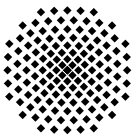

RUS
Rechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. R. Rühle
Allmandring 30, 70550 Stuttgart

ICA II
Institut für Computeranwendungen
Abt. Computersimulation
und Visualisierung

AIM
Anwendungen der Informatik
im Maschinenwesen

FORSCHUNGS- UND ENTWICKLUNGSBERICHTE

DIE GEREGELTE LOGICAL CLOCK MIT VERBESSERTEM REGLER

EINE GLOBALE UHR FÜR TRACEBASIERTES SOFTWARE-MONITORING PARALLELER ANWENDUNGEN IN WORKSTATION CLUSTERN

Rolf Rabenseifner

E-Mail:
rabenseifner@rus.uni-stuttgart.de

ISSN 0941-4665 März 1997 RUS - 38

Alle Rechte vorbehalten.

©1996 by

Rechenzentrum der Universität Stuttgart
Allmandring 30
D-70550 Stuttgart

Eine elektronische Version dieser Veröffentlichung und weitere Informationen finden Sie auf unserem WWW-Server unter folgender URL

`http://www.uni-stuttgart.de/People/rabenseifner/log_clock_rus38.html`

Abstract

Event tracing and monitoring of the program flow and the message exchanges 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 presented. It 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. With a new control algorithm an approximation of the maximum of all local processor clocks is used as global time.

No additional protocol overhead is needed for the new method while tracing the application. It can be implemented as a filter for tracefiles or it can be integrated into monitor and debug tools for parallel applications.

Das Aufzeichnen und Darstellen des Programmflusses und des Message-Austauschs paralleler Anwendungen ist schwierig, wenn jeder Prozessor eine eigene Uhr besitzt und diese Uhren nicht synchronisiert sind. Mehrere Strategien zur Bildung einer globalen Uhrzeit werden in einem Überblick dargestellt, und ihre Grenzen werden aufgezeigt.

Die geregelte logical Clock, eine neue Methode auf der Basis von Lamports Logical Clock, wird vorgestellt. Ungenaue Zeitstempel aus Tracefiles werden derart modifiziert, daß sie einerseits die Clock Condition erfüllen, d.h. daß der Empfang einer Message einen späteren Zeitstempel als das zugehörige Sende-Ereignis besitzt, und daß andererseits Performancemessungen möglich werden, bei denen die Events in verschiedenen Prozessen liegen. Mit einem neuen Regler wird das Maximum aller lokalen Prozessoruhren als Basis für eine globale Zeit angenähert.

Bei der Erstellung eines Tracefiles ist kein zusätzlicher Protokollaufwand nötig. Die geregelte logical Clock kann als Filter für Tracefiles implementiert werden. Sie kann aber auch in Monitor- und Debug-Tools integriert werden.

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

CR Kategorien:

D.1.3 [Concurrent Programming]	Distributed and parallel programming
D.2.5 [Testing and Debugging]	Monitors, Tracing
D.2.6 [Programming Environments]	
D.2.9 [Management]	Time estimation
D.4.8 [Performance]	Monitors, Measurements

Inhaltsverzeichnis

1 Einleitung **2**

2 Logical Clock und Synchronisation **4**

3 Die Clock Condition **6**

4 Lamports Logical Clock **7**

5 Die einfache logical Clock **8**

6 Die geregelte logical Clock **11**

7 Der Regler **13**
7.1 Reglerbedarf 13
7.2 Aufbau des Reglers 14
7.3 Regler $\mathcal{A}$ – eine obere Schranke 14
7.4 Regler $\mathcal{B}$ – eine notwendige Regelung 15
7.5 Regler $\mathcal{C}$ – eine hinreichende Regelung 17
7.6 Zusammenfassung der Regler 18

8 Fehleranalyse **19**

9 Implementierung **22**

10 Test des Algorithmus **23**
10.1 Tracefile einer realen Anwendung 23
10.2 Ein Beispiel mit großem Clocktick 25
10.3 Simulation einer FE-Rechnung 27

11 Anwendung der geregelten logical Clock **28**

12 Zusammenfassung **30**

13 Zukünftige Arbeiten **31**

Kapitel 1

Einleitung

Eine globale Uhrzeit ermöglicht eine optimale Darstellung von lokalen Ereignissen und Send-Receive-Paaren durch Monitortools zur Analyse paralleler und verteilter Anwendungen.

Abb. 1.1 zeigt ein Timeline-Diagramm auf der Basis einer globalen Uhrzeit t. Abb. 1.2 zeigt die gleiche Anwendung, aber basierend auf Zeitstempeln von ungenügend synchronisierten Prozessoruhren C_i. Sie zeigt, daß unter solchen Bedingungen bei der Visualisierung des Programflusses einer parallelen Message-Passing Anwendung die Kausalitäten falsch dargestellt werden, also Mesages revers, d.h. mit negativen Messagedelays gezeichnet werden.

Abb. 1.3 und Abb. 1.4 zeigen, daß es nicht nur genügt, Rückwärtsbezüge bei der Darstellung von Messages zu verhindern. Durch die nachgehende Uhr in der unteren Darstellung wird der Message-Passing Library unterstellt, daß sie im ersten Receive die erste Message nicht annimmt, sondern weiter wartet bis das zweite Send initiiert ist. Beide Beispiele zeigen, daß für das Monitoring eine globale Zeit, vergleichbar mit der physikalischen Zeit, nötig ist.

Auf Systemen, die keine globale Uhr besitzen, gibt es verschiedene Strategien, eine solche anzunähern. Dieser Bericht stellt hierzu ein neues Verfahren vor – die geregelte logical Clock. Sie wurde speziell für Workstationcluster oder vergleichbare Systeme entwickelt und kann, wie in Abb. 1.5 dargestellt, als Filter für Tracefiles eingesetzt werden.

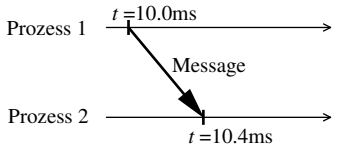

Abb. 1.1: Timeline-Diagramm basierend auf einer globalen Zeit

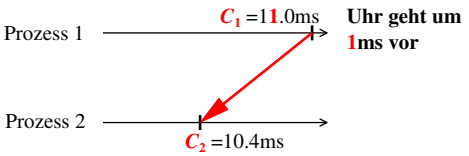

Abb. 1.2: Timeline-Diagramm mit ungenügend synchronisierten Uhren

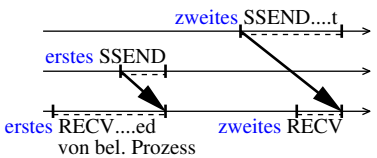

Abb. 1.3: Globales Receive mit korrekter ...

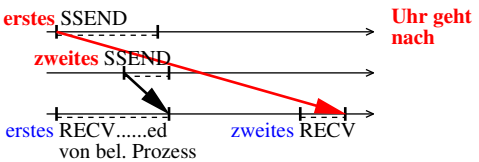

Abb. 1.4: ... und falscher Darstellung

Hierbei werden die für die Zeitstempel eingesetzten Prozessoruhren nachträglich besser synchronisiert, wenn der Tracefile Rückwärtsbezüge enthält. Rückwärtsbezüge sind hierbei Send-Receive-Paare, bei denen das Send-Event einen späteren Zeitstempel als das Receive-Event besitzt. Die geregelte logical Clock modifiziert die Zeitstempel so, daß sie die Clock Condition [Lam78] erfüllen, d.h. ein Receive-Event hat dann grundsätzlich einen späteren Zeitstempel als das zugehörige Send-Event.

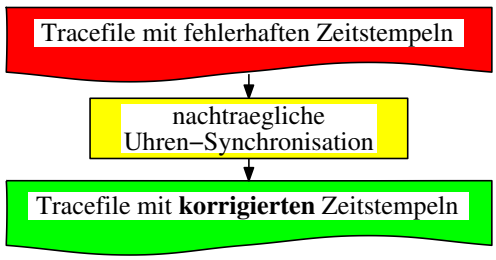

Abb. 1.5: Nachträgliche Korrektur, implementiert als Filter

Ein weiteres Einsatzgebiet sind Debugger. Außer der Erstellung der Zeitstempel und dem Zusammenführen der Traceinformation beim Monitor- oder Debuggingtool bedarf es keines zusätzlichen Protokollaufwands, wie z. B. der Austausch der Zeitstempel beim Messagetransfer selbst, da diese Tools eine vollständige Sicht der Anwendung besitzen.

Mit einem neuen Regler stellt die geregelte logical Clock die auf der Basis der lokalen Prozessoruhren erzeugten Zeitstempel annähernd auf die Zeit der am weitesten vorgehenden Prozessoruhren vor, d.h. eine Approximation des Maximum aller Prozessoruhren wird als globale Zeit verwendet.

Zur Minimierung der Fehler empfiehlt es sich vor dem Start der zu beobachtenden Anwendung eine Uhrensynchronisation durchzuführen. Beste Ergebnisse erhält man, wenn man außerdem nach Abschluß der Anwendung die Uhren ebenfalls synchronisiert und eine lineare Interpolation für die Dauer der Anwendung nachträglich durchführt, d.h. der geregelten logical Clock vorschaltet.

Die geregelte logical Clock läßt die Zeitstempel der System Clocks C_i unverändert, solange die Zeitstempel von Receive-Events später als die der zugehörigen Send-Events zuzüglich der minimalen Übertragungszeit sind. Die geregelte logical Clock LC_i ist eine Modifikation von Lamports stückweise differenzierbarer logical Clock mit $dLC_i(t)/dt := dC_i(t)/dt$, wie sie auch in [Jez89a] beschrieben ist, doch im Gegensatz zu dieser ist bei der geregelten logical Clock die Abweichung $LC - C$ beschränkt. Sie ist damit zur Darstellung von Causal-Bezügen geeignet, als auch zur Messung von durchschnittlichen Zeitdifferenzen zwischen Ereignissen in verschiedenen Prozessen. Für Zeitdifferenzen innerhalb eines Prozesses ist zwar die direkte Zeitmessung geeigneter, die Zeitstempel der geregelten logical Clock können aber trotzdem, z.B. für Vergleiche mit Transferzeiten, eingesetzt werden, da sie in der Regel im Durchschnitt um weniger als 1 % abweichen, und die meisten Monitortools nur eine einheitliche Zeitskala unterstützen.

Kap. 2 gibt einen Überblick über grundlegende Arbeiten in diesem Gebiet, Kap. 3 bis 7 motivieren und definieren die einfache und die darauf aufbauende geregelte logical Clock und in Kap. 8 werden ihre Fehler analysiert. In Kap. 9 wird die Implementierung beschrieben und deren Einsatz anhand dreier Beispiele in Kap. 10 untersucht. Ihre generelle Anwendung wird in Kap. 11 erörtert und in Kap. 12 werden die Ergebnisse zusammengefaßt. Kap. 13 gibt einen Ausblick auf zukünftige Arbeiten, z.B. durch die Uhrenanpassung mittels einer rückwärtsgerichteten linearen Amortisation lassen sich die derzeitigen Korrekturfehler noch deutlich reduzieren.

Kapitel 2

Logical Clock und Synchronisation

Bei der Darstellung von Events zum Debugging und für Performancemessungen gibt es unterschiedliche Ansätze zur Synchronisation der lokalen Uhren. Lamport's diskrete Logical Clock [Lam78] kann direkt zum Monitoring eingesetzt werden [CHK92, ZEA93]. Raynal [Ray87] beschreibt hierzu ein Verfahren zur Begrenzung der Abweichung der Logical Clocks der Prozesse untereinander. Die Vektor Clock, eine Erweiterung von Fidge [Fid88, Fid89] und Mattern [Mat89], ermöglicht eine äquivalente Darstellung der durch die Send-Receive-Paare gegebenen Causalbeziehung mittels Monitortools [DW91, LK93, EK94]. Weitere Verbesserungen, z.B. zum Memory-Bedarf, werden in [CB91, MSV91, SK90, SES89, Che92, BDM+94, RS95] aufgezeigt. In [HW88] wird der Begriff der globalen Events eingeführt und in [PYS92] werden zusätzlich spontane Ereignisse, wie z.B. Kollisionen, berücksichtigt. In der Zusammenfassung [SM94] werden aber auch die Grenzen der diskreten logical Clock und der Vektor Clock beleuchtet.

