarchy shown in Figure 2

It is also possible to adopt other CMO algorithms, and to even use different approaches for each cluster. Various degrees of modifications may be needed in adopting these approaches.

D. Correctness Justification

We shall justify that our mechanism ensures system-wide CMO. The key to the correctness is that whenever *sent*$(m) \rightarrow$ *sent*(m') holds for two messages m and m' addressed to the same process, our scheme ensures that the same relation will also be present in the destination cluster and thereby *deliv*$(m) \rightarrow$ *deliv*(m') by the CMO protocol employed there. This successful ordering enforcement relies on a property of our scheme: the *Replay Property*. Consider the scenario shown in Figure 5. If *sent*$(m) \rightarrow$ *sent*(m') is present in cluster A, the CMO protocol employed in A ensures that *deliv*$(m) \rightarrow$ *deliv*(m') at X (the agent of A) and the FIFO message relaying rule ensures that *sent*$(m) \rightarrow$ *sent*(m') will be present in cluster D. Thus for m and m', the relation *sent*$(m) \rightarrow$ *sent*(m') is "replayed" in the next cluster.

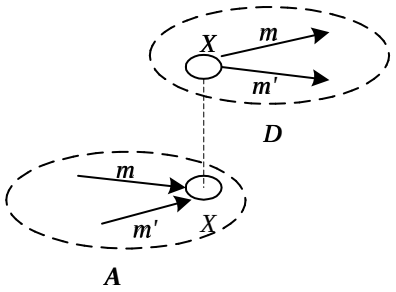

Fig. 5. Scenario illustrating the Reproduction Property.

Another important property is that by our message propagation rule, there is only one unique path between any two sites. Consider two messages m and $m^{'}$ addressed to the same process such that *sent*$(m) \rightarrow$*sent*$(m^{'})$. If they are sent by the same site, they will traverse the same propagation path. By the Replay Property, the same causal relation between them will be present in all clusters along the path. CMO will be respected in that case. If these messages are sent by different sites, the following theorem shows that CMO will still be preserved.

Theorem 1: Let m be a message sent by site X and destined for site Y. Let X_i be any site other than X and Y. Let m_i be the first message X sends to X_i after the sending of m. Then any message $m^{'}$ that is sent by X_i and destined for Y after the delivery of m_i will not be delivered before m at Y.

Proof: There are five possible message propagation routes among X, Y, and X_i, as shown in Figure 6. Note that as mentioned above, since there is only one unique path between any two sites, routes containing cycles are not possible. In (a), it follows from the Replay Property that causal relation *sent*$(m) \rightarrow$*sent*(m_i) will be recognized by all agents on the path from X to X_i (including X_i). Therefore, X_i will send (actually, relay) m before m'. By the Replay Property, causal relation *sent*$(m) \rightarrow$*sent*(m') will be present at all agents on the path from X_i to Y. CMO will thus be enforced by the CMO scheme employed in the last cluster.

In (b), *sent*(m) $\rightarrow$*sent*(m_i) will be recognized by all agents on the path from X to Y, since Y is located on the way of propagating m_i to X_i. Thus Y will deliver m before m_i. Message m' can never be delivered before m is because *sent*(m') occurs causally subsequent to *sent*(m_i). In (c), X sends m before sending m_i. Later, when message m' from X_i is delivered and then sent again (relayed) by X, *sent*(m) $\rightarrow$*sent*(m') will be honored. The Replay Property ensures that this relation will be present at agents on the path from X to Y. Thus m will be delivered before m' is. In (d), similar to the foregoing argument of (b), relation *sent*(m) $\rightarrow$*sent*(m_i) will be recognized at all sites from X to W (including W). Thus W sends (relays) m before m_i, and it is guaranteed by the Replay Property that *sent*(m) $\rightarrow$*sent*(m') will be recognized in the last cluster. Hence CMO is never violated. Case (e) is possible only when W_1, W_2, and W_3 are all in the same cluster. By the Reproduction Property, *sent*(m) $\rightarrow$*sent*(m_i) will by recognized by W_1. W_2 will observe *deliv*(m_i) $\rightarrow$*sent*(m') due to causality. Therefore, *sent*(m) $\rightarrow$*sent*(m') holds in this cluster. The CMO scheme at this cluster will guarantee the desired CMO. □

