2005－DPS－123（1）
2005／6／2

A Scalable Group Communication Protocol with Global Clock

Takshi Nishimura*, Naohiro Hayashibara*, Tomoya Enokido, and Makoto Takizawa***
Tokyo Denki University*, Japan
E-mail {take, haya, taki}@takilab.k.dendai.ac.jp
Rissho University, Japan**
E-mail eno@ris.ac.jp

Abstract

In peer-to-peer (P2P) applications, large number of peer processes are cooperating. In this paper, we discuss a scalable group of processes where processes are widely distributed in networks. Clocks of computers in every local network are synchronized by using the network time protocol (NTP) with a GPS time server. We discuss a global clock group (GCG) protocol where messages are causally ordered by using the physical time stamps. Messages not to be ordered by physical clock are furthermore ordered by using linear clock. We evaluate the protocol in terms of the number of messages ordered compared with the vector clock.

グローバルクロックを用いた大規模グループ通信プロトコル

西村 豪*　林原 尚浩*　榎戸 智也**　滝沢 誠*
東京電機大学理工学部情報システム工学科*
E-mail {take, haya, taki}@takilab.k.dendai.ac.jp
立正大学経営学部**
E-mail eno@ris.ac.jp

P2P アプリケーションでは多数の対等なプロセスが協調動作している。プロセスが広域に分散された大規模グループではベクタ時計による因果順序付けは、通信と処理の点で困難である。各ローカルネットワーク内の各コンピュータは、自身の時計を GPS 時刻サーバと NTP を用いて時刻同期できる。本論文では実時間を用いたメッセージの因果順序付け方法を提案する。実時間を用いて因果順序付けを行えない場合にも、線形時計を用いることにより、順序付けることにより、不要に順序付けが行われることを検出する。順序付けが行われるメッセージ数を評価し、本方式により線形時計にして不要な順序付けを減少できることを提案する。

1 Introduction

A large number, several thousands to possibly millions of peer processes are cooperating to achieve some objectives by exchanging messages with each other in peer-to-peer (P2P) systems [12]. A *group* [4, 11, 13] is a collection of cooperating peer processes. Here, messages have to be causally delivered to processes by using the vector clock [8]. In order to causally deliver messages, the vector clock is used in group protocols [4, 8]. However, the vector clock cannot be used in a scalable group due to the communication overhead, i.e. message length $O(n)$ for the number n of processes.

In the paper [5], processes are interconnected in a hierarchical loop network where messages are transmitted in a token passing mechanism. In the hierarchical daisy architecture [3], a group is composed of logical groups which provide the causally ordered delivery of messages by a *causal server* in presence of process faults. All causal servers are also members of causal servers group. Takamura *et al.* [15] discuss how to support the causally ordered delivery of messages in a hierarchical group by using the vector clock. Here, a group is composed of subgroups where processes in different subgroups exchange messages via gateway processes. Taguchi *et al.* [14] discuss hierarchical groups using where a vector clock whose size is the size of the subgroup. The authors [6] also discuss a two-layered group where processes in each local subgroup are synchronized by both physical clocks and linear clocks while gateway processes of subgroups are synchronized by vector clock.

Precise physical clocks like radio and GPS (Global Positioning System) clocks [10, 17] are now getting available even in a personal computer. A time server can be equipped with such a precise physical clock in a local network. Here, physical clocks of other computers can be synchronized with NTP (Network Time Protocol) [9] in a local network where variance of delay time is small. Processes in each local or personal area network can be synchronized so that every physical clock shows the “same” time. Every message is stamped with physical time when the message is transmitted. It is easy to design and implement algorithms for synchronizing multiple processes. In this paper, every process is synchronized by using the physical clock because of smaller overhead, i.e. the message length is $O(1)$. Messages are causally ordered by using the timestamps, maximum time differences of sender processes, and maximum delay time among processes. Even if the timestamp of a message

m_1 is smaller than another message m_2, m_1 might not causally precedes m_2. In order to prevent the unnecessarily ordering of messages, the linear clock is used in addition to the logical clock.

In section 2, we discuss types of clocks. In section 3, we discuss how to globally causally order messages in a scalable group. In section 4, we evaluate the protocol in terms of the number of messages to be ordered.