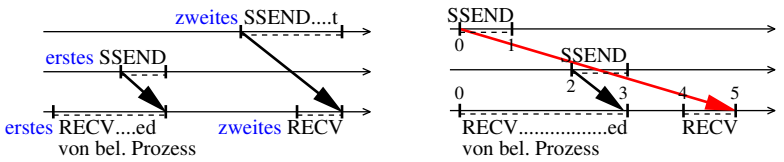

Abb. 2.1: Globales Receive dargestellt mittels Realtime und diskreter logical Clock

Die rechte Darsteluung in Abb. 2.1 zeigt eine weitere Beschränkung der Anwendbarkeit der logical Clock auf (s.a. Abb. 1.3 und 1.4). Hierbei wurden im mittleren Prozeß zu Beginn zwei zusätzliche lokale Ereignisse angenommen. Wie Abb. 1.4 erweckt auch die Darstellung mittels der logical Clock den Eindruck, daß das erste globale Receive die falsche Message empfängt. Dieses Beispiel zeigt damit, daß die logical Clock für die Darstellung von Monitorinformationen nicht ausreichend ist.

Eine Alternative hierzu ist, durch hinreichend genaue Synchronisierung der einzelnen Uhren eine Darstellung ohne Rückwärtsbezüge zu erreichen [Lam78, Hof93b, MT95]. Eine exakte Synchronisation am Anfang inklusive Driftbestimmung [Dun91, Dun94] oder besser am Anfang und Ende [MT95] mit einem linearen Ausgleich während der Dauer der Anwendung sind die in softwarebasierten Monitortools [RTV93] häufig angewandten Methoden. Für die genaue Synchronisation am Anfang und Ende sind deterministische [Sch87] und probabilistische [CF94, CF95] Verfahren geeignet. Eine kontinuierliche Uhrensynchronisation mit

geringer Resourcenbelastung, wie z. B. mit xntp [Mil92], ist normalerweise aufgrund der stark variierenden Messagedelays in einem lokalen Netz nicht ausreichend exakt.

Eine Alternative ist die tracebasierte Synchronisierung. Auf der Basis eines Tracefiles und der Prämisse, daß ein Receive-Event frühestens um die minimale Transferzeit nach dem Send-Event ankommen darf, werden die Uhrenabweichungen ermittelt.

Duda [DHHB87] entwickelte hierzu zwei Verfahren mittels Regressionsanalyse, bzw. mittels der Bildung einer konvexen Hülle. Jezequel [Jez89a] erweiterte diese Verfahren für eine beliebige Prozessor-Topologie mittels eines *minimal spawning tree* Algorithmus und Hofmann [Hof93a] verbesserte diese durch eine vereinfachte Minimum/Maximum-Bildung und schlug eine Aufteilung der Gesamtzeit in Teilintervalle vor, damit auch Drifttänderungen der Uhren bei lang laufenden Anwendungen ausgeglichen werden können. Abb. 2.2 skizziert den Algorithmus. Das Ergebnis ist eine Approximation der Uhrendifferenzen in Abhängigkeit von der Zeit. In dem unteren Diagramm werden die Transferzeiten basierend auf den Prozessoruhren wie angegeben als Punkte (x) eingetragen und anschließend der punktfreie Parallelstreifen dazwischen gebildet. Dessen Mitte approximiert die Uhrendifferenz.

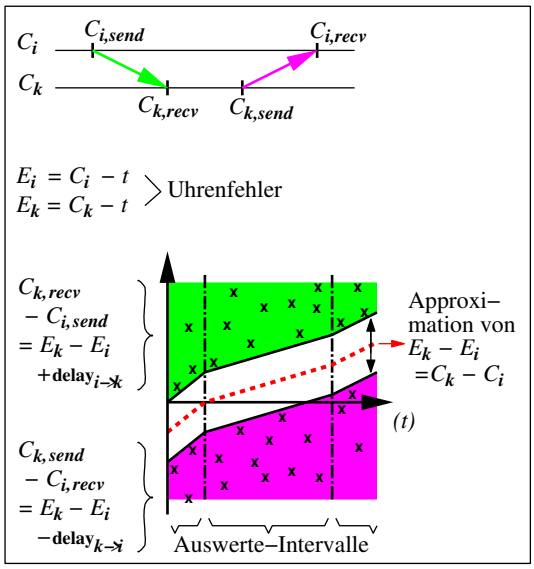

Abb. 2.2: Algorithmus von Duda, erweitert von Hofmann, und mit stetiger Approximation

Babaoğlu und Drummond [BD87, DB93] zeigen, daß eine fast kostenlose Synchronisierung möglich ist, wenn die Anwendung hinreichend oft einen vollständigen Messageaustausch zwischen den Prozessen durchführt. Einen Überblick über weitere Arbeiten findet man in [YM93]. Die Grenzen dieser Methoden sind bestimmt durch die Varianz der Messagedelays, den nicht linearen Zusammenhang von Messagedelay und Messagelänge, sowie durch die teilweise einseitige Kommunikationstopologie von Anwendungen (z.B. Producer/Consumer Szenarien).

Die geregelte logical Clock ist eine neuartige Weiterentwicklung von Lamports Logical Clock in beiden Varianten, der diskreten und der mit $dLC_i(t)/dt := dC_i(t)/dt$ teilweise stetigen. Sie enthält aber auch Komponenten der tracebasierten Synchronisierung, da sie im Rahmen einer Traceauswertung oder eines Debuggingtools keinen zusätzlichen Protokolloverhead benötigt, sondern eine nachträgliche Synchronisation anhand der vorhandenen Zeitstempel und Kommunikationsbeziehungen durchführt.

Im Gegensatz zur tracebasierten Synchronisation kann die geregelte logical Clock auch bei Uhren, deren Granularität größer als das minimale Messagedelay ist, angewandt werden.

Während oben genannte Verfahren meist eine Approximation des *Durchschnitts* der Prozessoruhrzeiten als globale Zeit verwenden, wird bei der geregelten logical Clock eine Approximation des *Maximums* realisiert. Diese Arbeit ist eine Erweiterung von [Rab96, Rab97] und erzielt wesentlich geringere Fehler und bedarf nicht mehr der Beschränkung der Uhrenabweichungen auf das 4-fache des minimalen Messagedelays.

Kapitel 3

Die Clock Condition

Problem: Für das Monitoring benötigt man Uhren, die einerseits für Performance-Messungen möglichst genau gehen sollten, und die andererseits für die Darstellung des Programmablaufs die nachfolgend definierte Clock Condition [Lam78] erfüllen sollten.

Definition 1: Es sei n die Anzahl der Prozesse,
e_i^j das j-te Ereignis im Prozeß i,
$E = \{e_i^j | i = 1..n, j = 0..j_{\max}(i)\}$ die Menge der Ereignisse, und
$M = \{(e_k^l, e_i^j) | e_k^l$ ist ein Sende-Ereignis und e_i^j ein zugehöriges Receive-Ereignis $\}$ ist die Menge der Send-Receive-Paare.
e_i^j ist ein internes Ereignis wenn es kein Sende- oder Receive-Ereignis ist.

Für zwei Ereignisse e_k^l, e_i^j gilt die Relation e_k^l *happened directly before* e_i^j, bzw. $e_k^l \tilde{\rightarrow} e_i^j$, wenn
(a) $(e_k^l, e_i^j) \in M$, oder (3.1)
(b) die Ereignisse im gleichen Prozeß aufeinanderfolgen, d.h. $k = i \wedge l = j - 1$. (3.2)

Die Relation *happened before*, bzw. $\rightarrow$, ist die transitive Hülle der Relation $\tilde{\rightarrow}$, d.h. die kleinste Relation, für die zusätzlich gilt:
(c) $e_k^l \rightarrow e_i^j \wedge e_i^j \rightarrow e_n^m \implies e_k^l \rightarrow e_n^m$ (3.3)

Definition 2: Eine Uhr $\mathcal{C} : E \mapsto \mathbb{R}$ erfüllt die *Clock Condition* wenn

$$\underset{e_k^l \in E, e_i^j \in E}{\forall} e_k^l \rightarrow e_i^j \implies \mathcal{C}(e_k^l) < \mathcal{C}(e_i^j) \tag{3.4}$$

Unter Vernachlässigung der Drift läßt sich folgender Satz einfach zeigen:

Satz 1 *Wenn die Abweichung der Prozessor-Uhren voneinander geringer ist als das kleinste Message-Delay, dann erfüllen die Prozessor-Uhren die Clock Condition.*

Eine Präzisierung für theoretisch stetige Uhren mit beschränktem Drift findet man in [Jez89b, MT95]. In [Hof93b] werden reale Uhren mit diskreten Sprüngen (Clocktick) untersucht. Dieser Bericht untersucht den Fall, daß die Prämisse dieses Satzes nicht erfüllt ist.

Kapitel 4

Lamports Logical Clock

Lamports Logical Clock ist die Basis dieser Arbeit. Sie wird daher hier kurz vorgestellt: C_i sei die Processoruhr des Prozesses i.

Algorithmus 1: Lamports Logical Clock $\boldsymbol{LC_i^{int}}$ ist definiert mit

$$LC_i^{int}(e_i^j) := \begin{cases} 0 \quad \text{für } j = 0 \\ LC_i^{int}(e_i^{j-1}) + 1 \quad \text{für interne und Sende-Ereignisse} \\ \max(LC_i^{int}(e_i^{j-1}) + 1, LC_k^{int}(e_k^l) + 1) \quad \text{wenn } \underset{e_k^l}{\exists} (e_k^l, e_i^j) \in M, \\ \qquad \text{d.h. wenn } e_i^j \text{ ein Receive-Ereignis ist und } e_k^l \text{ das} \\ \qquad \text{zugehörige Sende-Ereignis ist} \end{cases} \tag{4.1}$$

Und die globale Logical Clock ist definiert als

$$LC^{int}(e_i^j) := LC_i^{int}(e_i^j)$$

Abb. 4.1 zeigt ein Beispiel. Die blauen Ziffern zeigen hierbei an, daß in (4.1) der rechte Term maßgeblich war, d.h. daß die Logical Clock zusätzlich vorgestellt werden mußte damit keine Rückwärtsbezüge auftreten. Lamports Logical Clock erfüllt die Clock Condition, d.h. Rückwärtsbezüge (die roten Pfeile im untersten Timeline-Diagramm) werden verhindert, wie das mittlere Timeline-Diagramm zeigt. Lamports Logical Clock eignet sich aber nicht für Performance-Messungen, da sie keinen Bezug zur physikalischen Zeit besitzt. Außerdem, wie in Abb. 2.1 aufgezeigt, können auch globale Wartepunkte (z.B. `MPI_RECV(...MPI_ANY_SOURCE...)`, [MPI95]) nicht adäquat dargestellt werden.

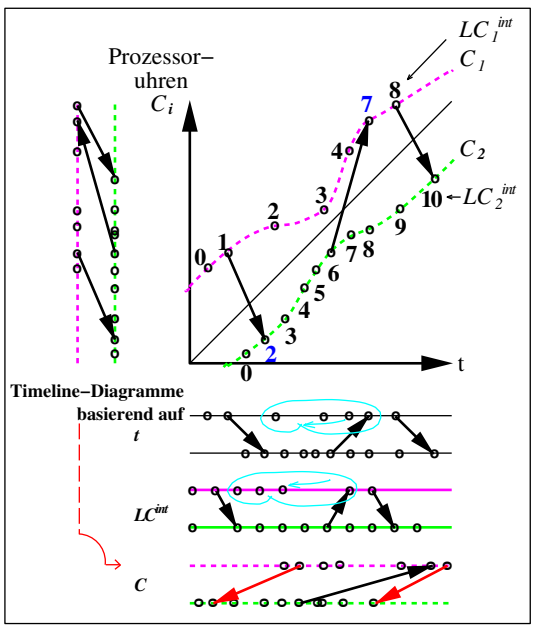

Abb. 4.1: Lamports Logical Clock

Kapitel 5

Die einfache logical Clock

Die einfache logical Clock ist eine Erweiterung von Lamports Logical Clock. Der Name *einfach* wurde zur Unterscheidung von der darauf aufbauenden *geregelten* logical Clock gewählt.

Definition 3: t sei die physikalische Uhrzeit, $T(t)$ sei eine globale Zeit zu der die Prozessoruhren $C_i(t)(i = 1..n)$ mit einer begrenzten Abweichung synchronisiert sind, d.h. es existieren Konstanten $-e_i^-$ und e_i^+ mit $-e_i^- \leq C_i(t) - T(t) \leq e_i^+$.

Die einfache logical Clock LC wird folgendermaßen gebildet: Normalerweise läuft sie wie C_i, d.h. $LC_i := C_i$. Beim Empfangen eines Events wird sie auf max ($LC_i, \mu + LC_k$ des Senders beim Absenden) vorgestellt, mit μ = minimales Messagedelay, das z.B. bei einer präzisen Synchronisation vor Beginn der Anwendung ermittelt werden kann. Wenn die logical Clock auf der Basis dieser Regel vorgestellt worden ist, dann bleibt sie anschließend nahezu stehen solange sie gegenüber der lokalen Prozessoruhr noch vorgeht, d.h. solange $LC_i > C_i$. Genaugenommen wird sie hierbei jeweils bei jedem Ereignis um ein kleines δ vorgestellt. Es handelt sich hierbei um eine Modifikation der von Lamport [Lam78] entwickelten Logical Clock. In Abb. 5.1 wird sie anhand des vorhergehenden Beispiels veranschaulicht.

