Will be / is published in the
Proceedings of the 5th EUROMICRO Workshop on Parallel and Distributed Processing (PDP'97), 22-24 January 1997, London, UK.


# The Controlled Logical Clock – a Global Time for Trace Based Software Monitoring of Parallel Applications in Workstation Clusters

Rolf Rabenseifner
Computing Center University of Stuttgart, Allmandring 30, D-70550 Stuttgart, Germany, Tel. ++49 711 6855530, e-mail: rabenseifner@rus.uni-stuttgart.de

## Abstract

*Event tracing and monitoring of parallel applications are difficult if each processor has its own unsynchronized clock. A survey is given on several strategies to generate a global time, and their limits are discussed.*

*The controlled logical clock is a new method based on Lamport's logical clock and provides a method to modify inexact timestamps of tracefiles. The new timestamps guarantee the clock condition, i.e. that the receive event of a message has a later timestamp than the send event. The corrected timestamps can also be used for performance measurements with pairs of events in different processes.*

*The controlled logical clock is motivated and it is analyzed in detail by computer simulations. No additional protocol overhead is needed for the new method while tracing an application. It can be implemented as a filter for tracefiles or it can be integrated into monitor tools for parallel applications.*

**Keywords:** Logical time, global time, clock, monitoring, clock synchronization, causality, distributed computing, parallel computing.

## 1. Introduction

With global time a correct representation of the time sequence of tracing information is possible. Parallel and distributed applications can be analyzed to obtain an insight into their behavior and their performance. On systems without global time there are different strategies of approximating a global time. This paper presents a new algorithm – the controlled logical clock. It was developed especially for workstation clusters or comparable systems and can be used as a filter for tracefiles. The processors' clocks used for generating timestamps are synchronized subsequently if the tracefile contains backward references. Backward references are messages with a timestamp of the send event that is later than the timestamp of the receive event. The controlled logical clock modifies the timestamps to fulfill the clock condition [18], i. e. the send event always has an earlier timestamp than the corresponding receive event. Besides the generation of the timestamps and their collection in the monitor tool there is no additional protocol effort. An exchange of the timestamps along with the messages is not necessary because those tools have complete insight into the trace information of the application.

The controlled logical clock requires clock differences to be limited to about 2 ms by a previous synchronization. This can be achieved by an explicit synchronization before and after the run of the application [20] and a linear interpolation – as done by several monitor tools –, or it can be done by a resources saving continuous synchronization, e. g. with xntp [22].

The controlled logical clock does not modify the timestamps of the original clocks $C_i$ as long as the timestamp of a receive event is later than the timestamp of the corresponding send event plus the minimum message delay. The controlled logical clock is a modification of Lamport's piecewise differentiable logical clock $LC_i$ with $dLC_i(t)/dt := dC_i(t)/dt$ as described also in [16]. Hence as opposed to this clock, the time difference $LC - C$ of the controlled logical clock is limited. It is well suited for visualizing the causal order. It can also be used for averaged time difference measurements between events in different processes. For such measurements within one process the original clock is more appropriate. But the timestamps of the controlled logical clock can be used also because, in general, its deviation is less than 5 %. The small deviation is evident because most monitor tools do not support more than one time scale.

## 2. Related Works

For monitoring and performance analysis of parallel and distributed applications there are different approaches to synchronizing local clocks. Lamport's discrete logical clock [18] can be used directly for monitoring [4, 31]. In addition to this Raynal [27] proposes an algorithm to prevent the drift between the logical clocks. The vector clock – an enhancement from Fidge [11, 12] and Mattern [21] – allows an equivalent representation of the causal order given by the send/receive event pairs. It is used in some monitor tools [5, 10, 19]. In [13] global events are introduced and in [25] spontaneous events (e.g. collisions on a network) are taken into account. In the summary in [29] the limits of the logical clock and the vector clock are illustrated.

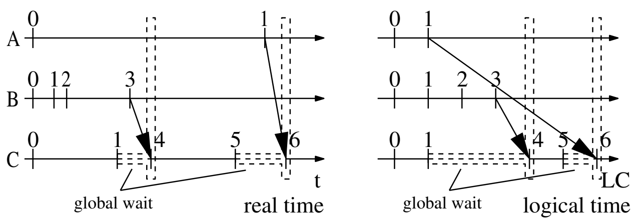



**Figure 1. Global waits visualized in real time and in discrete logical time**

A further limit of the usability of the logical clock is shown in Fig. 1. The example has two global waits. Process C waits until a message can be received from A or B. The time axis is dotted during periods of waiting. In the left picture the example is visualized by using a real time axis. In the right picture the logical time is used as time scale. This representation is only of limited value because it seems that process A has sent its message at $t = 1$, but this message is not taken into account in the global wait of process C also started at $t = 1$. The wait is not finished until process B sends its message at $t = 3$. This example shows that the logical clock is not sufficient for visualizing monitor information.

A representation without backward references can be achieved alternatively with a sufficiently exact synchronization of the local clocks [15, 18, 20]. Often used methods are an exact synchronization and drift estimation before the beginning of the application [8, 9] or better before and after the application [20] with a linear interpolation while the application is running. The exact synchronization can be done by deterministic [28] or probabilistic [2, 3] methods.

Continuous synchronization with little resource usage (e. g. xntp [22]) is normally not sufficiently exact due to the large jitter of the message delays in local area networks.