2 Clocks

2.1 Logical clocks

A group of processes are cooperating to achieve some objectives by exchanging messages with each other in distributed applications. Suppose a process p_1 sends a message m_1 to a pair of processes p_2 and p_3. The process p_2 sends a message m_2 after receiving m_1 from p_1. Here, m_1 causally precedes m_2 since m_1 is surely sent before m_2. More formally, the causality is defined based on the *happens-before* relation. A sending event of a message m *happens before* a receiving event of m. The *happens-before* relation is transitive. A messages m_1 *causally precedes* m_2 ($m_1 \rightarrow m_2$) if a sending event $s_1(m_1)$ of m_1 *happens before* the sending event $s_2(m_2)$. Every common destination process p_3 of m_1 and m_2 is required to receive m_1 before m_2 if $m_1 \rightarrow m_2$.

There are physical clocks and two types of logical clocks, linear clock [7] and vector clock [8]. In the *linear clock* [7], each process p manipulates a variable T showing its logical time named *linear time* (LT). Initially, $T = 0$. Each time a message m is sent, $T := T + 1$ and then m carries the logical time $m.T$ ($= T$). On receipt of a message m, $T := max(m.T, T)$. $m_1 \rightarrow m_2$ only if $m_1.T < m_2.T$. However, even if $m_1.T < m_2.T$, m_1 may not causally precede m_2. The message length is $O(1)$.

In the *vector clock* [8], each process p_s manipulates a *vector time* (VT) $T = \langle T_1, ..., T_n \rangle$ for number n of processes p_1, ..., p_n, where each element T_u is initially 0. Each time a message m is sent by p_s, $T_s := T_s + 1$. Then, m carries the vector T of p_s as $m.T$ ($= \langle m.T_1, ..., m.T_n \rangle$). On receipt of a message m from another process p_u, $T_u := max(T_u, m.T_u)$ ($u = 1, ..., n, u \neq s$) in each process p_s. $m_1 \rightarrow m_2$ if and only if (iff) $m_1.T < m_2.T$. m_1 is *causally concurrent* with m_2 ($m_1 \parallel m_2$) if neither $m_1.T \leq m_2.T$ nor $m_1.T \geq m_2.T$. The message length is $O(n)$.

2.2 Physical clocks

Every pair of physical clocks in different computers in nature do not always show the same time. Recently, radio [10] and GPS (Global Positioning System) clocks [10, 17] are available. A time sever with such a precise physical clock can be supported in each local network. Time differences of clocks synchronized with the radio clock of JST (Japan Standard Time) and GPS satellite clock are around 1.5 [msec] in Japan and 1 [μsec], respectively. NTP (Network Time Protocol) [9] with a time server is used to synchronize physical clocks of computers in a network. Processes communicate with a time server to obtain the current time by using NTP. Time difference between clocks of a time server and each computer is longer than 100 [msec] depending on variance of delay time in networks.

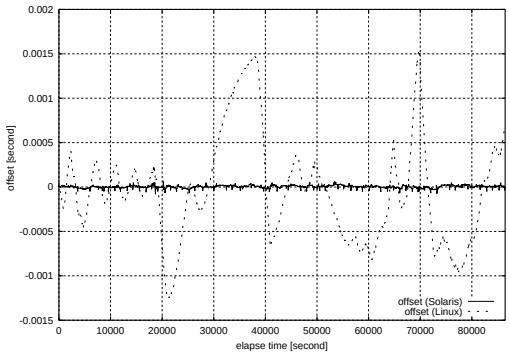

Figure 1. Time difference .