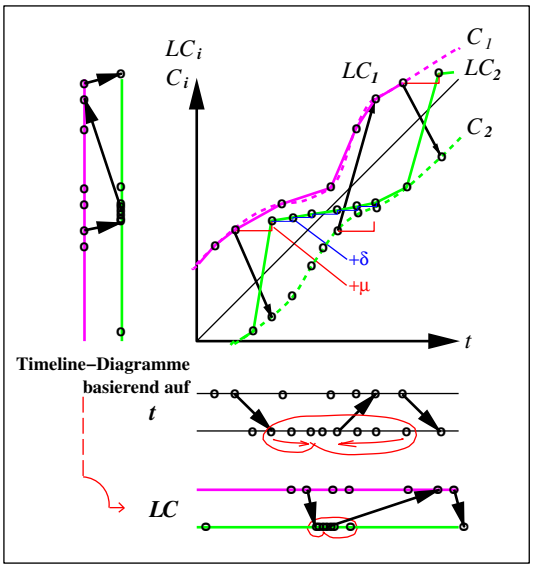

Abb. 5.1: Einfache logical Clock

Algorithmus 2: Die exakte Definition der **einfachen logical Clock *LC*** ist:

$$LC_i(e_i^j) := \begin{cases} \max(LC_k(e_k^l) + \mu_{k,i},\ LC_i(e_i^{j-1}) + \delta_i,\ C_i(t(e_i^j))) & (5.1) \\ \quad \text{wenn } \exists\limits_{e_k^l} (e_k^l, e_i^j) \in M & \\ \max(LC_i(e_i^{j-1}) + \delta_i,\ C_i(t(e_i^j))) \quad \text{andernfalls} & (5.2) \\ \text{wobei die Terme } LC_i(e_i^{j-1}) + \delta_i \text{ bei } j = 0 \text{ entfallen} & \end{cases}$$

mit
δ_i = minimaler Abstand zweier Events in Prozeß i, d.h. δ_i sind Konstanten für die gelten

$$\delta_i > 0 \quad \wedge \quad \underset{i,j}{\forall} T(t(e_i^j)) - T(t(e_i^{j-1})) \geq \delta_i \tag{5.3}$$

$\mu_{k,i}$ = minimales Delay einer Message von Prozeß k nach Prozeß i, d.h. $\mu_{k,i}$ sind Konstanten für die gelten

$$\mu_{k,i} > 0 \quad \wedge \quad \underset{(e_k^l, e_i^j) \in M}{\forall} T(t(e_i^j)) - T(t(e_k^l)) \geq \mu_{k,i} \tag{5.4}$$

Die globale logische Uhr ist definiert als

$$LC(e_i^j) := LC_i(e_i^j) \tag{5.5}$$

Satz 2 *Die einfache logical Clock LC erfüllt die* **clock condition**.

Beweis: Der Algorithmus 2 genügt Lamports Regeln IR1 und IR2 in [Lam78] und erfüllt damit die Clock Condition.

□

Im folgenden wird der Fehler ausgehend von C_i modelliert:

$$\underset{e_i^j}{\forall} t_0 < t(e_i^j) < t_e \ \wedge \ \underset{t\,:\,t_0<t<t_e}{\forall} -e^- \leq -e_i^- \leq C_i(t) - T(t) \leq e_i^+ \leq e^+ \tag{5.6}$$

d.h. die Prozessoruhren gehen gegenüber der globalen Zeit um maximal e^-, bzw. e_i^- nach und um maximal e^+, bzw. e_i^+ vor.

Satz 3 *Wenn alle Uhren C_i nie mehr als e^+ vorgehen, dann geht auch die einfache logical Clock LC nie um mehr als e^+ vor, d.h.*

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

Beweis: Angenommen es gibt ein (i, j) mit

$$LC(e_i^j) - T(t(e_i^j)) > e^+ \tag{5.7}$$

und o. B. d. A. sei e_i^j das früheste derartige Ereignis. (5.8)

Fall a) $LC(e_i^j) = LC_i(e_i^j)$ sei definiert mittels (5.1). Dann gilt

$$\begin{aligned} LC(e_i^j) - T(t(e_i^j)) &\overset{(5.5)}{=} LC_i(e_i^j) - T(t(e_i^j)) \\ &\overset{(5.1)}{=} \max(LC_k(e_k^l) + \mu_{k,i},\, LC_i(e_i^{j-1}) + \delta_i, \\ &\qquad C_i(t(e_i^j))) \ - \ T(t(e_i^j)) \end{aligned}$$

und (α) falls beim Maximum in (5.1) der erste Fall zutrifft:

$$\overset{(5.1,1.\text{Ausdruck})}{=} LC_k(e_k^l) + \mu_{k,i} - T(t(e_i^j)) \overset{(5.4)}{\leq} LC_k(e_k^l) - T(t(e_k^l)) \overset{(5.8)}{\leq} e^+$$

und (β) falls beim Maximum in (5.1) der zweite Fall zutrifft:

$$\overset{(5.1,2.\text{Ausdruck})}{=} LC_i(e_i^{j-1}) + \delta_i - T(t(e_i^j)) \overset{(5.3)}{\leq} LC_i(e_i^{j-1}) - T(t(e_i^{j-1})) \overset{(5.8)}{\leq} e^+$$

und (γ) falls beim Maximum in (5.1) der dritte Fall zutrifft:

$$\overset{(5.1,3.\text{Ausdruck})}{=} C_i(t(e_i^j)) - T(t(e_i^j)) \overset{\text{Prämisse}}{\leq} e^+$$

D.h. in allen drei Unterscheidungen ($\alpha - \gamma$) liegt ein Widerspruch zur Annahme (5.7) vor.

Fall b) $LC(e_i^j) = LC_i(e_i^j)$ sei definiert mittels (5.2). Dann gilt

$$\begin{aligned} LC(e_i^j) - T(t(e_i^j)) &\overset{(5.5)}{=} LC_i(e_i^j) - T(t(e_i^j)) \\ &\overset{(5.2)}{=} \max(LC_i(e_i^{j-1}) + \delta_i,\, C_i(t(e_i^j))) - T(t(e_i^j)) \\ &\overset{\text{Falla)}(\beta),(\gamma)}{\leq} e^+ \end{aligned}$$

ebenfalls im Widerspruch zur Annahme (5.7).

Da in allen möglichen Fällen ein Widerspruch zur Annahme vorliegt, ist der Satz bewiesen.

□

Satz 4 *Wenn in einem Prozeß i die Uhr C_i nie mehr als e_i^- nachgeht, dann geht auch die einfache logical Clock LC in diesem Prozeß nie um mehr als e_i^- nach, d.h.*

$$\underset{i}{\forall}\, [\, \underset{j}{\forall}\, T(t(e_i^j)) - C(e_i^j) \leq e_i^- \implies \underset{j}{\forall}\, T(t(e_i^j)) - LC(e_i^j) \leq e_i^- \,]$$

Beweis:

$$\begin{aligned} & T(t(e_i^j)) - LC(e_i^j) \overset{(5.5)}{=} T(t(e_i^j) - LC_i(e_i^j)) \\ & \overset{(5.1,5.2)}{=} T(t(e_i^j)) - \left\{ \begin{array}{ll} \max(LC_k(e_k^l) + \mu_{k,i},\, LC_i(e_i^{j-1}) + \delta_i,\, C_i(t(e_i^j))) & \text{für ...} \\ \max(LC_i(e_i^{j-1}) + \delta_i,\, C_i(t(e_i^j))) & \text{andernfalls} \end{array} \right\} \\ & \overset{-\max(\ldots\text{letzterTerm})}{\leq} T(t(e_i^j)) - \left\{ \begin{array}{l} C_i(t(e_i^j)) \\ C_i(t(e_i^j)) \end{array} \right\} \\ & \overset{\text{Prämisse}}{\leq} e_i^- \end{aligned}$$

□

Bemerkung 1 *Die beiden Sätze 3 und 4 sind nicht* symmetrisch *zueinander: Das Vorgehen der einfachen logical Clock LC ist nur global durch das Maximum des Vorgehens aller beteiligten Uhren begrenzt, während das Nachgehen von LC innerhalb eines jeden Prozesses durch das maximale Nachgehen der jeweiligen Prozessoruhr begrenzt ist.*

(Beweis durch entsprechende Wahl von $T := C_i$ bzw. $\max_k C_k$ und $e_i^- := 0$ bzw. $e^+ := 0$.)

Kapitel 6

Die geregelte logical Clock

Im Algorithmus 2 bewirkt der Term $LC_k(e_k^l) + \mu_{k,i}$ in (5.1) ggf. ein Vorstellen der logical Clock $LC(e_i^j)$ gegenüber der Uhr $C_i(t(e_i^j))$. Bei nachfolgenden Ereignissen im gleichen Prozeß bewirkt der Term $LC_i(e_i^{j-1}) + \delta_i$ in (5.1) und (5.2), daß LC im Prinzip **stehenbleibt** (bzw. – genau gesagt – um eine kleine Zeitspanne δ_i vorgestellt wird), bis sie dann wieder *zurückgefallen* ist auf den Wert $C_i(t(e_i^j))$ (siehe Term $C_i(t(e_i^j))$ in (5.1) und (5.2)).

Damit die logical Clock gleichmäßiger gehen kann, bzw. um das abwechselnde Vorstellen und Stehenbleiben derselben zu verhindern, wird die einfache logical Clock aus Algorithmus 2 folgendermaßen erweitert:

Algorithmus 3: LC' bildet die Basis für die geregelte logical Clock:

$$LC_i'(e_i^j) := \begin{cases} \max(LC_k'(e_k^l) + \mu_{k,i},\ LC_i'(e_i^{j-1}) + \delta_i, \\ \qquad LC_i'(e_i^{j-1}) + \gamma_i^j (C_i(t(e_i^j)) - C_i(t(e_i^{j-1}))), \\ \qquad C_i(t(e_i^j))) \quad \text{wenn } \exists\limits_{e_k^l} (e_k^l, e_i^j) \in M & (6.1) \\ \max(LC_i'(e_i^{j-1}) + \delta_i,\ LC_i'(e_i^{j-1}) + \gamma_i^j (C_i(t(e_i^j)) - C_i(t(e_i^{j-1}))), \\ \qquad C_i(t(e_i^j))) \quad \text{andernfalls} & (6.2) \\ \text{wobei die Terme } LC_i(e_i^{j-1}) + \delta_i \text{ bei } j = 0 \text{ entfallen} \end{cases}$$

mit δ_i und $\mu_{k,i}$ wie in Algorithmus 2 und mit frei wählbaren $\gamma_i^j \in [0, 1]$.

Die globale logische Uhr wird wie in Algorithmus 2 (5.5) ebenfalls aus den logical Clocks der einzelnen Prozesse gebildet:

$$LC'(e_i^j) := LC_i'(e_i^j) \tag{6.3}$$

Der Algorithmus wird mit dem Regler in Kap. 7 später vervollständigt.

Offensichtlich ist $LC_{\text{Alg.2}} = LC'_{\text{Alg.3}}$ für $\gamma_i^j \equiv 0$. Für $\gamma_i^j \equiv 1$ entspricht LC' der Definition von Lamport [Lam78] für eine stückweise differenzierbare Logical Clock LC mit $dLC_i/dt = dC_i/dt$ für alle Zeitpunkte zwischen Receive-Events.

Abb. 6.1 zeigt die Wirkungsweise dieses Algorithmus für $\gamma \equiv 1$. Durch die erste Message wird die C_2-Kurve parallel nach oben verschoben (d.h. vorgestellt); das ergibt LC_2'. Da C_1 die am weitesten vorgehende Uhr ist, wird sie nicht noch weiter vorgestellt, d.h. $LC_1' := C_1$. Damit verlaufen nun LC_1' und LC_2' nahezu gleich, abgesehen von kleinen Driftschwankungen. Diese Driftschwankungen können aber dazu führen, daß LC_2' weiter als die schnellste Prozessoruhr (hier C_1) vorgeht, siehe ☺.

Dies zu verhindern ist die Aufgabe des in den nächsten Kapiteln beschriebenen Reglers. Außerdem hat man zu Beginn (siehe) starke Fehler in der Darstellung solange die Uhrenangleichung mangels entsprechender Messages noch nicht durchgeführt ist. Dies kann am einfachsten dadurch verhindert werden, daß vor dem Start der zu beobachtenden Anwendung sämtliche Prozessoruhren synchronisiert werden. Das untere Timeline-Diagramm basiert auf der geregelten logical Clock und zeigt gegenüber der darüberliegenden Darstellung, die auf der physikalischen globalen Zeit basiert, abgesehen von den soeben beschriebenen Problemen nur noch geringe Abweichungen in der Darstellung.