The trace based synchronization is another alternative. The differences between the clocks are computed by the rule that a receive event must not arrive before the send event plus the minimum transfer time. Duda [7] has developed two algorithms, one with a regression analysis, and another with a convex hull. Using a minimal spanning tree algorithm Jezequel [16] has adopted Duda's algorithm for any processor topologies. Hofmann [14] has improved Duda's algorithm by using a simple minimum/maximum strategy, and he has proposed dividing the execution time into several intervals to compensate different clock drifts in long running applications. Babaoğlu and Drummond [1, 6] show that an almost no cost clock synchronization is possible if the application makes a full message exchange between all processors in sufficient short intervals. A survey of further work can be found in [30]. The limits of these methods are given by the message delay jitter, by the non linear relation of message delay and message length, and by a one-sided communication topology in some applications (e. g. producer/consumer scenarios).

The controlled logical clock is a novel development based on both Lamport's logical clocks, the discrete case and that with $dLC_i(t)/dt := dC_i(t)/dt$. It also contains components of the trace based synchronization because it does not need any further protocol overhead if it is used in monitor and debug tools. It implements a subsequent synchronization based on existing timestamps and their communication relations.

In contrary to the trace based synchronization, the controlled logical clock can also be used in systems with clock ticks longer than the minimum message delay. But the controlled logical clock needs a preceding synchronization that limits the differences between the original clocks to the fourfold of the minimum message delay.

## 3. The Clock Condition

For monitoring, clocks are needed which are sufficiently exact for performance measurements and which hold the clock condition defined below [18]. The clock condition must be held to enable the visualization of a program in a spacetime diagram.

**Definition 1** $n$ is the number of processes, $e_i^j$ is the $j^{\text{th}}$ event in the process $i$,
$E = \{e_i^j | i = 1..n, j = 0..j_{\max}(i)\}$ is the set of events, and $M = \{(e_k^l, e_i^j) | e_k^l$ is a send event, and $e_i^j$ is its corresponding receive event$\}$ is the set of send/receive pairs. $e_i^j$ is a internal event if it is not a send event and not a receive event.
For two events $e_k^l$, $e_i^j$ the relation $e_k^l$ *happened directly before* $e_i^j$, shortened by $e_k^l \tilde{\rightarrow} e_i^j$, is held, if and only if
(a) $(e_k^l, e_i^j) \in M$, or (1)
(b) the events are in the same process and they succeed one another, i. e. $k = i \wedge l = j - 1$. (2)

The relation *happened before*, shortened by $\rightarrow$, is the transitive hull of the relation $\tilde{\rightarrow}$, i. e. the smallest relation that additionally satisfies

(c) $e_k^l \rightarrow e_i^j \wedge e_i^j \rightarrow e_n^m \implies e_k^l \rightarrow e_n^m$ (3)

**Definition 2** A clock $\mathcal{C} : E \mapsto \mathbb{R}$ satisfies the *Clock Condition* if and only if

$$\forall_{e_k^l \in E, e_i^j \in E} e_k^l \rightarrow e_i^j \implies \mathcal{C}(e_k^l) < \mathcal{C}(e_i^j) \tag{4}$$

Neglecting different clock drifts the following theorem can be simply shown:

**Theorem 1** *If the differences between the processors' clocks are constantly less than the minimum message delay then the clock condition is held by the processors' clocks.*

A more precise theorem for theoretical continuous clocks with limited drifts can be found in [17, 20]. In [15] real clocks with discrete clock ticks are analyzed. This paper examines the case that the clock differences are in general longer than the minimum message delay, but limited by about the fourfold of the minimum message delay. This limit can be achived with low cpu and network costs by usual software synchronization tools. In this case the premise of Theorem 1 is not held.

## 4. The Simple Logical Clock

The simple logical clock is an enhancement of Lamport's logical clock. The name *simple* is chosen to distinguish clearly between it and the *controlled* logical clock that is itself based on the simple one.

First a *weak synchronization* must be done. There are different methods to achive clock errors less than about the fourfold of the minimum message delay, e.g. SBA – sampling before and after the application's run with a linear balancing in the meantime [20] or a low cost clock synchronization by software (timeslave, timed, xntp). Because these methods must not synchronize with the wall clock time (UTC), the synchronization is modeled with:

**Definition 3** $t$ is the wall clock time, $T(t)$ is the global time to which the process clocks $C_i(t)$ $(i = 1..n)$ are synchronized with limited errors, i.e. constants $e_i^-$ and $e_i^+$ exist with $-e_i^- \leq C_i(t) - T(t) \leq e_i^+$.

Now for each process the logical clock $LC_i$ will be defined: it nearly stops while $LC_i > C_i$ and else it equals $C_i$; at each receive event it is set on the maximum of its current value and of the sender's clock at the time of sending plus the minimum transfer delay $\mu$. The minimum transfer delay should be estimated as a byproduct of the synchronization at the beginning or at the end. This logical clock is a modification of Lamport's logical clock. In the following it is named the **simple logical clock**. In the next section it will be enhanced to the controlled logical clock.