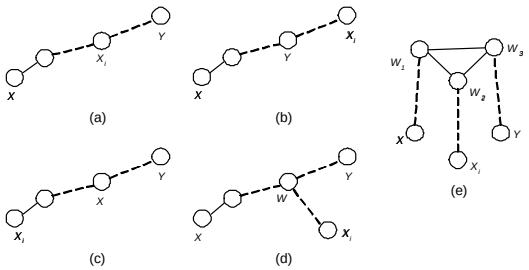

Fig. 6. Five possible routes for message propagations among X, Y, and X_i

IV. Performance Analysis and Comparison

We shall analyze and compare the performances of our approach to those of the centralized and the RST algorithm. We are primarily concerned with two measurements, namely, message cost and processing load. We assume that the system is organized in the following way. When the number of sites, n, is not greater than a certain number d, the whole system forms a single cluster. Otherwise, all sites are equally partitioned into b clusters, where $1 < b < d$. If after partition, the number of sites in a cluster is still greater than d, each cluster is further partitioned into b smaller clusters. The clustering process terminates when the number of sites in a cluster is less than or equal to d. In this way, the number of layers, l, that an n-site system will form can be derived by

$$l = \lceil \log_b \frac{n}{d} \rceil + 1 \qquad (1)$$

A. Message Cost

In this subsection we analyze the effects of clustering on reducing message cost. The message cost is defined as the extra space required in each message's header for CMO. In fact, this measure contributes to message transmission time as well as message processing time. We shall show that the message cost in our scheme is only a linear or even a logarithmic function of the number of processes.

The RST algorithm incurs $O(n^2)$ message cost. Suppose that the RST algorithm is employed in every cluster. The message cost of an intra-cluster message is proportional to the square of the cluster size. The message cost of an inter-cluster message is the sum of the costs of individual intra-cluster messages, each of which corresponds to a hop of that message. We are interested in evaluating the worst-case message cost. Let $C(n)$ denote the worst-case message cost for an n-site system. When n is not greater than d, the whole system forms a single cluster, and clearly $C(n) = cn^2$, where c is a constant. When n is greater than d, all sites together form a hierarchy of two or more layers. In that case, $C(n)$ corresponds to the cost of propagating a message from some non-agent site in Layer 1 to the top layer, exchanging the message between two agents in the top layer, and then propagating it to another non-agent site in Layer 1. Therefore we have

$$C(n) = \begin{cases} cn^2 & 1 \le n \le d \\ 2D(n) + cb^2 & n > d \end{cases}$$

where $D(n)$ denotes the worst-case cost of propagating a message between a site in Layer 1 and another site in the top layer. We define $D(n) = 0$ when $1 \le n \le d$. When $d < n \le bd$, the whole system forms a two-layer hierarchy, and $D(n)$ is proportional to the square of the Layer 1 cluster size. Thus $D(n) = c(n/b)^2$. When $n > bd$, the whole system forms a hierarchy of more than two layers. The value of $D(n)$ for a k-layer hierarchy is essentially the value of $D(n/b)$, the worst-case cost of propagating a message between a site in Layer 1 and a site in Layer $(k-1)$, plus cb^2, the cost of propagating a message between a site in Layer $(k-1)$ and a site in Layer k. Therefore, the value of $D(n)$ can be expressed as a recurrence relation:

$$D(n) = D(n/b) + cb^2$$

It is known [15] that $D(n) = O(\log n)$. Therefore, we arrive at the following equation:

$$C(n) = \begin{cases} cn^2 & 1 \le n \le d \\ c(2n^2/b^2 + b^2) & d < n \le bd \\ O(\log n) & n > bd \end{cases}$$

From the above we obtain the following result. When the whole system forms a two-layer hierarchy ($d < n \le bd$), setting $b = \sqrt{n}$ yields $O(n)$ message cost. When the whole system forms a hierarchy of three or more layers, the message cost is only $O(\log n)$, for b being fixed to some small number in comparison with n.

B. Processing Load

The processing load on a single site is defined as the average number of messages arriving at that site per unit of time, referred to as the site's *message arrival rate*, when the whole system is in a steady state. Suppose that in equilibrium, every site issues an average of λ messages per unit of time. Let message destination be a random variable that is uniformly distributed over all n sites (including the sender). If the system employs the RST algorithm, each site contributes a message arrival rate of λ/n to each of the other sites. So the processing load on each site is λ, independent of the number of sites in the system. On the other hand, if the system adopts the centralized approach, the message arrival rate at the coordinator is $n\lambda$. Clearly, the processing load on the coordinator increases in proportion to the number of sites in the system.