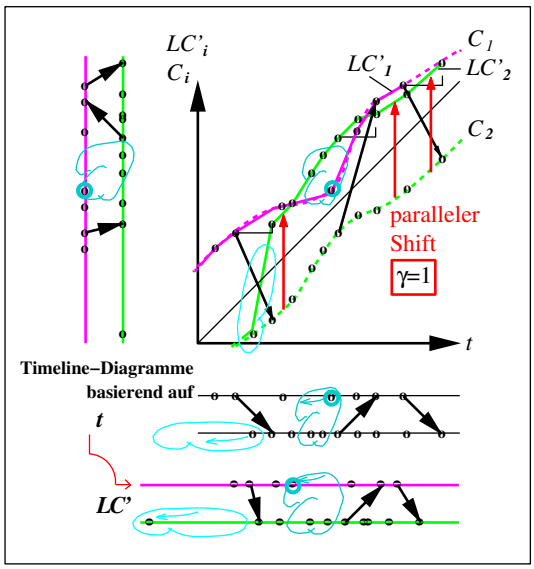

Abb. 6.1: Die geregelte logical Clock

Abb. 6.2 stellt den gleichen Sachverhalt dar, aber auf der Ordinate sind nicht die Uhren (C, LC), sondern ihr Vorgehen $(C-t, LC-t)$ aufgetragen. Als Beispiel dienen hier Prozessoruhren, die konstant vor- bzw. nachgehen. Die Wirkungsweise der geregelten logical Clock und der Einfluß realer Messagedelays und des Faktors γ kann man hier sehen:

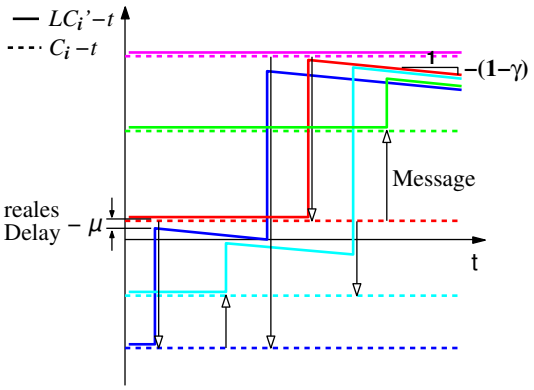

Abb. 6.2: Die geregelte logical Clock – eine andere Darstellung

- LC_i' wird bei einem Receive (bis auf das reale Messagedelay abzüglich μ) auf das Niveau der LC_k' des Senders angehoben, wenn dessen Uhr entsprechend stärker vorgeht,
- die angehobenen Kurven $LC_i' - t$ fallen um die Steigung $-(1-\gamma)$ gegenüber einer gedachten Parallele von $C_i - t$.

Außerdem sieht man, daß wenn es Messageketten von dem Prozeß mit der am weitesten vorgehenden Prozessoruhr zu jedem anderen Prozeß gibt, und deren Messagedelays in der Nähe von μ liegen, und wenn $\gamma \cong 1$, dann gilt für die geregelten Clocks aller Prozesse

$$LC_i' \cong \max_{k=1..n}(C_k)$$

und somit auch global

$$LC' \cong \max_{k=1..n}(C_k)$$

Hieraus ergibt sich andererseits, daß der Regler den Wert von γ nahe 1 belassen sollte.

Kapitel 7

Der Regler

7.1 Reglerbedarf

Der Regler soll verhindern, daß die geregelte logical Clock weiter vorgeht als die am weitesten vorgehende Prozessoruhr. Dies kann dadurch verhindert werden, daß γ_i^j reduziert wird, sobald bei einem Prozessor die geregelte logical Clock LC_i' weiter vorgeht als alle Prozessoruhren, d.h. sobald

$$\exists_i LC_i' > \max_{k=1..n}(C_k)$$

Dies kann vorkommen, wenn

$$\rho_{\text{am weitesten vorgehende Uhr}} < \rho_{\text{beliebige Uhr}} - (1-\gamma)$$

mit

$$\rho = dC/dt - 1 \quad \text{(Drift)}$$

Abb. 7.1 veranschaulicht dies auf der linken Seite.

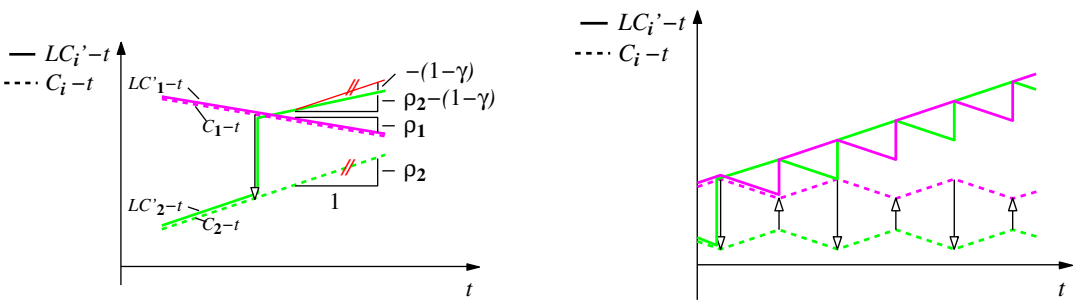

Abb. 7.1: Unbeschränktes Vorgehen der geregelten logical Clock bei zu großem γ

Während der Satz 4 über die Beschränktheit des Nachgehens ebenfalls für LC' gilt[1], kann man zeigen, daß der Satz 3 über die Beschränktheit des Vorgehens für LC' nicht gilt. Man kann sogar zu jeder Wahl eines Wertes e^+ und einer beliebig kleinen positiven unteren Schranke für γ_i^j ein Beispiel konstruieren, indem für ein e_i^j die logical Clock LC' mehr als e^+ vorgeht. Die rechte Seite von Abb. 7.1 zeigt ein Beispiel. Weitere Beispiele findet man in [Rab96]. Da aber normalerweise γ-Werte im Bereich nahe 1 wünschenswert sind, ist es sinnvoll, $\gamma_i^j \in [0,1]$ über einen Regelkreis zu steuern.

[1] Der Beweis kann direkt übernommen werden

7.2 Aufbau des Reglers

Abb. 7.2 zeigt seinen prinzipiellen Aufbau.

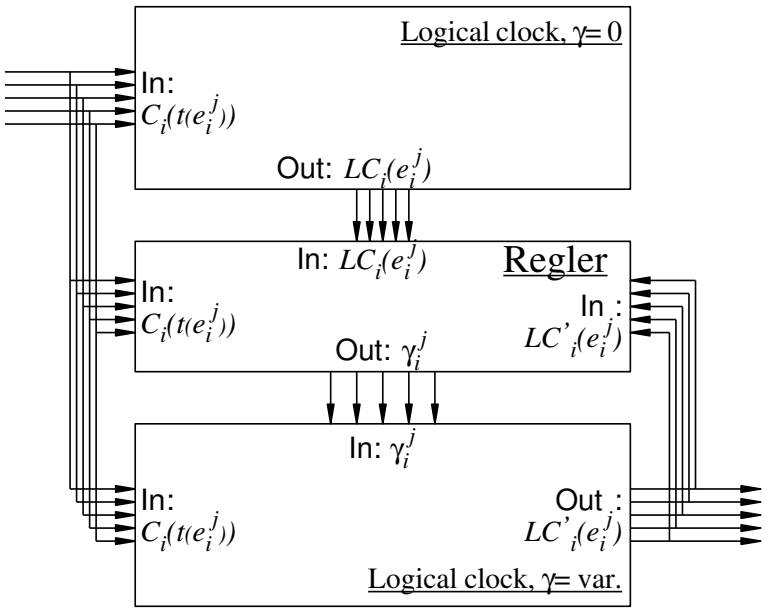

Abb. 7.2: Regelkreis für γ_i^j

Der Regler soll verhindern, daß LC' weiter vorgeht als die am weitesten vorgehende Prozessoruhr, d.h. als $\max_{k=1..n}(C_k)$. Da aber die Abbildung $t \mapsto C_k(t)$ unbekannt ist, kann dieses Kriterium nicht erfaßt werden. Daher sind andere Kriterien nötig. Im folgenden werden drei Regelmethoden $\mathcal{A}$, $\mathcal{B}$ und $\mathcal{C}$ vorgestellt. Diese werden kombiniert in dem das Minimum der aus diesen Methoden resultierenden γ-Werte gebildet wird:

$$\gamma = \min(\gamma_{\mathcal{A}}, \gamma_{\mathcal{B}}, \gamma_{\mathcal{C}})$$

7.3 Regler $\mathcal{A}$ – eine obere Schranke

Satz 5 *Es sei $\gamma_{\max}$ eine Konstante $\in (0,1)$. Wenn alle $\gamma_i^j \leq \gamma_{\max}$ und alle $|\rho| \leq \frac{1-\gamma_{\max}}{2}$ dann bleibt $LC_i' \leq \max_{k=1..n}(C_k)$.*

Beweis: Ergibt sich direkt aus der Wirkungsweise der geregelten logical Clock, siehe auch Abb. 7.1, links.

□

Durch die Wahl einer oberen Grenze $\gamma_{\mathcal{A}} := \gamma_{\max}$ wird das Problem somit gelöst, wenn die Drifts der Prozessoruhren $\leq \frac{1-\gamma_{\max}}{2}$ sind, wie im folgenden Bild dargestellt:

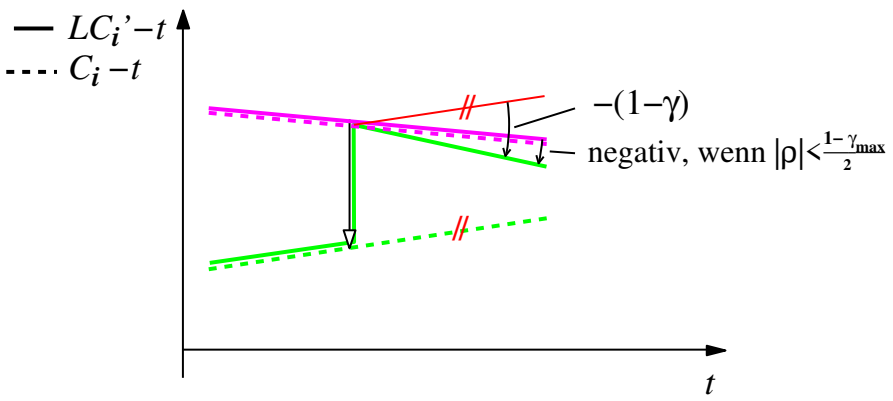

Abb. 7.3: $\gamma_{\max}$, eine Obergrenze für γ

In den meisten Fällen ist das Problem des zu starken Vorgehens von LC' damit gelöst. Für z.B. $\gamma_{\max} = 1-2\cdot10^{-5}$ ist somit das Problem für Drifts $|\rho| \leq 10^{-5}$ gelöst. Damit sind alle Probleme basierend auf typischen Quarzdrifts im Bereich von 10^{-6} und Drifts aufgrund schlechter Uhreneinstellungen im Bereich 10^{-5} gelöst. Der hierdurch induzierte Fehler bei Messungen innerhalb eines Prozesses ist auf $1-\gamma_{\max} = 2\cdot10^{-5}$ beschränkt und damit vernachlässigbar. Falls die Prozessoruhren aber eine größere Drift besitzen, dann kann es sein, daß die geregelte logical Clock LC' mehr als die am stärksten vorgehende Prozessoruhr vorgeht. Dieses Problem wird von den beiden folgenden Regelmechanismen gelöst werden.

7.4 Regler $\mathcal{B}$ – eine notwendige Regelung

Satz 6 *Es gilt* $\min_k(LC'_k - C_k) > 0 \implies \exists_i LC'_i > \max_{k=1..n}(C_k)$

Beweis: Für i = Index mit $C_i = \max_{k=1..n}(C_k)$ ergibt sich die rechte Seite, da aufgrund der Prämisse $LC'_i > C_i$ gilt.

□

Dieser Satz besagt, daß $\min_k(LC'_k - C_k) > 0$ ein hinreichendes Kriterium dafür ist, daß γ reduziert werden muß, bzw. wenn γ durch dieses Kriterium reduziert wird, dann ist diese Reduktion **notwendig**. Abb. 7.4 veranschaulicht dieses Kriterium.

Die Prämisse von Satz 6 wird erfüllt sobald die lila LC'_i größer wird als ihre Prozessoruhr (gestrichelte Linie, siehe). Da dieses Kriterium aber nicht aufzeigt, welche geregelte logical Clocks weiter als erwünscht vorgehen (hier die lila und die türkisfarbene), ist es sinnvoll global alle γ_i zu reduzieren.