**Algorithm 1.** The simple logical clock $LC$ is exactly defined with

$$LC_i(e_i^j) := \begin{cases} \max(LC_k(e_k^l) + \mu_{k,i},\, LC_i(e_i^{j-1}) + \delta_i, & \\ \quad C_i(t(e_i^j))) \text{ if } \exists_{e_k^l} (e_k^l, e_i^j) \in M & (5) \\ \max(LC_i(e_i^{j-1}) + \delta_i,\, C_i(t(e_i^j))) & \\ \quad \text{otherwise} & (6) \\ \text{and if } j = 0 \text{ then the terms} & \\ \quad LC_i(e_i^{j-1}) + \delta_i \text{ must be omitted} & \end{cases}$$

with

$\delta_i$ = minimal difference between two events in process $i$, i.e. $\delta_i$ are constants with

$$\delta_i > 0 \quad \wedge \quad \forall_{i,j} T(t(e_i^j)) - T(t(e_i^{j-1})) \geq \delta_i \tag{7}$$

$\mu_{k,i}$ = minimum message delay of messages from process $k$ to process $i$, i.e. $\mu_{k,i}$ are constants with

$$\mu_{k,i} > 0 \quad \wedge \quad \forall_{(e_k^l, e_i^j) \in M} T(t(e_i^j)) - T(t(e_k^l)) \geq \mu_{k,i} \tag{8}$$

The global simple logical clock is then defined as

$$LC(e_i^j) := LC_i(e_i^j) \tag{9}$$

**Theorem 2** *The simple logical clock LC satisfies the* **clock condition**.

*Proof.* The algorithm 1 satisfies Lamport's rules IR1 and IR2 in [18] and therefore holds the clock condition. □

In the following the error will be modeled based on $C_i$:

$$\forall_{e_i^j} t_0 < t(e_i^j) < t_e \quad \wedge$$

$$\forall_{t : t_0 < t < t_e} -e^- \leq -e_i^- \leq C_i(t) - T(t) \leq e_i^+ \leq e^+ \tag{10}$$

i.e. the corrected clocks $C_i$ are at maximum $e_i^+$ fast and at minimum $e_i^-$ slow in comparison with the global time T.

**Theorem 3** *If all clocks $C_i$ are not more than $e^+$ fast then the simple logical clock LC is also not more than $e^+$ fast, i.e.*

$$\forall_{e_i^j} C_i(t(e_i^j)) - T(t(e_i^j)) \leq e^+ \implies \forall_{e_i^j} LC(e_i^j) - T(t(e_i^j)) \leq e^+$$

*Proof.* Assuming there is a $(i, j)$ with

$$LC(e_i^j) - T(t(e_i^j)) > e^+ \tag{11}$$

and without loss of generality $e_i^j$ may be the earliest event satisfying (11) (12)

Case a) $LC(e_i^j) = LC_i(e_i^j)$ is defined with (5). Then

$$\begin{aligned} LC(e_i^j) - T(t(e_i^j)) &\overset{(9)}{=} LC_i(e_i^j) - T(t(e_i^j)) \\ &\overset{(5)}{=} \max(LC_k(e_k^l) + \mu_{k,i}, \\ &\qquad LC_i(e_i^{j-1}) + \delta_i, \\ &\qquad C_i(t(e_i^j))) - T(t(e_i^j)) \end{aligned}$$

and ($\alpha$) if in the maximum in (5) the first term is valid:

$$\begin{aligned} &\overset{(5,1^{\text{st}}\text{ term})}{=} LC_k(e_k^l) + \mu_{k,i} - T(t(e_i^j)) \\ &\overset{(8)}{\leq} LC_k(e_k^l) - T(t(e_k^l)) \overset{(12)}{\leq} e^+ \end{aligned}$$

and ($\beta$) if in the maximum in (5) the second term is valid:

$$\begin{aligned} &\overset{(5,2^{\text{nd}}\text{ term})}{=} LC_i(e_i^{j-1}) + \delta_i - T(t(e_i^j)) \\ &\overset{(7)}{\leq} LC_i(e_i^{j-1}) - T(t(e_i^{j-1})) \overset{(12)}{\leq} e^+ \end{aligned}$$

and ($\gamma$) if in the maximum in (5) the third term is valid:

$$\overset{(5,3^{\text{rd}}\text{ term})}{=} C_i(t(e_i^j)) - T(t(e_i^j)) \overset{\text{premise}}{\leq} e^+$$

Therefore in all three cases ($\alpha - \gamma$) there is a contradiction to the assumption (11).

Case b) $LC(e_i^j) = LC_i(e_i^j)$ is defined with (6). Then analog to the cases a) ($\beta$) and a) ($\gamma$) the contradiction can be shown.

Therefore in all possible cases a contradiction to the assumption (11) can be shown and therefore the theorem is proved. □

**Theorem 4** *If in a process $i$ the clock $C_i$ is never more than $e_i^-$ slow then the simple logical clock $LC_i$ in this process is also never more than $e_i^-$ slow, i.e.*

$$\forall_i \big[\forall_j\, T(t(e_i^j)) - C(e_i^j) \leq e_i^- \implies \forall_j\, T(t(e_i^j)) - LC(e_i^j) \leq e_i^-\big]$$

*Proof.* $T(t(e_i^j)) - LC(e_i^j) \overset{(9)}{=} T(t(e_i^j) - LC_i(e_i^j))$

$$\overset{(5,6)}{=} T(t(e_i^j)) - \begin{cases} \max(LC_k(e_k^l) + \mu_{k,i},\ LC_i(e_i^{j-1}) + \delta_i, \\ \quad C_i(t(e_i^j))) \quad \text{if} \ldots \\ \max(LC_i(e_i^{j-1}) + \delta_i,\ C_i(t(e_i^j))) \text{ otherwise} \end{cases}$$

$$\overset{-\max(\ldots\text{last term})}{\leq} T(t(e_i^j)) - \begin{Bmatrix} C_i(t(e_i^j)) \\ C_i(t(e_i^j)) \end{Bmatrix} \overset{\text{premise}}{\leq} e_i^- \qquad \square$$

**Observation 1** *There is no symmetry between Theorem 3 and Theorem 4: The being fast of the simple logical clock $LC$ is limited only globally by the maximum of the being fast of all involved clocks, whereas the being slow of $LC$ is limited in each process independently by the minimal being slow of the corresponding $C_i$.*

A modeling of the errors $e_i^-$ and $e^+$ for continuous and discrete clocks can be seen in [26].