Regarding our approach, we assume that the centralized approach is adopted in every cluster, and the coordinator's functionality is incorporated in each cluster's agent. In addition, an extra site not in the system is employed to act as the coordinator for sites at the top layer. This site forwards messages for all other sites without initiating any messages of its own.

We shall derive the message arrival rates at the coordinator and at agents. The following basic rules [16] are needed in our derivation.

- If there are k independent message streams fed into site S at respective arrival rates $\lambda_1, \lambda_2, \ldots, \lambda_k$, the message arrival rate at S is $\sum_{i=1}^{k} \lambda_i$.
- Let m be a message stream feeding into site S_i with arrival rate λ. Suppose that S_i has total k downstream links leading respectively to $S_{i,1}, S_{i,2}, \ldots, S_{i,k}$, to each of which m can be forwarded by S_i with respective probabilities $p_{i,1}, p_{i,2}, \ldots, p_{i,k}$. Let $p_i = 1 - \sum_{j=1}^{k} p_{i,j}$ be the probability of directing m to S_i itself. Letting $\lambda_{i,j}$ be the average rate at which S_i forwards messages to $S_{i,j}$ (referred to as the *message departure rate* from S_i to $S_{i,j}$), we have $\lambda_{i,j} = p_{i,j} \cdot \lambda$. The same argument holds for message streams originating from S_i.

Given an n-site system, let $L(n)$ be the message arrival rate at the coordinator of the top layer. When $1 \leq n \leq d$, the system forms a one-layer hierarchy and $L(n) = n\lambda$, which is essentially the same as with the purely centralized approach. When $n > d$, the system forms a hierarchy of l layer ($l \geq 2$) as indicated by Equation (1). Let $m_{n,k}$ be a layer-k coordinator of an n-site system, where $1 \leq k \leq l$, and let $R(n,k)$ be the message arrival rate at $m_{n,k}$ that is contributed by all sites it serves. Since a layer-1 coordinator serves n/b^{l-1} sites, we have $R(n,1) = n\lambda/b^{l-1}$. Any other coordinator in layer-k, where $1 < k \leq l$, serves b sites, each of which is a layer-$(k-1)$ coordinator with message arrival rate $R(n,k-1)$. Since messages fed into a layer-$(k-1)$ coordinator can be addressed to some site served by the same coordinator, in this case an intercluster message propagation to the layer-k coordinator is not always required. Assume that a portion p of $R(n,k-1)$ constitutes the message departure rate from a layer-$(k-1)$ coordinator to the layer-k coordinator. Then we can formulate p as the ratio of how many sites are not served by the layer-$(k-1)$ coordinator to n, the total number of sites in the system. For a layer-2 coordinator, p is equal to

$$\frac{n - n/b^{l-1}}{n} = \frac{b^{l-1} - 1}{b^{l-1}}$$

Therefore,

$$\begin{aligned} R(n,2) &= b \cdot \frac{b^{l-1}-1}{b^{l-1}} \cdot R(n,1) \\ &= \frac{b^{l-1}-1}{b^{2l-3}} \cdot n\lambda \end{aligned}$$

For a layer-3 coordinator, p is equal to

$$\frac{n - n/b^{l-2}}{n} = \frac{b^{l-2} - 1}{b^{l-2}}$$

Therefore,

$$\begin{aligned} R(n,3) &= b \cdot \frac{b^{l-2}-1}{b^{l-2}} \cdot R(n,2) \\ &= \frac{(b^{l-1}-1)(b^{l-2}-1)}{b^{3l-6}} \cdot n\lambda \end{aligned}$$

In general, for a layer-k coordinator,

$$R(n,k) = \frac{\prod_{i=1}^{k-1}(b^{l-i}-1)}{b^j} \cdot n\lambda$$

where $j = kl - \sum_{i=1}^{k} i$. Therefore,

$$L(n) = R(n,l) = \frac{\prod_{i=1}^{l-1}(b^{l-i}-1)}{b^{l(l-1)/2}} \cdot n\lambda$$