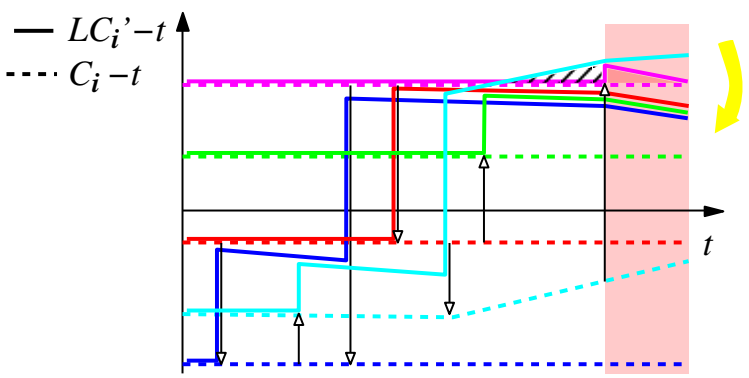

Abb. 7.4: Eine notwendige Regelung

Dies geschieht in dem rot unterlegten Bereich und führt zu einem Absenken aller LC_i-Kurven, die oberhalb ihrer C_i-Kurven liegen, siehe gelber Pfeil. Für die Reduktion, d.h. für $\gamma_{\mathcal{B}}$ wird folgende Regler-Charakteristik angewandt.

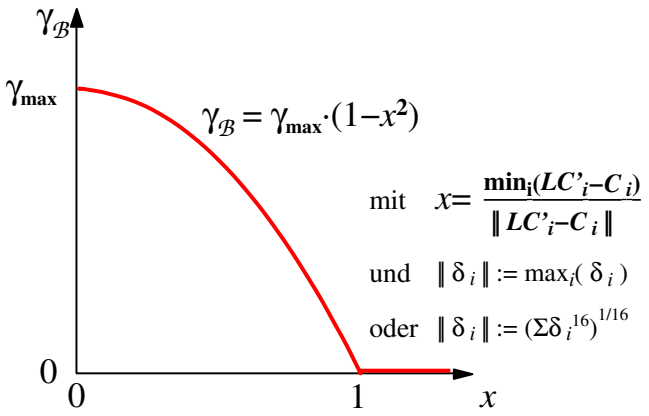

Abb. 7.5: Regler-Charakteristik der notwendigen Regelung

Die hierbei beim Divisor angewandte Normierung ist von minderer Bedeutung. Daher kann die Maximum-Norm, die bei einer großen Anzahl von Prozessen numerisch nicht skaliert, jederzeit auch durch die Norm mittels der 16. Potenz ersetzt werden.

Das in diesem Absatz angegebene hinreichende Kriterium ist aber nicht notwendig, d.h. es gibt Fälle, in denen eine Reduktion von γ nötig wäre (siehe), aber trotzdem $\min_k(LC'_k - C_k) = 0$ ist (siehe). Mögliche Szenarien für diesen Fall sind:

- Kurze Phasen bevor durch eine Message der Regler ausgelöst wird, wie in obigem Beispiel.
- Es gibt Gruppen von Prozesse, die zwar innerhalb der Gruppe kommunizieren, aber über längere Zeit nicht zwischen den Gruppen. Dann kann innerhalb einer Gruppe eine logical Clock übermaßen vorgehen, aber das Kriterium spricht nicht an, da in einer anderen Gruppe mindestens eine logical Clock nicht vorgeht, d.h. $LC'_i = C_i$ ist. Dies kann z.B. in Clustern von Parallelrechnern und darauf abgestimmter Anwendungssoftware passieren, vorausgesetzt die Parallelrechner besitzen intern keine Hardwaresynchronisation.

Nur das letztere Szenarium führt in der Regel zu Abweichungen, die einen weiteren Regelmechanismus erforderlich machen. Dieser wird im folgenden Abschnitt beschrieben.

7.5 Regler $\mathcal{C}$ – eine hinreichende Regelung

Der dritte Regler sei anhand der Abb. 7.6 erklärt.

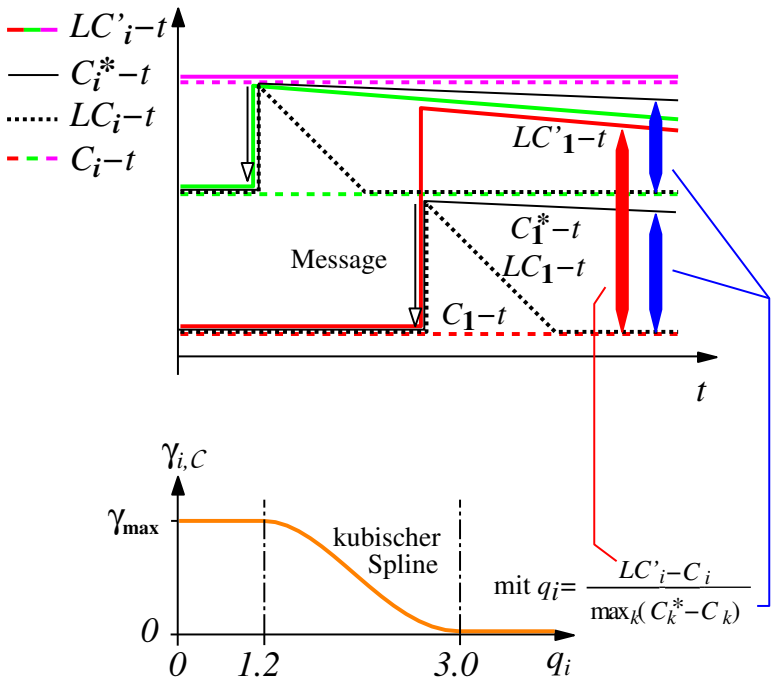

Abb. 7.6: Eine hinreichende Regelung

Zuerst sei die Kurve C_i^* definiert: Sie ist am Anfang identisch mit der einfachen logical Clock LC_i, wird immer soweit vorgestellt, daß sie mindestens denauso weit vorgeht wie LC_i und sie geht mit $[1 - \frac{1}{2}(1 - \gamma)]dC_i/dt$. Sie ist mit den schwarzen Linien dargestellt. Insgesamt ist aber für den dritten Regler nur $\max_i(C_i^* - C_i)$ relevant. Daher wird folgender Algorithmus angewandt, der im Term $-\frac{1}{2}(1-\gamma)d\,C_i/dt$ eine geringfügige und vernachlässigbare Abweichung besitzt.

Algorithmus 4:

Variablen:

MAX für $\max_i(C_i^* - C_i)$ für das letzte berechnete Ereignis e_i^j

LC_{MAX} für den den Wert von LC_i bei der letzten Erhöhung von MAX

Initialisierung:

$MAX = 0$ und $LC_{MAX} = LC_i(e_i^0)$ beim ersten Event

Vorschrift bei jedem weiteren Event e_i^j:

$MAX := MAX - \frac{1}{2}(1 - \gamma_{\max})max(0, LC_i(e_i^j) - LC_{MAX})$

$MAX := \max(0, MAX)$

if $(LC_i(e_i^j) > C_i(e_i^j))$ then

$MAX = LC_i(e_i^j) - C_i(e_i^j)$

$LC_{MAX} = LC_i(e_i^j)$

endif

Für die Kurven C_i^* gilt, daß ihre Abstände zu C_i genauso beschränkt sind wie bei den einfachen logical Clocks LC_i. Aber C_i^* erfüllt nicht die Clock Condition. Im Prinzip handelt es sich bei C_i^* um Parallelverschiebungen wie bei der geregelten logical Clock, nur daß sie durch einzelne Messages nur auf die Prozessorzeit des jeweiligen Senders vorgestellt werden kann, während die geregelten logical Clock über Messageketten auf die Prozessorzeit der Initiatoren der Ketten vorgestellt werden kann. Für die Regelung wird nun das Verhältnis

$$\frac{LC_i' - C_i}{\max_k(C_k^* - C_k)}$$

berechnet. Wenn es von dem Prozeß mit der am weitesten vorgehenden Prozessoruhr Messages mit minimaler Laufzeit zu dem Prozeß mit der am weitesten nachgehenden Uhr gibt, dann gibt der Divisor $\max_k(C_k^* - C_k)$ den maximalen Abstand zweier Uhren an. Dieser Regler regelt spezifisch das $\gamma_{i,\mathcal{C}}$ jedes einzelnen Prozesses i auf der Basis der in Abb. 7.6 angegebenen Reglercharakteristik. Diese bewirkt, daß im ungünstigsten Fall der Regler erst das γ_i zu reduzieren beginnt, wenn die geregelte logical Clock LC_1' um das 1.2-fache der maximalen Uhrendifferenz gegenüber der am weitesten vorgehenden Prozessoruhr vorgeht, d.h. spätestens wenn ein

$$LC_i' > \max_k(C_k) + 1.2 \cdot \max_{k,m}(C_k - C_m)$$

setzt der Regler $\mathcal{C}$ ein. Dieser Regler garantiert auf der Basis obiger Überlegung des ungünstigsten Falls und aufgrund der auf Null absinkenden Reglercharakteristik aber auch, daß die Differenz $LC' - \max_k(C_k)$ nach oben beschränkt bleibt.

Andererseits hat der Regler absichtlich eine Regelkurve, die vom 1.2-fachen bis zum 3-fachen geht, da das Kriterium dieses Reglers kein hinreichendes ist, d.h. daß es Fälle gibt in denen der Regler γ herunterregelt, obwohl es dafür keine Notwendigkeit gibt, d.h. alle $LC_i' \leq \max_k(C_k)$ sind. Der Regler kann zu restriktiv sein, wenn

$$\max_{\substack{i,k \text{ mit Messages} \\ \text{von } i \text{ nach } k}}(C_i - C_k) < 1.2 \cdot \max_{i,k}(C_i - C_k)$$

wobei der Faktor 1.2 in der Praxis aufgrund des Unterschieds zwischen den real auftretenden Messagedelays und dem minimalen Delay meist größer ist. Der Fall eines zu restriktiven Reglers kann z.B. auftreten, wenn bei einer array-förmigen Anordnung der Prozessoren die Uhren von einer Seite zur anderen immer mehr nachgehen und die Anwendung nur Messages zwischen den direkten Nachbarn austauscht.

7.6 Zusammenfassung der Regler

Die Regler werden mittels $\gamma_i^j = \min(\gamma_{\mathcal{A}}, \gamma_{\mathcal{B}}, \gamma_{i,\mathcal{C}})$ kombiniert. Der Regler $\mathcal{A}$ ist ausreichend, wenn die Drifts der Prozessoruhren hinreichend klein sind, und somit in der Praxis der am meisten relevante. Es wird $\gamma_{\max} = 1 - 2 \cdot 10^{-5}$ empfohlen. Der Regler $\mathcal{B}$ ist in den meisten anderen Fällen ausreichend, aber nicht in allen, hat ein minimales Überschwingen und kontrolliert das Ziel ($LC' \cong \max_k(C_k)$) sehr scharf. Der Regler $\mathcal{C}$ ist ausreichend in allen Fällen, aber in manchen zu restriktiv und wurde daher nur mit einer schwach regelnden Charakteristik versehen. Er ist der einzige der γ prozeßspezifisch regelt, d.h. $gamma_{i,\mathcal{C}}$ kann für unterschiedliche i verschieden sein. Da die Charakteristiken in den Reglern $\mathcal{B}$ und $\mathcal{C}$ ebenfalls γ auf maximal $\gamma_{\max}$ beschränken, bedarf der erste Algorithmus keines gesonderten Implementierungsaufwands.

Kapitel 8

Fehleranalyse

Man hat drei Grundtypen von Meßintervallen zu unterscheiden:

d_M Messagedelay,

d_R Delay zwischen zwei aufeinanderfolgenden Ereignissen im gleichen Prozeß und das letztere ist ein Receive-Event, und

$d_{S/I}$ Delay zwischen zwei aufeinanderfolgenden Ereignissen im gleichen Prozeß und das letztere ist ein Internal- oder Send-Event.

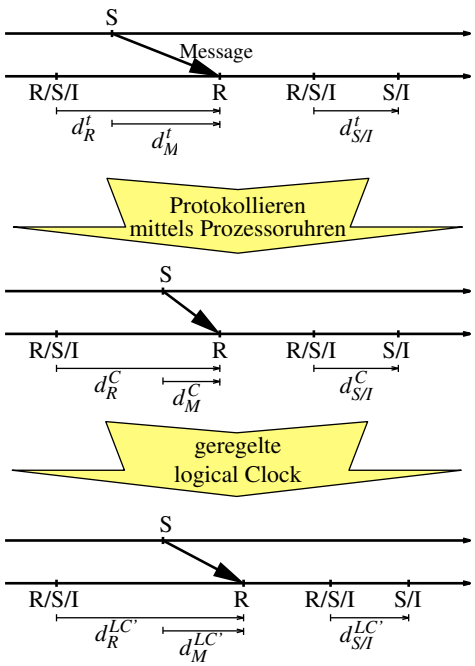

Abb. 8.1: Delays (R=Receive, S=Send, I=Internal Event)

Bei dieser Unterscheidung wurde der Funktionsweise der geregelten logical Clock Rechnung getragen. Jedes Delay zwischen zwei beliebigen Events läßt sich als Summe und Differenz derartiger Intervalle berechnen (vorausgesetzt der Graph aller Messages (Kanten) zwischen den Prozessen (Knoten) ist zusammenhängend).