## 5. The Controlled Logical Clock

In the Algorithm 1 the term $LC_k(e_k^l) + \mu_{k,i}$ in (5) can cause the logical clock $LC(e_i^j)$ to be set forward compared to $C_i(t(e_i^j))$. At subsequent events in the same process the term $LC_i(e_i^{j-1}) + \delta_i$ in (6) causes that in principle $LC$ **stops** (i.e. it advances only the small ticks $\delta_i$) until it has **fallen back** to the value of $C_i(t(e_i^j))$ that is the last term in (5) and (6)).

Now the simple logical clock from Alg. 1 will be enhanced in two steps to the controlled logical clock in order to run more continuously and to prevent the alternate advancing and stopping.

**Algorithm 2.** $LC'$ is the basis of the controlled logical clock:

$$LC_i'(e_i^j) := \begin{cases} \max(LC_k'(e_k^l) + \mu_{k,i},\ LC_i'(e_i^{j-1}) + \delta_i, \\ \ LC_i'(e_i^{j-1}) + \gamma_i^j(C_i(t(e_i^j)) - C_i(t(e_i^{j-1}))), \\ \ C_i(t(e_i^j))) \quad \text{if } \exists_{e_k^l} (e_k^l, e_i^j) \in M \qquad (13) \\ \max(LC_i'(e_i^{j-1}) + \delta_i, \\ \ LC_i'(e_i^{j-1}) + \gamma_i^j(C_i(t(e_i^j)) - C_i(t(e_i^{j-1}))), \\ \ C_i(t(e_i^j))) \quad \text{otherwise} \qquad (14) \\ \text{and if } j = 0 \text{ then the terms} \\ \ LC_i(e_i^{j-1}) + \delta_i \text{ must be omitted} \end{cases}$$

with $\delta_i$ and $\mu_{k,i}$ defined as in Alg. 1 and with freely selectable $\gamma_i^j \in [0, 1]$. This algorithm will be completed later by the control unit in Alg. 3. The global logical clock is based on the logical clock of the individual processes

$$LC'(e_i^j) := LC_i'(e_i^j) \tag{15}$$

Obviously $LC_{\text{Alg.1}} = LC'_{\text{Alg.2}}$ for $\gamma_i^j \equiv 0$. For $\gamma_i^j \equiv 1$ the definition of $LC'$ is identical with Lamport's piecewise differentiable logical clock $LC$ with $d\,LC_i/dt = d\,C_i/dt$ for the time between receive events.

Theorem 4 – the limitation of being slow – is also valid for $LC'$. The proof is the same. However Theorem 3 – the limitation of being fast – is **not** valid for $LC'$. It is possible to construct examples for any large $e$ and any positive lower limit $\gamma_{min}$ that has for each choice of $\gamma_i^j \geq \gamma_{min}$ an event $e_i^j$ with $LC'(e_i^j)$ being fast more than $e$, i.e. $LC'(e_i^j) - T(t(e_i^j)) > e$. The technical report [26] shows two examples with unlimited error increase.

In most cases – related to the distribution of the clock errors – and with normal applications, that not only communicate but also compute – one can assume that $\gamma_{min} > 0.5$, i.e. that for $\gamma$ values less than 0.5 the logical clock $LC'$ is limited in its being fast.

To limit the being fast of $LC'$ it makes sense to control $\gamma_i^j \in [0, 1]$ by a control loop because normally $\gamma$ values nearby 1 are desirable and also possible. Fig. 2 shows its

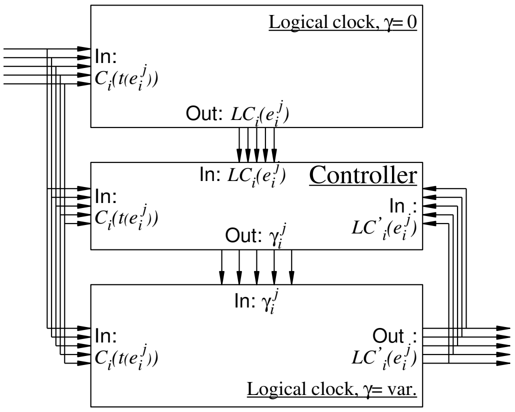



**Figure 2. Control loop for $\gamma_i^j$**

structure. The controller tries to limit the differences between $LC'$ and $T$, i.e. it limits the output error $LC' - T$. The controller must estimate the output error indirectly because $T(t(e_i^j))$ is unknown. The input error $C - T$ can not be determined for the same reason. Provided that the errors of $C_i$ are limited

$$\underset{e_i^j}{\forall} \; -e^- \leq -e_i^- \leq C_i(t(e_i^j)) - T(t(e_i^j)) \leq e_i^+ \leq e^+ \quad (10)$$

the Theorem 3 imply

$$\underbrace{LC_i - C_i}_{\text{measurable}} \leq e^+ + e_i^-$$

and the error of $LC' - T$ is limited by

$$\underbrace{(LC_i' - T)}_{\substack{\text{to be limited}\\ \text{by the controller}}} \leq e_i^+ + \underbrace{(LC_i' - C_i)}_{\text{measurable}}$$

This is the motivation for the following controller algorithm:

**Algorithm 3.** For this algorithm the logical clocks in the Alg. 1 and 2 are calculated stepwise. In each step $s$ new values for $LC_i(e_i^j)$ and $LC_i'(e_i^j)$ are computed[1] for those processes $i \in P_s$ for which the input values are already determined (i.e. $i \notin P_s$, if $\underset{e_k^l}{\exists} (e_k^l, e_i^j) \in M$ and $LC_k(e_k^l)$ and $LC_k'(e_k^l)$ are not computed in a former step). After each step the control variables $\gamma_i^j$ are calculated again for each $i \in P_s$ by the following algorithm:

$$D_{i,0} := q_{init} \quad (16)$$
$$D'_{i,0} := q_{init} \quad (17)$$

[1] $j$ is an abbreviation of $j_{i,s}$ and $j_{i,s}$ is the index of that event, of which the logical clock is computed in process $i$ in step $s$.

$$\begin{aligned} D_{i,s} &:= \max(LC_i(e_i^j) - C_i(t(e_i^j)), \\ &\qquad q_{factor}(D_{i,s-1} - q_{min}) + q_{min}) \quad (18) \\ D'_{i,s} &:= \max(LC_i'(e_i^j) - C_i(t(e_i^j)), \\ &\qquad q_{factor}(D'_{i,s-1} - q_{min}) + q_{min}) \quad (19) \\ \gamma_i^1 &:= \gamma_{max} \quad (20) \end{aligned}$$

$$\gamma_i^{j+1} := \begin{cases} \gamma_i^j \cdot \gamma_{degress} & \text{if } D'_{i,s} > l_{upper} \cdot D_{i,s} \quad (21) \\ \min(\gamma_i^j / \gamma_{degress}, \gamma_{max}) & \text{if } D'_{i,s} < l_{lower} \cdot D_{i,s} \quad (22) \\ \gamma_i^j & \text{otherwise} \quad (23) \end{cases}$$

Explanations:

In each step, new logical clock values are computed for as many processes as possible. The proposed control mechanism computes $\gamma_i^{j+1}$ only on the basis of data belonging to process $i$. Therefore the computation of the logical clock values can be partially parallelized and the computation of the controller can be fully parallelized.

With $q_{init}, q_{min}, q_{factor}, \gamma_{max}, \gamma_{degress}, l_{upper}$ and $l_{lower}$ the controller must be adjusted. Based on the results in Section 6 the following values are recommended for Ethernet or FDDI based clusters: $q_{init} = 250\mu$s, $q_{min} = 250\mu$s, $q_{factor} = 0.9$, $\gamma_{max} = 0.95$, $\gamma_{degress} = 0.9$, $l_{upper} = 2.0$, $l_{lower} = 1.8$.

$D_{i,s}$ is a measurement for the deviation of the simple logical clocks $LC_i$ from the system clocks $C_i$. $D_{i,s}$ computes the maximum of $LC - C$ under *forgetting* older values after some time. $q_{factor} \in (0, 1)$ is the *forget-factor*, see (18). $q_{min}$ in (18) is a lower limit for $D_{i,s}$. It says which clock errors should be tolerated. $q_{min}$ should also be used as start value in $q_{init}$ in (16). $D'_{i,s}$ is defined in the same way as $D_{i,s}$. It measures the deviation of the controlled logical clocks $LC_i'$ from the system clocks $C_i$. $D'_{i,s}$ computes the maximum of $LC' - C$ (see (17) and (19)).

With (21) the controller tries to limit the deviation $D'_{i,s}$ as soon as it exceeds the upper limit ($l_{upper} \cdot D_{i,s}$). By the reduction of $\gamma_i^j$ the behavior of the controlled logical clock becomes stepwise more alike the behavior of the simple logical clock. $\gamma_i^j$ in (22) is stepwise increased as soon as $D'_{i,s}$ comes below the lower limit ($l_{lower} \cdot D_{i,s}$). The upper boundary for this increase is $\gamma_{max}$. $1 - \gamma_{max}$ determines how much the controlled logical clock is slowed down to compensate for being fast.

## 6. Simulation of the Controlled Logical Clock

The efficiency of the controlled logical clock defined by Alg. 2 and 3 was examined by computer simulations. The components of this computer simulation are the following:

(a) the simulated run of a parallel FE-computation on a regular grid produces the set of events $E = \{e_i^j\}$ with the wall clock timestamps $t(e_i^j)$, (b) simulated system clocks with a given error behavior are used to generate differences between $C(t(e_i^j))$ and $t(e_i^j)$, (c) the controlled logical clock computes the values of $LC'(e_i^j)$ and $LC(e_i^j)$, and (d) an analysis module evaluates the controlled logical clock. This section summarizes the most important results. Details can be found in the technical report [26].

### 6.1. The Set of Events

The base is a fictitious FE-computation with $n_1 \times n_2$ processes. Each iteration in each process consists of

1. computation of finite elements at the borders to the neighbors,
2. sending the new results to the neighbors,
3. computing the remaining finite elements,
4. and receiving the new values from the neighbors.

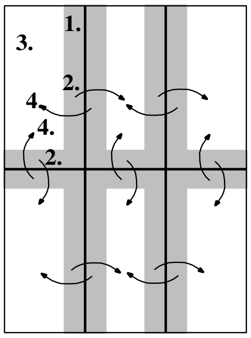



**Figure 3. A fictitious parallel computation**

Execution times and message delays were defined randomly between given limits. These limits were varied in different simulations.