Let $c_s(t)$ show local real time shown by the physical clock c_s of a computer on which a process p_s is performed at UTC (Coordinated Universal Time) [16] t. $c_s(t)$ shows *local real time* (RT) or simply *local time* of p_s at UTC t. Let $\delta_s(t)$ show the time difference $|t - c_s(t)|$ of the local time $c_s(t)$ of a process p_s at UTC t. Let τ_s be the maximum time difference of p_s, i.e. $|t - c_s(t)| \leq \tau_s$ for every UTC t. Here, UTC t is assumed to be given by the GPS time server ts. We measured the time difference $\delta_{spc}(t)$ of each of three types of operating systems, Solaris [2], Linux [1], and Windows of a computer *spc* (Celeron 700MHz CPU, 256MB memory). The computer spc is connected with the time server ts in a Gigabit Ethernet with NTP. The delay time in the local network is almost stable, which is obtained by measuring the round trip time between ts and spc. The time difference of Windows is longer than 10 milliseconds which is too long. Figures 1 shows the time differences $\delta_{spc}(t)$ for UTC t on Solaris and Linux. The maximum time difference τ_{spc} of the Solaris and Linux spc with a GPS clock are 0.1 and 1.5 [msec], respectively. As shown here, each process p_s is characterized by its maximum time difference τ_s, mainly depending on t_s operating system. Let ct_s show the local time of a process p_s. ct_s is *newer* than ct_u ($u \neq s$) ($ct_s >_{st} ct_u$) iff $|ct_s - ct_u| \geq \tau_s + \tau_u$. ct_s is *equivalent* with ct_u ($ct_s \equiv_{su} ct_u$) iff $|ct_s - ct_u| \leq \tau_s + \tau_u$. A pair of physical clocks c_s and c_u are *equivalent* iff $c_s(t) \equiv_{su} c_u(t)$ for every UTC t.

Suppose a pair of events e_1 and e_2 occurs at UTC t_1 and t_2, respectively. The event e_1 physically happens before e_2 if $t_1 < t_2$

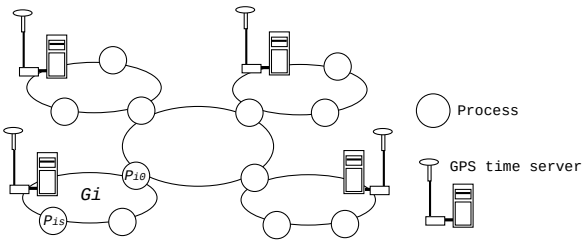

Figure 2. Hierarchical group.

3 Realtime-Based Causality

3.1 Broadcast group

A group G is composed of *local* subgroups $G_1, ..., G_k$ which are interconnected with a *global* subgroup G_0 as shown in Figure 2. Each local subgroup G_i is composed of a gateway process p_{i0}, normal processes p_{i1}, ..., p_{il_i} ($l_i \geq 1$), and a time server ts_i. Processes in each local subgroup are interconnected in a local area network. The gateway process p_{i0} communicates with not only local processes in G_i but also the other gateway processes. Thus, the global subgroup G_0 is a collection of the gateway processes p_{10}, ..., p_{k0}. Gateway processes are interconnected in a wide-area network. The physical clocks of all processes in each local subgroup are synchronized with each other by NTP with a GPS time server.

Suppose there are three processes p_{is}, p_{jv}, and p_{ku} in local subgroups G_i, G_j, and G_k, respectively. Each message m carries timestamp $m.RT$ showing physical time when the sender process sends m. Suppose that a process p_{is} sends a message m_1 at local time $c_{is}(t_1)$ for UTC t_1 and another process p_{jv} sends m_2 at local time $c_{jv}(t_2)$ to a process p_{ku} for UTC t_2 [Figure 3]. Here, $m_1.RT = c_{is}(t_1)$ and $m_2.RT = c_{jv}(t_2)$. As presented before, each process p_{ik} has its own maximum time difference τ_{ik}. Here, $|c_{is}(t_1) - t_1| \leq \tau_{is}$ and $|c_{jv}(t_2) - t_2| \leq \tau_{jv}$. $t_1 < t_2$ or $t_1 > t_2$ if $|c_{is}(t_1) - c_{jv}(t_2)| > \tau_{is} + \tau_{jv}$. However, if $|c_{is}(t_1) - c_{jv}(t_2)| \leq \tau_{is} + \tau_{jv}$, we cannot decide whether $t_1 < t_2$ or $t_1 > t_2$. If $(m_2.RT_2 - m_1.RT_1) \geq (\tau_{is} + \tau_{jv})$, a sending event $s_{jv}(m_2)$ of m_1 *physically happens* before a sending event $s_{is}(m_1)$ of m_2 with respect to UTC. However, it is not sure whether or not $m_1 \rightarrow m_2$.