Except for the coordinator of the top layer, every layer-k coordinator also has a message stream from its upper layer destined for some site within its service cluster. Let $S(n,k)$ be the message arrival rate at $m_{n,k}$ that is contributed by its layer-$(k+1)$ coordinator. Under our assumption of uniformly distributed destinations, a layer-k coordinator will divide its $S(n,k)$ into b equal parts, each of which contributes $S(n,k-1)$. Therefore,

$$S(n,k) = \begin{cases} L(n)/b & \text{if } k = l-1 \\ S(n,k+1)/b & \text{if } 1 \leq k \leq l-2 \end{cases}$$

Solving this recurrence relation, we have $S(n,k) = L(n)/b^{l-k}$. The total message arrival rate at $m_{n,k}$ is the sum of $R(n,k)$ and $S(n,k)$. As an example, consider a two-layer hierarchy. We have $L(n) = (b-1)n\lambda/b$ and $R(n,1) + S(n,1) = n\lambda/b + L(n)/b$. Figure 7 shows the values of $L(n)$ and $R(n,1) + S(n,1)$ for various settings of b. It can be seen that, on the condition of b being 2,

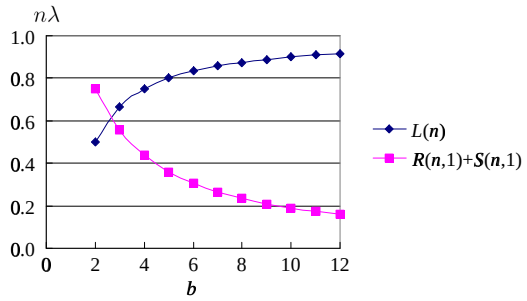

Fig. 7. Message arrival rates at coordinators versus b.

the processing load on the top-layer coordinator is only half of that on the coordinator in the centralized approach; however, the processing loads on the bottom-layer coordinators are relatively high. As b increases, the processing load on the layer-two coordinator increases while processing loads on other coordinators decrease. This can be explained by the fact that when $b = 2$, the amount of traffic due to inter-cluster messages is the same as that of intra-cluster messages. As b increases, the cluster size becomes smaller, thus an agent serves fewer clients. Meanwhile, inter-cluster messages increase, placing more processing loads on the top-layer coordinator.

V. Conclusions

Conventional CMO solutions take either centralized or fully distributed approaches. We have proposed a CMO scheme that unifies both kinds of CMO approaches by partitioning processes into clusters which can be further structured into a hierarchy. This manner of process clustering is susceptible to outstanding scalability and thereby efficient communication, in the following lines. In the centralized approach, the coordinator becomes a performance bottleneck when the number of processes increases beyond a certain value, making the approach unsuitable for large-scale distributed systems. By means of process clustering, twofold advantages result. The processing load can be alleviated substantially from the central coordinator and shared among agents, achieving better scalability. On the other hand, fully distributed approaches like the RST algorithm demand $O(n^2)$ message cost, which increases both message transmission and processing delays. Given our scheme, message overhead is decomposed into a number of small components. By setting cluster size appropriately, the message cost can be reduced to only $O(n)$ or even $O(\log n)$.

As a remark on our proposal, we note that the number of intermediate nodes to be traversed for propagating messages, namely hop count, can increase when more layers are introduced into the system. This, however, does not necessarily imply longer communication delay. Indeed, a simulation study conducted in [5] showed that the size of message space overhead can dominate the overall system performance. The significant reduction of this overhead by our scheme could outbalance the side effect of longer routing paths. Yet another remark pertinent to our proposal is stated as follows. All causal ordering, point-to-point protocols, including ours, are subject to a form of potential deadlock, referred to as *causal gap* [9] in the literature, if both node and link failures can occur. In contrast, causal multicast protocols like those deployed in the ISIS system [11], [14] do not suffer from this problem.

Nowadays large-scale networks such as the Internet are organized as a collection of subnetworks. If we view each subnetwork as a cluster and designate one site in each subnetwork as the agent, the system-wide CMO can be maintained with the flexibility that any CMO solution can be locally adopted in each cluster. This makes our proposal a practical solution and apt for modern non-propriety networks.

References

[1] A. Acharya and B. R. Badrinath. Recording distributed snapshots based on causal order of message delivery. *Inform. Process. Lett.*, 44:317–321, December 1992.

[2] F. Adelstein and M. Singhal. Real-time causal message ordering in multimedia systems. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 36–43, June 1995.

[3] M. Ahamad, P. Hutto, and R. John. Implementing and programming causal distributed memory. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 274–281, 1991.