Außerdem ist zu unterscheiden zwischen

d^t reales Delay,

d^C Delay gemessen auf der Basis der Prozessoruhren, und

$d^{LC'}$ Delay bestimmt nach der Korrektur von d^C durch die geregelte logical Clock.

Folgende Abschätzungen sind möglich:

Zu d_M:

Unter der Voraussetzung, daß μ das minimale Messagedelay ist (d.h. $\forall d_M^t : d_M^t \geq \mu$), gilt $d_M^{LC'} \geq \mu$, d.h. fehlerhafte Messwerte d_M^C mit $d_M^C < \mu$ werden garantiert auf einen Wert

$d_M^{LC'} \geq \mu$ korrigiert. Wenn außerdem zwischen allen Prozessen hinreichend oft Messageketten existieren, bei denen die einzelnen Messagedelays nahe μ sind, dann kann die geregelte logical Clock in allen Prozessen das Maximum aller Prozessoruhren annähern und damit eine einheitliche globale Zeit bilden und die Fehler $|d_M^{LC'} - d_M^t|$ minimieren. Die durch diese Abschätzungen charakterisierte Korrektur der Messagedelays ist ein wesentlicher **positiver Effekt** der geregelten logical Clock. Die Beispiele in Kapitel 10.1 und 10.2 zeigen diese Wirkung insbesondere in der Praxis in den Abb. 10.1 (S. 24) und 10.2 (S. 26).

Die beiden folgenden Abschnitte befassen sich dagegen mit **zusätzlichen Fehlern**, die durch die geregelte logical Clock entstehen und damit mit Nachteilen.

Zu $\mathbf{d_{S/I}}$:

Es sei $\hat{\rho}$ die maximale absolute Drift der Prozessoruhren. Dann gilt $d_{S/I}^t(1-\hat{\rho}) \leq d_{S/I}^C \leq d_{S/I}^t(1+\hat{\rho})$. Aufgrund der Terme mit γ in Alg. 3 gilt für die geregelte logical Clock $d_{S/I}^C(1-(1-\gamma)) \leq d_{S/I}^{LC'} \leq d_{S/I}^C$. Zusammenfassend ergibt sich unter Vernachlässigung höherer Terme

$$d_{S/I}^t(1-\hat{\rho}-(1-\gamma)) \leq d_{S/I}^{LC'} \leq d_{S/I}^t(1+\hat{\rho})$$

und für den relativen Fehler

$$\epsilon_{S/I} := \left| \frac{d_{S/I}^{LC'} - d_{S/I}^t}{d_{S/I}^t} \right| \leq \hat{\rho} + (1-\gamma)$$

Typischerweise liegt $(1-\gamma)$ bei $1-\gamma_{\max} = 2 \cdot 10^{-5}$ und $\hat{\rho}$ bei 10^{-6} und für $\hat{\rho} > 10^{-5}$ sollte der Regler im Normalfall γ nur soweit herunterregeln, daß $1-\gamma \leq 4\hat{\rho}$ erhalten bleibt (dies berücksichtigt 2-faches Überschwingen und gegenläufige maximale Drift wie ca. im linken Diagramm in Abb. 7.1). Damit gilt in der Regel

$$\epsilon_{S/I} \leq \max(2 \cdot 10^{-5} + \hat{\rho}, 4\hat{\rho})$$

d.h. **diese zusätzlichen Fehler** durch die geregelte logical Clock sind **vernachlässigbar** oder zumindest im Bereich der Drift-Fehler der Prozessorclocks.

Zu $\mathbf{d_R}$:

Aufgrund des möglichen Vorstellens der geregelten logical Clock bei Receive-Events (siehe Term $LC'_{k=\text{sender}} + \mu$ in (6.1)) um die Clock Condition (3.4) zu gewährleisten, entstehen bei $d_R^{LC'}$ ggf. größere Fehler. Hierbei sind zwei Phasen zu unterscheiden:

1. Phase: Aufgrund anfänglicher Messages werden die LC_i dem $\max_k(C_k)$ angenähert;

2. Phase: Nötige Korrekturen aufgrund der Uhrendrifts oder späterer Messages, deren Delay näher an μ liegt.

In der 1. Phase sind die absoluten Fehler $e_R := \left| d_R^{LC'} - d_R^t \right|$ im Bereich der Uhrenfehler $|C_i - C_k|$. Dies kann aber durch eine Synchronisation der Uhren vor Beginn der zu beobachtenden Anwendung vermieden werden, da dann zu Beginn $\left| d_R^{LC'} - d_R^t \right| \cong 0$. In der 2. Phase hängen die Fehler davon ab, ob die Uhren kontinuierlich arbeiten oder ob sie Sprünge machen.

Außerdem ist zu berücksichtigen, wie viel Zeit verstreicht, bis durch eine geeignete Message die geregelte logical Clock korrigiert wird, denn dies kann zu einem größeren Korrekturbedarf und damit zu einem größeren Fehler e_R führen. Die geregelte logical Clock kann den Fehler e_R für jeden Tracefile und jedes Receive-Event berechnen und z.B. Maximum und Durchschnitt ausgeben. Bezüglich des relativen Fehlers $\epsilon_R := e_R/d_R^t$ kann man keine sinnvolle Abschätzung garantieren, da er bei sehr kurzen Intervallen d_R^t beliebig groß werden kann. Doch in der Praxis sollte ϵ_R **im Durchschnitt über alle Receive-Events höchstens einige Prozent** betragen. Kap. 10.1 enthält hierzu ein Beispiel. Mit der in Kap. 13 vorgestellten nachträglichen stückweise linearen Amortisation werden diese Fehler in der Regel nochmals stark reduziert.

Bei Tracefiles bei denen alle Messagedelays größer oder gleich μ sind, ist zur Erreichung der Clock Condition keine Korrektur nötig und es wird auch keine durchgeführt. In diesem Fall entstehen durch die geregelte logical Clock keine zusätzlichen Fehler.

Zusammenfassend läßt sich sagen, daß in der Regel der Gewinn durch die sinnvolle Korrektur der Messagedelays (d_M) die zusätzlichen Fehler bei Delays innerhalb eines Prozesses ($d_{S/I}$ und d_R) bei weitem übersteigt. Durch eine präzisere Bestimmung von μ für jede einzelne Message in Abhängigkeit von deren Länge ließe sich das Verfahren bei Bedarf noch deutlich verbessern.

Kapitel 9

Implementierung

Abb. 9.1 zeigt das Schema der Implementierung der geregelten logical Clock als Filter. In einem **single-pass** wird ein Tracefile mit möglicherweise fehlerhaften Zeitstempeln gelesen, die geregelte logical Clock berechnet und die Events mit den damit gebildeten neuen Zeitstempel wieder ausgegeben. Es wird vorausgesetzt, daß die Events des Eingabefiles gemäß ihren Zeitstempeln sortiert sind. Der *Input-Puffer* dient zur Speicherung von Receive-Events und weiteren ggf. nachfolgenden Events solange die geregelte logical Clock des zugehörigen Send-Events noch nicht berechnet ist und diese Events folglich noch nicht bearbeitet sind. Der *Puffer für berechnete Send-Events* speichert die neuen Zeitstempel von Send-Events bis das zugehörige Receive-Event bearbeitet wird.

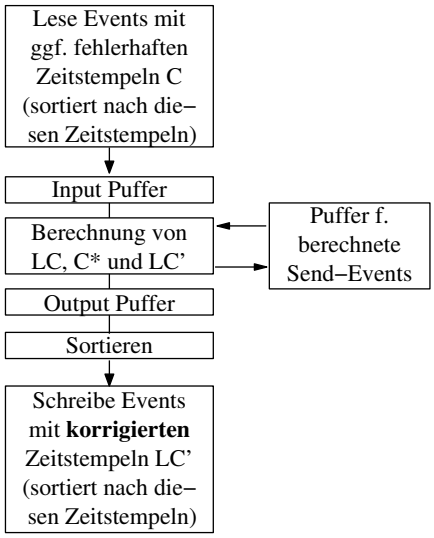

Abb. 9.1: Implementierung als Filter

Der *Output-Puffer* speichert die Events mit den neu berechneten Zeitstempeln solange es möglich ist, daß noch weitere Events mit geringeren Zeitstempeln kommen können. Der Output-Puffer ist zweigeteilt. In die erste Gruppe werden alle Events eingespeichert deren neuer Zeitstempel kleinergleich als eine Schranke ist, in die zweite alle restlichen. Sobald ein Event eingelesen und entweder bearbeitet oder im Input-Puffer zwischengespeichert wurde und dessen alter Zeitstempel größergleich der Schranke zwischen den beiden Ausgabegruppen ist, dann wird die erste Gruppe sortiert und ausgegeben und anschließend die zweite Gruppe zur ersten gemacht und die Schranke neu auf das Maximum aller neuen Zeitstempel in dieser Gruppe festgelegt.

Die benötigte Rechenzeit ist linear in der Anzahl der Ereignisse, da die Anzahl der bei der Ausgabe zu sortierenden Ereignisse durch $(3+1) \cdot \max_{i,k}(|C_i - C_k|) \cdot \Delta E/\Delta t$ beschränkt ist. $\Delta E/\Delta t$ gibt hierbei die Anzahl der Ereignisse pro Zeitintervall an. Der Faktor 3 ergibt sich aus der oberen Grenze in der Reglercharakteristik des hinreichenden Reglers ($\mathcal{C}$).

Alternativ können die Ereignisse in dem Tracefile oder in den pro Prozeß erstellten Tracefiles auch nur bezüglich des jeweiligen Prozesses sortiert sein. Dann kann der Sortiervorgang vor der Ausgabe entfallen, da der Algorithmus 3 die Reihenfolge der zu einem Prozeß gehörenden Ereignisse nicht ändert.

Kapitel 10

Test des Algorithmus

Der Algorithmus wurde mit drei Beispielen getestet:

1. Mittels eines Tracefiles einer realen Anwendung mit 16 Prozessen auf einem Parallelrechner mit ungenügender Uhrensynchronisation.

2. Ein Beispiel mit großem Clocktick. Es handelt sich hierbei um das Beispiel 1 in [Rab96]. Ohne die Regelung, d.h. bei $\gamma \equiv 1$, würde bei diesem Beispiel LC' unbegrenzt wachsen.

3. Mittels einer simulierten FE-Rechnung.

Diese sind in den folgenden Kapiteln beschrieben.

10.1 Tracefile einer realen Anwendung

Abb. 10.1 auf S. 24 zeigt links oben einen Ausschnitt des Timeline-Diagramms einer realen Anwendung und rechts oben das zugehörige Diagramm der durchschnittlichen Transferraten, beide Diagramme auf der Basis der nicht korrigierten Zeitstempel der Prozessoruhren. Man sieht bei von Knoten 2 ausgehenden Messages extrem unrealistische Transferraten im Bereich 120 (nach Knoten 1), 270 (nach Knoten 5) und 240 (nach Knoten 9) kB/sec. Nach einer Korrektur mittels der geregelten logical Clock mit $\mu = 500\mu$s ergibt sich die wesentlich realistischere Darstellung links unten. Hierbei war nur der Regler $\mathcal{A}$ mit $1 - \gamma = 2 \cdot 10^{-5}$ aktiv. Die Regler $\mathcal{B}$ und $\mathcal{C}$ stellten keinen Regelbedarf fest. Damit ergibt sich für Zeitmessungen innerhalb eines Prozesses für Intervalle zwischen einem beliebigen Ereignis und einem darauffolgenden Send- oder Internal-Event ein durch die geregelte logical Clock verursachter maximaler relativer zusätzlicher Fehler von **0.002 %**. Für derartige lokale Zeitintervalle, die aber mit einem Receive-Event enden, ergaben sich in der Anfangsphase (erste 0.2 sec, aufgrund einer fehlenden ausreichenden Synchronisation zu Beginn) zusätzliche absolute Fehler von maximal 0.8 ms und ein durchschnittlicher relativer Fehler von 55 % . Während der restlichen 3.15 sec ist der durchschnittliche Fehler nur noch 0.13 %. Hierbei kommt es trotzdem noch vor, daß vereinzelt Intervalle bis zu 180 % verlängert werden. Beide Phasen zusammen ergeben im Durchschnitt einen relativen Fehler von **4.6 %**.