In this paper, we assume every network is synchronous. Let $\delta_{is,jv}$ and $\Delta_{is,jv}$ show the minimum delay time and maximum delay time between a pair of processes p_{is} and p_{jv}, respectively. In Figure 3, a process p_{jv} sends a message m_2 after receiving m_1. This means, a sending event $s_{jv}(m_2)$ of happens in $\delta_{is,jv}$ time units after a sending event $s_{is}(m_1)$. It is straightforward for the following theorem to hold:

[Theorem 1] Suppose messages m_1 and m_2 are sent by processes p_{is} and p_{jv}, respectively. If m_1 is sent to the process p_{jv} and $c_{jv}(t_2) - c_{is}(t_1) \geq \tau_{is} + \tau_{jv} + \Delta_{is,jv}$, m_1 causally precedes m_2 ($m_1 \rightarrow m_2$).

In a *broadcast* group, every message is broadcast to every process. Causally concurrent messages can be ordered by using the theorem:

[Theorem 2] Every process p_{ku} can deliver a message m_1 before another message m_2 ($m_1 \Rightarrow m_2$ if $m_2.RT - m_1.RT > \tau_{is} + \tau_{jv} + \Delta_{is,jv}$ in a broadcast group.

Every common destination process p_{ku} of messages m_1 and m_2 cannot decide which one of m_1 and m_2 causally precedes the other in a subgroup by using the timestamps if $\delta_{is,jv} \leq |m_2.RT - m_1.RT| - (\tau_{is} + \tau_{jv}) \leq \Delta_{is,jv}$. Here, let $m.LT$ show the linear time of a message m given by the sender process.

[Theorem 3] Let m_1 and m_2 be messages sent by processes p_{is} and p_{jv} in local subgroups G_i and G_j, respectively. In a broadcast group G, m_1 causally precedes m_2 ($m_1 \rightarrow m_2$) only if

1. $m_2.RT - m_1.RT > \tau_{is} + \tau_{jv} + \Delta_{is,jv}$.
2. $m_2.LT > m_1.LT$ if $\delta_{is,jv} \leq |m_2.RT - m_1.RT| - (\tau_{is} + \tau_{jv}) \leq \Delta_{is,jv}$.

From the theorem 3, for a pair of messages m_1 and m_2 received from p_{is} and p_{jv}, respectively, "m_1 precedes m_2" ($m_1 \Rightarrow m_2$) by the conditions in the theorem.. It is trivial that $m_1 \Rightarrow m_2$ if m_1 causally precedes m_2 ($m_1 \rightarrow m_2$).

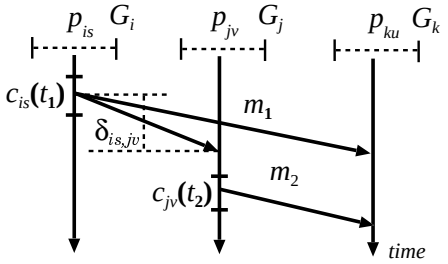

Figure 3. Causality.

3.2 Multicast group

In a multicast group, each message is sent to some destination processes, no necessarily all the processes. Even if $m_2.RT - m_1.RT > \tau_{is} + \tau_{jv} + \Delta_{is,jv}$, a message m_1 may not causally precede another message m_2.

[Theorem 4] Suppose a pair of messages m_1 and m_2 are sent by processes p_{is} and p_{jv}, respectively, in a $multicast$ group. m_1 causally precedes m_2 ($m_1 \rightarrow m_2$) only if the following conditions hold:

1. $m_2.RT - m_1.RT \geq \tau_{is} + \tau_{jv} + \Delta_{is,jv}$ if $m_1.LT < m_2.LT$.
2. $m_1.LT < m_2.LT$ if $\delta_{is,jv} \leq |m_2.RT - m_1.RT| - (\tau_{is} + \tau_{jv}) \leq \Delta_{is,jv}$.

Even if $m_1.LT < m_2.LT$, a common destination process of m_1 and m_2 cannot deliver m_1 before m_2 since m_1 may not causally precede m_2. Here, m_1 is delivered before m_2 only if m_1 is surely sent before m_2 in G_j, i.e. $\tau_{is} + \tau_{jv} + \Delta_{is,jv} > m_2.RT - m_1.RT \geq \tau_{is} + \tau_{jv} + \delta_{is,jv}$.