[4] S. Alagar and S. Venkatesan. An optimal algorithm for distributed snapshots with causal message ordering. *Inform. Process. Lett.*, 50:311–316, 1994.

[5] S. Alagar and S. Venkatesan. Causal ordering in distributed mobile systems. *IEEE Trans. Comput.*, 46(3):353–361, March 1997.

[6] L. Alvisi and K. Marzullo. Message logging: Pessimistic, optimistic, and causal. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, June 1995.

[7] R. Baldoni, A. Mostefaoui, and M. Raynal. Efficient causally ordered communications for multimedia real-time applications. In *Proceedings of the 4th International Symposium on High Performance Distributed Computing*, pages 140–147, August 1995.

[8] K. Birman and T. Joseph. Reliable communications in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, February 1987.

[9] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, 1991.

[10] K. P. Birman. The process group approach to reliable distributed computing. *Comm. ACM*, 36(12):37–53,103, December 1993.

[11] K. P. Birman. *Building Secure and Reliable Network Applications*. Manning Publications Co., 1996.

[12] K. P. Birman and T. A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 123–138. ACM SIGOPS, November 1987.

[13] K. P. Birman and T. A. Joseph. Exploiting replication in distributed systems. In S. Mullender, editor, *Distributed Systems*. Addison-Wesley, New York, 1989.

[14] K. P. Birman and R. van Renesse. *Relibale Distributed Computing with ISIS Toolkit*. IEEE Computer Society Press, 1994.

[15] G. Brassard and P. Bratley. *Fundamentals of Algorithms*, chapter 4, pages 133–134. Prentice Hall, 1996.

[16] P. J. Burke. The output of a queuing system. *Operations Research*, 4:699–704, 1956.

[17] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. *Oper. Syst. Rev.*, 27:44–57, 1993.

[18] K.-H. Chi, L.-H. Yen, C.-C. Tseng, and T.-L. Huang. A causal multicast protocol for mobile distributed systems. *IEICE Transactions on Information and Systems*, E83-D(12):2065–2074, December 2000.

[19] D. Dolev and D. Malki. The transis approach to high availability cluster communication. *Comm. ACM*, 39(4):64–70, April 1996.

[20] J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference*, pages 56–66, February 1988.

[21] H. Garcia-Molina and A. Spauster. Ordered and reliable multicast communication. *ACM Trans. Comput. Syst.*, 9(3):242–271, August 1991.

[22] X. Jia. A total ordering multicast protocol using propagation trees. *IEEE Trans. Parallel Distrib. Syst.*, 6(6):617–627, 1995.

[23] T. Joseph and K. Birman. Low cost management of replicated data in fault-tolerant distributed systems. *ACM Trans. Comput. Syst.*, 4(1):54–70, 1986.

[24] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An efficient reliable broadcast protocol. *Oper. Syst. Rev.*, 23(4):5–19, 1989.

[25] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7):538–565, July 1978.

[26] F. Mattern. Virtual time and global states of distributed systems. In M. C. et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, North-Holland, 1989. Elsevier Science.

[27] S. Mishra, L. Peterson, and R. Schlicting. Implementing fault-tolerant replicated objects using psync. In *Proceedings of 14th Symposium on Operating System Principles*, Asheville, North Carolina, December 1993.

[28] R. Prakash, M. Raynal, and M. Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *J. Parallel Distrib. Comput.*, 41:190–204, 1997.

[29] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inform. Process. Lett.*, 39:343–350, September 1991.

[30] M. Reiter and L. Gong. Securing causal and relationships in distributed systems. *Computer Journal*, 38(8):633–642, 1996.

[31] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, 1989. Also published in *Lecture Notes in Computer Science*, 392.

[32] S. D. Stoller and F. B. Schneider. Verifying programs that use causally-ordered message-passing. *Science of Computer Programming*, 24:105–128, 1995.

[33] R. van Renesse. Causal controversy at Le Mont St.-Michel. *Oper. Syst. Rev.*, 27(2):44–53, April 1993.

[34] L.-H. Yen. Probabilistic analysis of causal message ordering. In *Proc. of 7th Int'l Conf. on Real-Time Computing Systems and Applications*, pages 409–413, Cheju Island, South Korea, November 2000.

[35] L.-H. Yen, T.-L. Huang, and S.-Y. Hwang. A protocol for causally ordered message delivery in mobile computing systems. *Mobile Networks and Applications*, 2(4):365–372, 1997.