### 6.2. Simulated Clock Errors

$T(t) := t$ can be chosen because the difference between $T$ and $t$ does not influence the outcome of the algorithms. The different clock errors used in the simulations are denoted by **pictograms**. They plot $C_i(t) - t$ against $t$.

Most types are continuous clocks with varying drift rates. The default maximum clock error is $\Delta = 1000\mu$s. And one type is a discrete clock with the tick length default $\tau_l = 1000\mu$s.

### 6.3. The Logical Clock Module

The logical clock module is an implementation of the Algorithms 1, 2 and 3. It needed approximately sizeof(void$*$)$\cdot$ $n^2 + 100 \cdot n$ bytes of memory for its variables and about $\max(6.2/n^{0.0726}, 2.85n^{0.1911})\mu$s execution time for each event on a R8000 processor, i.e. between 5 and 11 $\mu$s for each event if up to 1000 processors are producing the events.

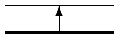
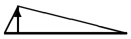
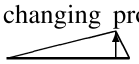
### 6.4. The Analysis

Many observables were examined but only a few main results will be presented here. One advantage of computer simulations is that observables can be analyzed that are not usually accessible in real experiments. In our experiments this is the global time $t$ and all derived observables as $LC' - T$.

For the evaluation the following observables of the controlled logical clock are used: $\overline{\mathcal{F}_{LC'}}$ the averaged being fast of $LC'$, $\overline{\mathcal{S}_{LC'}}$ the averaged being slow of $LC'$, and $\overline{\mathcal{A}_{LC'}}$ the averaged absolute deviation of $LC'$. $\overline{\mathcal{A}_{LC'}}$ is the average over all processes of the sum of $|\vartriangle LC' - \vartriangle T|$ for all the time intervals of two succeeding events and related to the whole execution time. All these observables were also examined for the simple logical clock: $\overline{\mathcal{F}_{LC}}$, $\overline{\mathcal{S}_{LC}}$, and $\overline{\mathcal{A}_{LC}}$. Then criteria for the evaluation were defined as:

- $\overline{\mathcal{F}_{LC'}}/\overline{\mathcal{F}_{LC}}$ should be less than 2, i.e. the controller should limit the being fast to the double of the being fast of the simple logical clock;
- $\overline{\mathcal{S}_{LC'}}/\overline{\mathcal{S}_{LC}}$ should be clearly less than 1, i.e. the controlled logical clock should clearly improve any being slow;
- $\overline{\mathcal{A}_{LC'}}$ is a measurement for the usefulness of the controlled logical clock for performance measurements. It should be less than 5 %.

### 6.5. The Results

Experiments with  and  show that the results are fairly independent of the duration $\tau_{all}$ but strictly dependent on the clock error $\Delta$. Experiments with different drift changing profiles like , , and  show that the results are independent of these curves. Therefore the following experiment is representative of a lot of other experiments: One clock from 20 clocks is constantly 1000 $\mu$s fast.

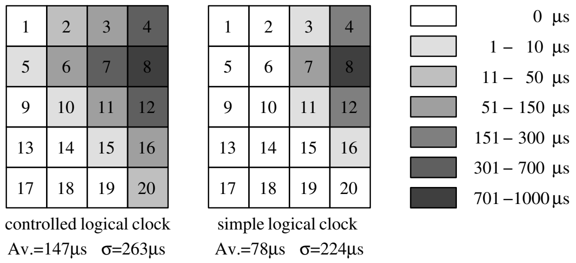



**Figure 4. Average of being fast $\overline{\mathcal{F}_{LC'}}$ and $\overline{\mathcal{F}_{LC}}$**

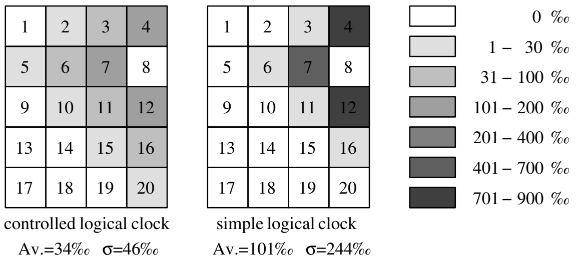



**Figure 5. Average of the absolute deviation $\overline{\mathcal{A}_{LC'}}$ and $\overline{\mathcal{A}_{LC}}$**

Fig. 4 and Fig. 5 represent the absolute results for each process and the averages $\overline{\mathcal{F}}$ and $\overline{\mathcal{A}}$ for both logical clocks $LC'$ and $LC$. Process No. 8 is fast by 1000 $\mu$s all the time. Fig. 4 shows that the controlled logical clock is not really more fast than the simple logical clock, however the controlled logical clock is fast in a wider neighborhood of process No. 8. This is a good sign because the controlled logical clock tries to bring all clocks to the fastest one. Fig. 5 shows that the absolute deviation of the controlled logical clock is clearly better than that of the simple logical clock. In the protocol of the experiment it can be seen that only in six processes $\mathcal{A}_{LC'}$ is greater than 5 % and that the maximum is 13 % whereas the simple logical clock has values of 40 - 82 % in three processes.

The criteria defined in the last section are fulfilled by the controlled logical clock in this experiment.

Another type of experiment is the usage of clocks that are slow. Here the controlled logical clock can compensate up to 65 % of the being slow. The absolute deviation of that process is reduced from $\mathcal{A}_{LC} = 96.1$ % to $\mathcal{A}_{LC'} = 13{,}2$ % ($\overline{\mathcal{A}_{LC'}} = 0.7$ %) and the critical observables are clearly less than the defined criteria.