3.3 Delay time

In a scalable group, it is not easy to maintain information on maximum time difference τ_{is} of each process p_{is} and maximum delay time $\Delta_{is,jv}$ between every pair of p_{is} and p_{jv}. In this paper, each process p_{is} is assumed to know its maximum time difference τ_{is}. Each message m sent by p_{is} carries its maximum time difference $m.\tau$ ($= \tau_{is}$). On receipt of a message m from the process p_{is}, a process p_{jv} can know the maximum time difference τ_{is} ($= m.\tau$) of p_{is}.

For every pair of processes p_{is} and p_{iv} in a local subgroup G_i, we assume the minimum and maximum delay times $\delta_{is,iv}$ and $\Delta_{is,iv}$ between p_{is} and p_{iv} to be constants δ_i and Δ_i, respectively. In fact, Δ_i can be the maximum delay time among every pair of processes in G_i. Every message m sent by a process p_{is} in a subgroups G_i carries the minimum delay time $m.\delta$ ($= \delta_i$) and the maximum delay time $m.\Delta$ ($= \Delta_i$). Let δ_{ij} and Δ_{ij} be the minimum and maximum delay times between a pair of gateways p_{i0} and p_{j0} of G_i and G_j, respectively. A gateway p_{i0} monitors the delay time with another gateway p_{j0}. Each process p_{is} is assumed to obtain the minimum and maximum delay times δ_{kl} and Δ_{kl} between every pair of G_k and G_l. The minimum and maximum delay times $\delta_{is,jv}$ and $\Delta_{is,jv}$ between p_{is} and p_{jv} can be obtained by computing $\delta_i+\delta_{ij}+\delta_j$ and $\Delta_i+\Delta_{ij}+\Delta_j$ where $\langle \delta_i, \Delta_i \rangle$ and $\langle \delta_j, \Delta_j \rangle$ are obtained by messages from p_{is} and p_{jv}, respectively.

Although δ_i is stable in each G_i, δ_{ij} and Δ_{ij} may not be assumed to be stable in a wide area network. In such environment, $m_1 \Rightarrow m_2$ by the following rule:
[Ordering rule] A message m_1 precedes another message m_2 ($m_1 \Rightarrow m_2$) if

1. $m_2.RT - m_1.RT > \tau_{is} + \tau_{jv} + E_{is,jv}$ if $m_1.LT < m_2.LT$.
2. $m_1.LT < m_2.LT$ if $\epsilon_{ij} \leq |m_2.RT - m_1.RT| - (\tau_{is} + \tau_{jv}) \leq E_{is,jv}$ where $\epsilon_{ij} = \delta_i + \delta_j + \delta_{ij}$ and $E_{ij} = \Delta_i + \Delta_i + \Delta_{ij}$ if $G_i \neq G_j$, else $\epsilon_{ij} = \delta_i + \delta_j$ and $E_{ij} = \Delta_i + \Delta_j$.

4 Evaluation

We evaluate the GCG (global clock group) protocol discussed in this paper compared with a traditional group protocol of linear clock. In a GCG protocol, a group G is composed of local subgroups $G_1, ..., G_k$ with a global subgroup G_0. An LT (linear time) group G is composed of the same normal processes as the GCG group G, where all the processes directly communicate with each other. We measure how many messages are causally ordered with each other in a scalable group by the GCG and LT protocols. A pair of messages m_1 and m_2 are referred to as *unnecessarily ordered* if the messages are causally concurrent ($m_1 \mid\mid m_2$) but are ordered, $m_1 \Rightarrow m_2$ or $m_2 \Rightarrow m_1$ in a protocol.

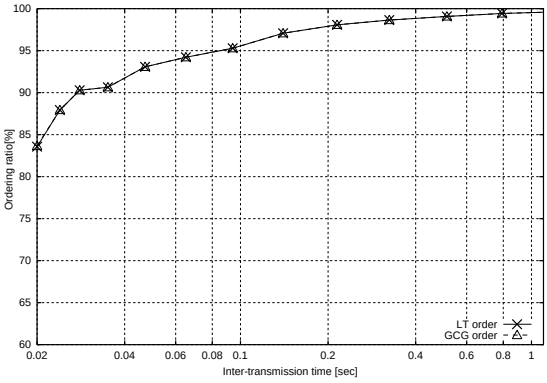

Figure 4. Ordering ratios of GCG and LT (broadcast).