Die Qualität der **Verbesserung** von Zeitmessungen der Messagedelays ergibt sich aus dem Vergleich der beiden Diagramme rechts oben und links unten. Die wesentlichen Fehler in der oberen Darstellung sind eliminiert. Die verbleibenden Abweichungen in den Zeilen 2,

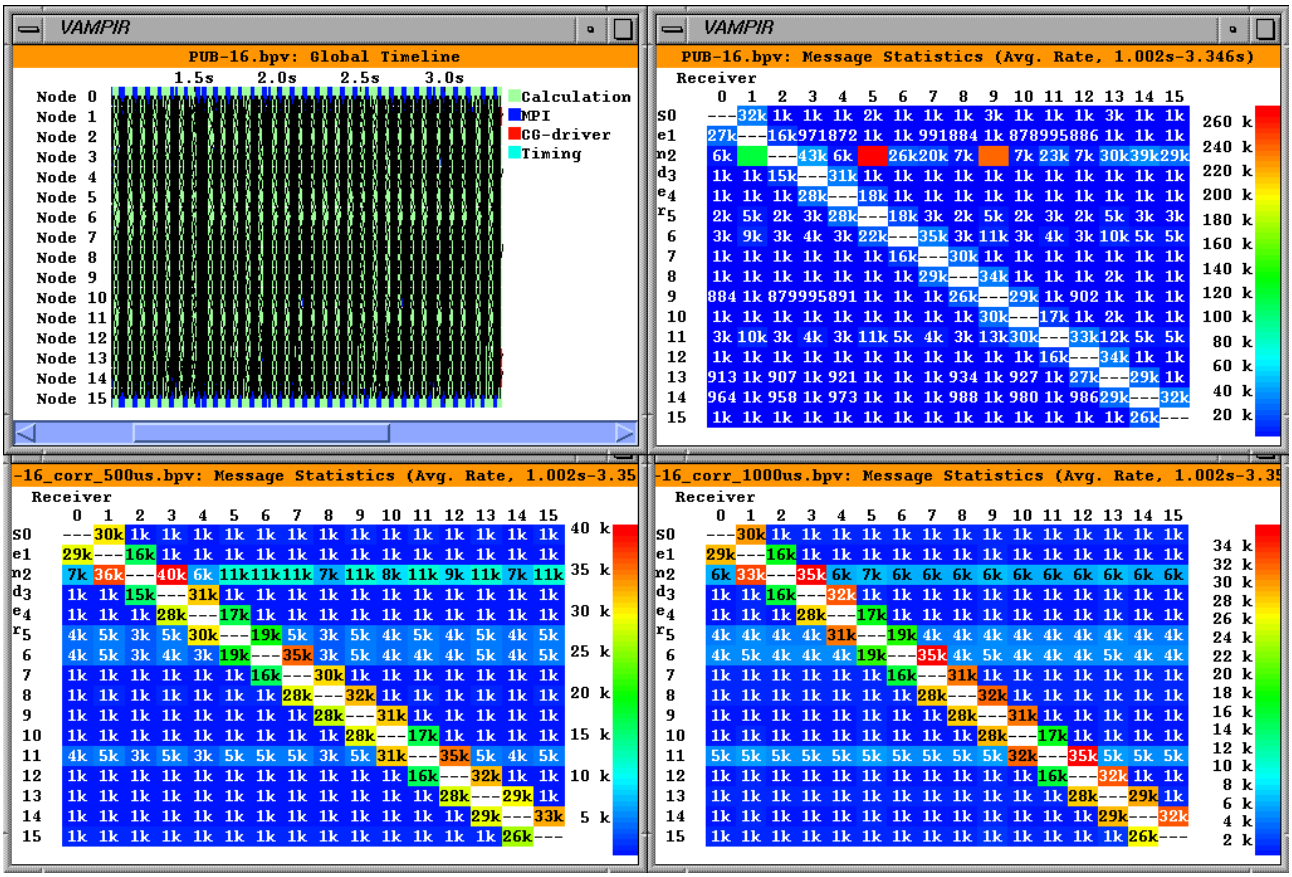

Abb. 10.1: Timeline-Diagramm und durchschnittliche Transferraten bei einer realen Anwendung,
Transferraten rechts oben basierend auf C,
links unten basierend auf LC', berechnet mit $\mu = 500\mu$s,
rechts unten basierend auf LC', berechnet mit $\mu = 1000\mu$s

5, 6 und 11 können real sein, aber auch auf mangelnder Synchronisation aufgrund fehlender Minimal-Delay-Messages beruhen.

In dem Diagramm rechts unten wurde bei der geregelten logical Clock ein $\mu = 1000\mu$s zugrunde gelegt. Dieses μ liegt deutlich über dem mit horizontalen Parallelstreifen in Dudas Algorithmus (s. Abb. 2.2 auf S. 5) ermittelten minimalen Messagedelay von 620μs, aber noch knapp unter dem Wert, ab welchem die logical Clock mit $\gamma \equiv 1$ immer stärker vorgeht. Der Algorithmus arbeitet hierdurch sehr scharf und erzielt bzgl. Zeitmessung von Messagedelays $d_M^{LC'}$ ein noch besseres Ergebnis, wie man dem Diagramm entnehmen kann, doch Zeitmessungen innerhalb eines Prozesses werden deutlich schlechter, bei $d_{S/I}^{LC'}$ von 0.002 auf 0.023 % und bei $d_R^{LC'}$ von 4.6 auf 10.5 %. Mit der in Kap. 13 skizzierten Amortisation läßt sich der durchschnittliche Fehler von $d_R^{LC'}$ von 4.6 % auf einen Wert unter 0.21 % und der maximale von 180 % auf einen Wert unter 1.4 % reduzieren.

10.2 Ein Beispiel mit großem Clocktick

Dieses Beispiel testet die Regler und die Qualität der neuen Zeitstempel in einer extremen Umgebung: Einerseits besitzen die Prozessoruhren ein Clocktick das 40 mal größer als das minimale Messagedelay ist und andererseits sind die Uhren so justiert, daß ohne die Regelug das LC' gegenüber C unbeschränkt wachsen würde. 20 Knoten (0...19) senden eine Messagekette von Knoten 0, über Knoten 1, 2, usw. nach Knoten 19 und anschließend wieder zurück nach Knoten 0. Die Uhren haben ein Clocktick von 10 ms, das Messagedelay sei 250 μs, und das Weiterschalten der Uhren sei so, daß in der Messagekette von Knoten 0 nach 19 die Messages jeweils ein scheinbares Messagedelay von -9.750 ms besitzen, und in der Messagekette von 19 nach 0 ein scheinbares Delay von -250 ms. Abb. 10.2 auf S. 26 zeigt oben den Anfang des zugehörigen Timeline-Diagramms auf der Basis der Zeitstempel der Prozessoruhren. Es stellt den Sachverhalt extrem verzerrt dar. Der wirkliche Verlauf ist nicht erkennbar. Z.B. bei Knoten 0 wird die zweite Message nach Knoten 1 bei 20.25 ms abgeschickt (s. vertikaler Pfeil). Die Darstellung dieser Message ist revers, d.h. es hat den Anschein, daß sie von Knoten 1 nach Knoten 0 geht, bzw. das dargestellte Delay ist negativ (-9.7 ms, s. innere Box). Um ca. 9.9 ms später wird dann von Knoten 0 als nächstes eine Message von Knoten 1 empfangen. Aufgrund des bis dahin noch nicht erfolgten Clockticks schließt sich diese in der Darstellung – fehlerhafterweise – sofort an, s. horizontaler Pfeil. Sie ist ebenfalls revers dargestellt.

Die Korrektur der Zeitstempel mittels der geregelten logical Clock ermöglicht die sehr realistische Darstellung des Programmverlaufs in dem mittleren Diagramm. Hierbei wurde bei der Korrektur $\mu = 250\,\mu$s angenommen. Links sieht man den Einschwingvorgang während der ersten 7 Messages. Die beiden oben beschriebenen Messages findet man in dieser Darstellung nun richtig von Knoten 0 nach Knoten 1 (s. vertikaler Pfeil, 39.368-39.618 ms) und von Knoten 1 nach Knoten 0 richtigerweise ca. 9.5 ms später (s. horizontaler Pfeil, 49.036-49.286 ms). Eine Analyse während der Berechnung der neuen Zeitstempel zeigt, daß die Regler γ auf ca. 0.17 herabregeln.

Die untere Abbildung zeigt eine weniger korrekte Darstellung. Hierbei wurde für die Berechnung der geregelten logical Clock ein $\mu = 150\,\mu$s angenommen, d.h. das reale Messagedelay lag $100\,\mu$s über dem zugrunde gelegten minimalen Messagedelay. Dieses Beispiel zeigt, daß die geregelte logical Clock auch bei extrem nicht-kontinuierlichen Prozessoruhren und auch bei extremem Regelbedarf eine sinnvolle Darstellung des Programmflusses ermöglichen kann. Dies war durch die in Kap. 2 beschriebenen Methoden seither nicht möglich.

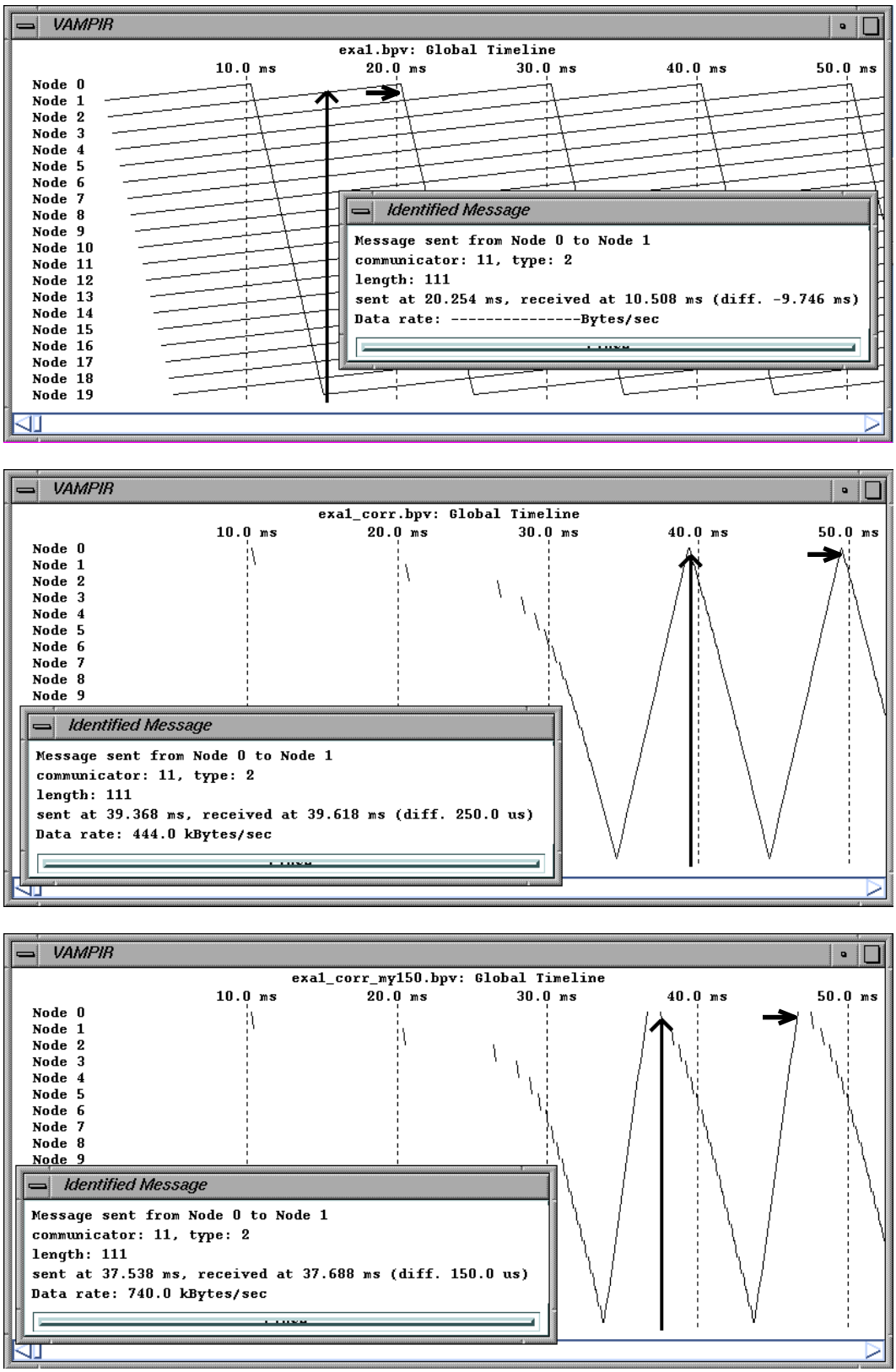

Abb. 10.2: Beispiel mit großem Clocktick

10.3 Simulation einer FE-Rechnung

Die Komponenten dieser Computersimulationen sind

- der simulierte Ablauf einer parallelisierten FE-Rechnung auf einem regulären Gitter erzeugt eine Ereignismenge $E = \{e_i^j\}$ an den Zeitpunkten $t(e_i^j)$,
- simulierte Betriebssystem-Uhren mit vorgegebenem Fehlverhalten erzeugen Abweichungen zwischen $C(t(e_i^j))$ und $t(e_i^j)$,
- die geregelte logical Clock errechnet zu $C(t(e_i^j))$ die Werte $LC'(e_i^j)$ und $LC(e_i^j)$,
- und ein Analysemodul bewertet die geregelte logical Clock.