Experiments with different height $\Delta$ of the clock error show that $\Delta$ should be limited by about the fourfold of the minimum message delay if local time differences in the area of the tenth of the minimum message delay should be measured with an error of less than 5 % in the average over all processes.

The variation of the controller parameters shows that the controller is very stable.

In the last experiment clocks with a clock tick of 1 ms are used. In this case the simple logical clock already provides all benefits. The controlled logical clock does not really improve the results but also does not make them worse.

## 7. Usage of the Controlled Logical Clock

The controlled logical clock is designed for *monitoring* distributed and parallel applications. The controlled logical clock can be used as a filter. A tracefile with backward references, i.e. with timestamps not fulfilling the clock condition, will be modified subsequently. Then the clock condition is guaranteed and in general performance measurements are possible with the new timestamps.

The controlled logical clock is designed primarily to work on events sequentially from the beginning. If it should be used inside of a monitor tool then there is a problem if the tool allows interactive repositioning with subsequent scrolling forward and backward. To solve this, the Algorithms 1, 2 and 3 can be started at each position in a tracefile. For scrolling backward one can use these algorithms but the sign of the timestamps must be changed and the role of the send and receive events must be exchanged. There is one disadvantage: the values of the modified timestamps depend on the choice of the start event, i.e. the user sees the same event with (slightly) different timestamp values.

By assigning two timestamps to each event the monitor can compensate for this disadvantage. The two values are

- the time of the system clocks, normally corrected by algorithms fulfilling Def. 3; it can be used for measurements of time differences inside a process or used to refer to an event;
- the time of the controlled logical clock; it can be used to visualize the events in spacetime diagrams and to measure time differences between two processes; the values depend on the start event chosen after the last repositioning.

It is recommended that an estimate of the minimum transfer delay is obtained as a byproduct of the used synchronization algorithm. The variance of the message delays has a significant region below the most frequent delay values. Our measurements in an Ethernet and an FDDI ring indicate that in general the minimum message delay is larger than a quarter of the most frequent round trip time of an empty remote procedure call.

Additional application areas are possible because the Alg. 1, 2 and 3 can be integrated into parallel applications in the same way as Lamport's logical clock. For this, only the already computed timestamps $LC_k(e_k^l)$ and $LC'_k(e_k^l)$ must be additionally transferred inside the messages $(e_k^l, e_i^j) \in M$. In this way all necessary data required to compute Alg. 1 and 2 is available in each process. The control operation can also be done locally, in which case the step index $s$ must be substituted by the event index $j$.

## 8. Summary

The controlled logical clock is a method of correcting timestamps of a parallel application. It guarantees that the corrected timestamps fulfill the clock condition. Also the being fast of the new timestamps is bounded and the being slow is reduced. The new timestamps are suited for performance measuring and for event visualization in spacetime diagrams. Previously synchronized clocks with a maximum difference of about two milliseconds are necessary. Usually in workstation clusters such synchronization methods are already installed for other purposes. The controlled logical clock is insensitive to a drift jitter of a few percent sometimes used for synchronizing the system clocks. In combination with such synchronization methods an additional synchronization before and after the sampling can be dropped.

Mainly in the case of long execution times the controlled logical clock is better than other methods because these ones assume a little variance of the system clocks' speed.

## 9. Future Work

First tests of the controlled logical clock in practice as a filter for VAMPIR/PARvis [24] monitor tracefiles are done. They show that the $\gamma$ reduction rule (21) can be weakened.

The principle of the current controller limits the value of $\gamma$ to below 1, in the practice to 0.95. For this reason the controlled logical clock often falls back to the value of the system clock $C_i$. It is planed to allow $\gamma = 1$ or at least $\gamma = 1 - 10^{-5}$ with a modified controller. It is planned to modify the controller to achieve by the controlled logical clock $LC'$ a better approximation of the maximum of all system clocks $C_i$.

The simulation has also shown that the clock errors should be limited to about the fourfold of the message passing delay. Reasons and more precise rules for that limitation must be examined.

Additional questions arise if one wants to use the controlled logical clock in systems combing message passing with one-sided communication, i.e. GET and PUT operations directly into the memory of a remote process. In [23] an ATM adapter is described in which the PUTs are implemented mainly in hardware. Besides the problem that normally it is not possible to record a trace event in the remote process, one must study the effects of the different latencies of the message passing and of the one-sided communication.

## References


[1] O. Babaoğlu and R. Drummond. (Almost) no cost clock synchronization. In *Proceedings of 7th International Symposium on Fault-Tolerant Computing*, pages 42–47. IEEE Computer Society Press, July 1987.

[2] F. Cristian and C. Fetzer. Probabilistic internal clock synchronization. In *Proceedings. 13th Symposium on Reliable Distributed Systems, Dana Point, CA, USA, Oct. 25-27, 1994*, pages 22–31. IEEE Computer Society Press, 1994.

[3] F. Cristian and C. Fetzer. Probabilistic internal clock synchronization. Technical Report CS94-367, University of California, San Diego, May 18 1995. ftp://cs.ucsd.edu/pub/team/internalProbClockSync.ps.Z, ftp://cs.ucsd.edu/pub/cfetzer/CS94-367.ps.Z.

[4] J. E. Cuny, A. A. Hough, and J. Kundu. Logical time in visualizations produced by parallel programs. In *Proceedings. Visualization '92, Boston, MA, USA, Oct. 19-23, 1992*, pages 186–193. IEEE Computer Society Press, 1992.