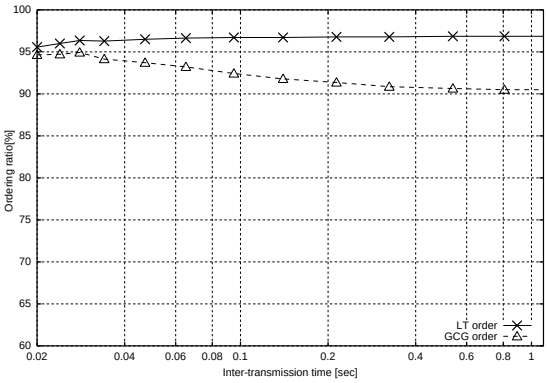

Figure 5. Ordering ratios of GCG and LT (multicast).

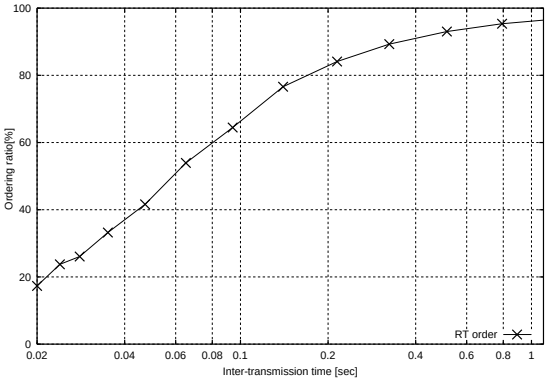

Figure 6. Ordering ratio of RT (broadcast).

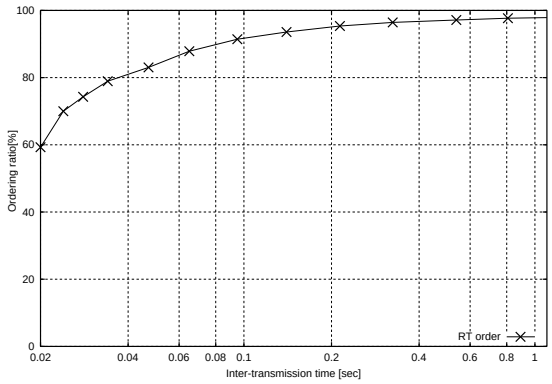

Figure 7. Ordering ratio of RT (multicast).

We assume that each local process sends a message every λ time units to destination processes. In the multicast group, destination processes are randomly selected for each message in the group G. A network is assumed to be reliable and the delay time between every pair of processes is 1 [msec] in this evaluation. We assume a physical clock of every process p_i in a local subgroup is synchronized in the maximum time difference τ_i ($1 < \tau_i < 5$) [msec]. For example, a physical clock of every computer is synchronized in NTP with a GPS time server in each local subgroup. Here, maximum time difference τ_i is determined by measuring the time difference with the GPS time server.

In the evaluation, each local subgroup G_i includes five processes p_{i0}, ..., p_{i4} and the global subgroup G_0 includes ten local subgroups. Processes in each local subgroup are realized in one blade (Pentium M 1.0[GHz] CPU, 512[MB] memory, Linux Kernel 2.4.26). The group G is realized by ten blades in the HP Blade server (ProLiant BL10e). Every blade is interconnected with a GPS time server in the 100base-T Ethernet.

Messages received are message is stored in a receipt queue RQ_{is} of a process p_{is} in the GCG and LT protocols. Here, messages are dequeued for the queu RQ_{is} if the messages are surely received by every destination process. Messages in the receipt queue RQ_{is} are ordered in $\Rightarrow$ by the ordering rules discussed in this paper. On receipt of a message m, messages in the receipt queue RQ_{is} are compared with the message m to order the messages. Here, we count up the number of messages in the receipt queue RQ_{is} which precede the message m by the ordering rule. The ratio RQ_{is} (m) for message m is the ratio of as the number of messages ordered in the α protocol ($\alpha\epsilon$ [GCG, LT]) to the number of messages in the queue for the message m. The ordering ratio OR_α is the average ratio $OR_\alpha(m)$ for every message m.

Figures 4 and 5 show the ordering ratios[%] of the number of messages ordered in the GCG and LT protocols to the total number of messages transmitted in each of the GCG and LT protocols for inter-transmission time λ in broadcast and multicast groups, respectively. On receipt of a message m, a process compares the timestamp of m with messages in the receipt queue. Figure 4 shows the same number of the broadcast messages are causally ordered in the GCG and LT protocols. On the other hand, the number of multicast messages causally ordered are reduced about 7% in the GCG protocol than the LT protocol as shown in Figure 5. This means, the number of messages unnecessarily ordered can be reduced in the GCG protocol.

Figures 6 and 7 show the ratios[%] of the number of messages ordered by the real time (RT) timestamp with the maximum time difference τ of each process in the GCG protocol for broadcast and multicast groups. In Figure 6, the ratio is monotonically increased from 20% to 90% for inter-transmission time λ. Figure 7 shows that the ratio is monotonically increased from 60% to 98%.

In scalable groups, the number of messages unnecessarily ordered in the GCG protocol is smaller than the LT protocol. In addition, the message length is $O(1)$ in the GCG protocol as well as the LT protocol.

5 Concluding Remarks

In distributed applications, a large number of peer processes are cooperating. In this paper, we proposed a global clock group (GCG) protocol where every process uses the physical time to do the synchronization with the other processes. Each process supports a physical clock fully synchronized with the other processes by taking usage with NTP and a GPS time server in each local subgroup. Messages are ordered by using physical time. In addition, we discussed how to prevent messages from unnecessarily ordered by additionally using the linear clock. We showed the number of messages unnecessarily ordered can be decreased in the GCG protocol compared with the LT protocol.

References

[1] Linux Kernel Archives. http://www.kernel.org/.

[2] Sun Microsystems, Inc., Solaris Operating System. http://www.sun.com/.

[3] R. Baldoni, R. Beraldi, R. Friedman, and R. van Renesse. The Hierarchical Daisy Architecture for Dausal Delivery. *Distrib. Syst. Engng.*, 6(2):71–81, 1999.

[4] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, 1991.

[5] X. Chen, L. E. Moser, and P. M. Melliar-Smith. Reservation-Based Totally Ordered Multicasting. In

Proc. of the 16th IEEE International Conf. on Distributed Computing Systems (ICDCS-16), pages 511–519, 1996.

[6] S. Kawanami, T. Enokido, and M. Takizawa. A Group Communication Protocol for Scalable Causal Ordering. In *Proc. of IEEE the 18th International Conf. on Advanced Information Networking and Applications (AINA-2004)*, volume 1, pages 296–301, 2004.

[7] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21(7):558–565, 1978.

[8] F. Mattern. Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[9] D. L. Mills. Network Time Protocol. RFC 1350, 1992.

[10] D. L. Mills. Improved Algorithms for Synchronizing Computer Network Clocks. *IEEETNWKG: IEEE/ACM Transactions on NetworkingIEEE Communications Society, IEEE Computer Society and the ACM with its Special Interest Group on Data Communication (SIGCOMM), ACM Press*, 3:245–254, 1995.

[11] A. Nakamura and M. Takizawa. Causally Ordering Broadcast Protocol. In *Proc. of the 14th IEEE International Conf. on Distributed Computing Systems (ICDCS-14)*, pages 48–55, 1994.

[12] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[13] L. Rodrigues and M. Raynal. Atomic Broadcast in Asynchronous Crash-Recovery Distributed Systems. In *Proc. of the 20th IEEE International Conf. on Distributed Computing Systems (ICDCS-20)*, pages 288–295, 2000.

[14] K. Taguchi, T. Enokido, and M. Takizawa. Causally Ordered Delivery for a Hierarchical Group. In *Proc. of IEEE 10th International Conf. on Parallel and Distributed Systems (ICPADS-10)*, pages 453–460, 2004.

[15] M. Takizawa, M. Takamura, and A. Nakamura. Group Communication Protocol for Large Group. In *Proc. of the 18th IEEE Conf. on Local Computer Networks (LCN)*, pages 310–319, 1993.

[16] The Bureau International des Poids et Mesures (BIPM). http://www1.bipm.org/.

[17] P. Verissimo, L. Rodrigues, and A. Casimiro. Cesiumspray: a Precise and Accurate Global Time Service for Large-scale Systems. *Real-Time Systems*, 12(3):243–294, 1997.