[5] G. v. Dijk and A. v. d. Wal. Partial ordering of sychronization events for distributed debugging in tightly-coupled multiprocessor systems. In A. Bode, editor, *Distributed Memory Computing, 2nd European Conference, EDMCC2, Munich, FRG,* LNCS 487, pages 100–109. Springer-Verlag, April 22-24 1991.

[6] R. Drummond and O. Babaoğlu. Low-cost clock synchronization. *Distributed Computing*, 6(4):193–203, July 1993.

[7] A. Duda, G. Harrus, Y. Haddad, and G. Bernard. Estimating global time in distributed systems. In *Proceedings of the 7th International Conference on Distributed Computing Systems, Berlin, September 21-25, 1987*, pages 299–306. IEEE Computer Society Press, 1987.

[8] T. H. Dunigan. Hypercube clock synchronization. Technical Report ORNL TM-11744, Oak Ridge National Laboratory, TN, February 1991.

[9] T. H. Dunigan. Hypercube clock synchronization. ORNL TM-11744 (updated), September 1994. http://www.epm.ornl.gov/∼dunigan/clock.ps.

[10] D. Edwards and P. Kearns. DTVS: a distribute trace visualization system. In *Proceedings. Sixth IEEE Symposium on Parallel and Distributed Processing, Dallas, Oct. 26-29, 1994*, pages 281–288. IEEE Computer Society Press, 1994.

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

[12] C. J. Fidge. Partial orders for parallel debugging. *ACM SIGPLAN Notices*, 24(1):183–194, January 1989.

[13] D. Haban and W. Weigel. Global events and global breakpoints in distributed systems. In *Proceedings of 21st Hawaii International Conference on System Sciences*, pages 166–175, vol. II, 1988.

[14] R. Hofmann. Gemeinsame Zeitskala für lokale Ereignisspuren. In B. Walke and O. Spaniol, editors, *Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, 7. GI/ITG-Fachtagung, Aachen, 21.-23. September 1993*. Springer-Verlag, Berlin, 1993. ftp://faui79.informatik.uni-erlangen.de/pub/doc/mmb93_globtime.ps.Z.

[15] R. Hofmann. Gesicherte Zeitbezüge für die Leistungsanalyse in parallelen und verteilten Systemen. Dissertation, Universität Erlangen-Nürnberg, Technische Fakultät, 1993. ftp://faui79.informatik.uni-erlangen.de/pub/doc/immd26#3.ps.Z.

[16] J.-M. Jézéquel. Building a global time on parallel machines. In J.-C. Bermond and M. Raynal, editors, *Proceedings of the 3rd International Workshop on Distributed Algorithms,* LNCS 392, pages 136–147. Springer-Verlag, 1989.

[17] J.-M. Jézéquel. *Outils pour l'expérimentation d'algorithmes distribués sur machines parallèles*. PhD thesis, Université de Rennes, 1. Oct. 1989.

[18] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[19] W. S. Lloyd and P. Kearns. Tracing the execution of distributed programs. *Journal of Systems and Software*, 21(3):201–214, June 1993.

[20] E. Maillet and C. Tron. On efficiently implementing global time for performance evaluation on multiprocessor systems. *Journal of Parallel and Distributed Computing*, 28:84–93, 1995.

[21] F. Mattern. Virtual time and global states of distributed systems. In M. Cosnard and P. Quinton, editors, *Proceedings of International Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France, October 1988*, pages 215–226. Elsevier Science Publishers B. V., Amsterdam, 1989.

[22] D. L. Mills. Network time protocol (version 3), specification, implementation and analysis. RFC 1305, Request for Comments, March 1992.

[23] T. Mummert, C. Kosak, P. Steenkiste, and A. Fisher. Fine grain parallel communication on general purpose LANs. In *International Conference on Supercomputing, ACM, Philadelphia*, May 1996. http://www.cs.cmu.edu/afs/cs/project/iwarp/archive/nectar-papers/96ics.ps.

[24] W. E. Nagel and A. Arnold. Performance visualization of parallel programs: The parvis environment. Technical report, Forschungszentrum Jülich, 1995. http://www.kfa-juelich.de/zam/PT/ReDec/SoftTools/PARtools/PARvis.html.

[25] R. L. Probert, H. Yu, and K. Saleh. Relative-clock-based specification and test result analysis of distributed systems. In *Eleventh Annual International Phoenix Conference on Computers and Communications, Scottsdale, AZ, USA, April 1-3, 1992*, pages 687–694. IEEE, New York, 1992.

[26] R. Rabenseifner. Die geregelte logical Clock – Definition, Simulation und Anwendung. Technical Report RUS-30, Rechenzentrum, Universität Stuttgart, Germany, May 1996. http://www.uni-stuttgart.de/People/rabenseifner/log_clock_rus30.html.

[27] M. Raynal. A distributed algorithm to prevent mutual drift between n logical clocks. *Information Processing Letters*, 24:199–202, 1987.

[28] F. B. Schneider. Understanding protocols for byzantine clock synchronization. Technical Report 87-859, Department of Computer Science, Cornell University, August 1987. http://cs-tr.cs.cornell.edu/TR/CORNELLCS:TR87-859/Print.

[29] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: in search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[30] Z. Yang and A. Marsland, T.˙ Annotated bibliography on global states and times in distributed systems. *Operating Systems Review*, 27(3):55–74, July 1993.

[31] M. Zaki, M. El-Nahas, and H. Allam. DPDP: an interactive debugger for parallel and distributed processing. *Journal of Systems and Software*, 22(1):45–61, July 1993.