Die FE-Rechnung wird auf einem rechteckigen Gitter mit $n_1 \times n_2$ Prozessen ausgeführt. Jeder Prozeß hat die Aufgabe in jedem Simulationsschritt eine größere Anzahl finiter Elemente zu berechnen.
Der Ablauf in jedem Prozeß ist folgender

1. Berechnung der finiten Elemente an den Rändern zu den Nachbarn.
2. Versenden dieser neuen Ergebnisse zu den Nachbarn.
3. Berechnen der restlichen finiten Elemente.
4. Empfangen der neuen Nachbarwerte.

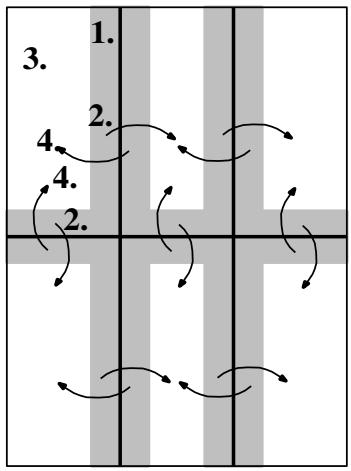

Abb. 10.3: Eine fiktive parallele FE-Berechnung

Diese Simulationsumgebung ermöglichte das Testen und verifizieren der theoretischen Grundlagen der geregelten logical Clock.

Kapitel 11

Anwendung der geregelten logical Clock

Die geregelte logical Clock ist für Monitoring Tools konzipiert. Beim *Monitoring* verteilter und paralleler Anwendungen kann sie als Filter eingesetzt werden. Ein Tracefile mit Rückwärtsbezügen, d.h. dessen Zeitstempel die Clock Condition (3.4) nicht erfüllen, wird nachträglich so modifiziert, daß er einerseits die Clock Condition erfüllt, und daß andererseits die Zeitstempel für Zeitmessungen möglichst nur geringfügig verändert werden.

Da die geregelte logical Clock zuerst nur für eine Bearbeitung der Events vom Beginn an entworfen ist, gibt es beim Einsatz innerhalb eines Monitortools Probleme, wenn eine beliebige Positionierung innerhalb des Tracefiles mit anschließendem Vorwärts- oder Rückwärtsscrollen möglich sein soll. Einerseits kann man die Korrekturen von Anfang des Tracefiles an berechnen und für das Scrollen innerhalb des Files zwischenspeichern, andererseits kann man auch die geregelte logical Clock an einer beliebigen Stelle innerhalb des Traces beginnen lassen. Für das Rückwärtsscrollen kann man den Algorithmus ebenfalls einsetzen, indem man das Vorzeichen der Zeitstempel wechselt und die Rolle von Send- und Receive-Events vertauscht. Letzteres Verfahren hat den Nachteil, daß die durch die geregelte logical Clock modifizierten Zeitstempel von der Wahl des Anfangsevents abhängen, d. h. der Anwender sieht das gleiche Event u. U. mit unterschiedlichen Zeitstempeln dargestellt.

Dieser Nachteil kann ausgeglichen werden, indem das Monitortool jedem Ereignis zwei verschiedene Zeitstempel zuordnet,

- die Zeit der lokalen Prozessoruhr, ggf. linear korrigiert mittels der in [MT95] beschriebenen SBA-Methode (**s**ampling **b**efore and **a**fter); sie kann für Zeitdifferenzmessungen innerhalb eines Prozesses und zum Referenzieren eingesetzt werden;

- die Zeit der geregelten logical Clock; sie dient zur Darstellung der Ereignisse in Zeitdiagrammen und für Zeitdifferenzmessungen zwischen Ereignissen in verschiedenen Prozessen; ihr Wert ist jeweils abhängig vom Anfangsevent nach der letzten Repositionierung.

Die geregelte logical Clock kann auch für Monitoring-Komponenten innerhalb von zentralen *Debug-Tools für parallele Programme* eingesetzt werden. Die fehlende Synchronisation am Ende sollte hierbei durch eine resourcenarme fortlaufende Synchronisation parallel zur Anwendung ersetzt werden. Die Synchronisation am Anfang kann dann in der Regel ebenfalls entfallen. Bei derartigem on-line Einsatz ist zu beachten, daß das LC' eines Ereignisses erst berechnet werden kann, wenn von allen vorhergehenden Receive-Events (ggf. einschließlich

des aktuellen Events) das LC' der zugehörigen Send-Events berechnet sind. Dies ist z.B. gegeben, wenn die Prozessorzeit C aller bzgl. der physikalischen Zeit vorhergehenden Ereignisse für die Berechnung von LC' zur Verfügung stehen.

Es wird empfohlen, eine Abschätzung für das minimale Messagedelay μ als Nebenprodukt des eingesetzten Synchronisationsalgorithmus zu erlangen oder eine Hardware bezogene Abschätzung durchzuführen. Diese Abschätzung sollte vor dem ersten Event in den Tracefile eingetragen werden. Die Streuung der Messagedelays kann noch einen signifikanten Bereich unterhalb der Delays mit der größten Häufigkeit besitzen. Unsere Messungen haben in einem Ethernet und in einem FDDI-Ring ergeben, daß das minimale Delay im allgemeinen größer als ein Viertel der häufigsten Roundtriptime eines leeren RPCs ist. Da aber Messages mit minimalen Messagedelays bei einer derartigen Wahrscheinlichkeitsverteilung äußerst selten sind, kann man in der Regel $\mu = 90\% \cdot \frac{1}{2} häufigste\ Roundtriptime$ wählen.

Im Gegensatz zu dem in [Rab96, Rab97] beschriebenen Regler handelt es sich hier bei den Reglern $\mathcal{B}$ und $\mathcal{C}$ um Algorithmen, die eine zentrale Verfügbarkeit aller Zeitstempel erfordern und die daher nicht für eine auf die einzelnen Prozesse verteilte Implementierung geeignet sind. Daher ist das Einsatzgebiet der hier beschriebenen Regler auf Monitortools und vergleichbare Anwendungen beschränkt.

Kapitel 12

Zusammenfassung

Die geregelte logical Clock ist eine Methode zum Korrigieren der Zeitstempel mit folgenden Eigenschaften

- Die korrigierten Zeitstempel erfüllen die Clock Condition.

  D.h. ein causal nachfolgendes Ereignis hat auch einen späteren Zeitstempel. Die neuen Zeitstempel sind damit zur Darstellung der Ereignisse in einem Monitortool geeignet.

- Etablieren einer globalen Zeit.

  Damit kann sie problemlos für durchschnittliche Zeitmessungen zwischen Ereignissen zweier verschiedener Prozesse eingesetzt werden, z.B. zur Messung der Dauer eines Nachrichtentransfers.

- Geringe Meßfehler für Zeitspannen zwischen Ereignissen im selben Prozeß.

  Damit entfällt die Notwendigkeit, für die Darstellung der Ereignisse, für die Zeitmessungen zwischen verschiedenen Prozessen und für die Zeitmessungen innerhalb der Prozesse unterschiedliche Zeitstempel zu benutzen.

Um Fehler beim Start der geregelten logical Clock zu minimieren, ist die Synchronisation vor Beginn der zu monitorenden Anwendung empfohlen. Hierbei können auch resourcenarme Synchronisationsmethoden, wie z.B. XNTP, eingesetzt werden. Sie sind meistens zu anderen Zwecken in Workstationclustern schon installiert. Die geregelte logical Clock ist hierbei unempfindlich, wenn zum Zwecke der Synchronisation die Uhrengeschwindigkeit um einige Prozent geändert wird. In Kombination mit solchen Uhrensynchronisationsverfahren kann eine zusätzliche Synchronisation am Anfang und Ende des Programmlaufs entfallen. Vor allem bei längeren Programmlaufzeiten ist die geregelte logical Clock damit anderen Methoden überlegen, da diese meist eine geringe Varianz der Uhrengeschwindigkeiten voraussetzen.

Ein typische Einsatzgebiet ist das Monitoring paralleler Anwendungen in Workstation-Clustern.

Kapitel 13

Zukünftige Arbeiten

Es ist geplant die Fehlerabschätzungen zu präzisieren, unnötige γ-Reduktionen des Reglers $\mathcal{C}$ zu analysieren, ggf. die Reglercharakteristiken auf der Basis der Theorie der Regelungstechnik zu verbessern, die Implementierung mit einer für den Anwender relevanten Fehleranalyse zu versehen, und die Implementierung dann öffentlich verfügbar zu machen.

Außerdem wird die in Abb. 13.1 skizzierte nachträgliche stückweise lineare Amortisation der durch das Vorstellen auf $LC'(zugehöriges\ Send-Event)+\mu$ verursachten Unstetigkeiten realisiert werden.

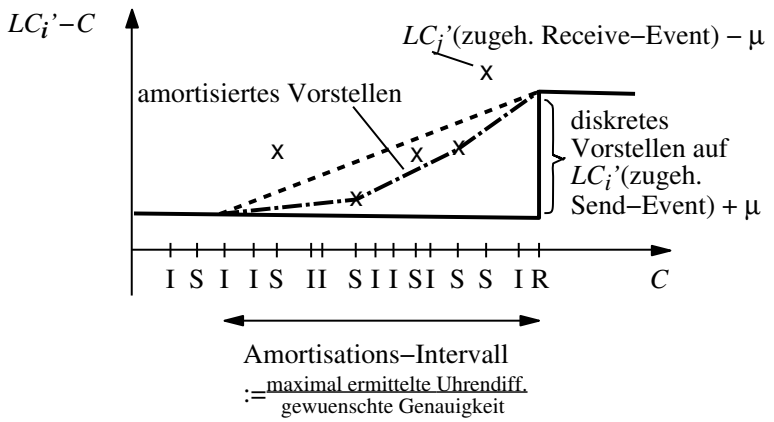

Abb. 13.1: Nachträgliche stückweise lineare Amortisation

Falls in dem in der Abbildung angegebenen Amortisationsintervall sich keine Send-Events mit $LC'_j(zugehöriges\ Receive-Event)-\mu$ unterhalb der gestrichelten Kurve befinden, dann wird LC'_i durch dei gestrichelte Kurve korrigiert, andernfalls wird die strichpunktierte stückweise lineare konvexe untere Hüllkurve als Korrektur angewandt.

Im Gegensatz zu der von F. Schmuck und F. Cristian in [SC90] untersuchten Amortisation wird hier die diskrete Änderung nicht in einem an dem Synchronisationszeitpunkt beginnenden Zeitintervall ausgeglichen, sondern der Ausgleich wird rückwirkend in einem Intervall vor dem Synchronisationszeitpunkt durchgeführt. Die zeitliche Länge dieses Amortisationsintervalls ist beschränkt durch die maximale Uhrendifferenz und die gewünschte Mindestgenauigkeit, typischerweise in einem Workstation-Cluster auf z.B. $5\,\mathrm{ms}\,/1\,\% = 0.5\,\mathrm{sec}$. Eine rückwirkende Amortisation ist nur möglich, wenn der Einsatz der geregelten logical Clock eine derartige nachträgliche Änderung der Zeitstempel zuläßt. Bei der in Kap. 9 vorgestellten

Filterimplementierung ist das möglich, bewirkt aber eine Vergößerung der zu sortierenden Ereignismenge auf (*die Länge des Amortisationsintervalls*) $\cdot$ $(\Delta E/\Delta t)$. Bei der in Kap. 11 skizzierten on-line Implementierung im Rahmen eines Debug-Tools muß berücksichtigt werden, daß sich die Darstellung durch die Amortisation nachträglich ändert. Dies kann z.B. bei einem Timeline-Diagramm sukzessive im Rahmen des beim zeitlichen Fortschreiten notwendigen Scrollens realisiert werden, wodurch dies in den meisten Fällen für den Anwender verborgen bleibt.

Literaturverzeichnis

[BD87] Özalp Babaoğlu and Rogério 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.

[BDM+94] P. Baldy, H. Dicky, R. Medina, M. Morvan, and J.F. Vilarem. Efficient reconstruction of the causal relationship in distributed systems. In Michel Cosnard et al., editor, *Parallel and Distributed Computing, Theory and Practice, First Canada-France Conference, Montréal, May 19-21, 1994*, pages 101–113. Springer-Verlag, 1994.

[CB91] Bernardette Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39(7):11–16, July 1991.

[CF94] Flaviu Cristian and Christof 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.

[CF95] Flaviu Cristian and Christof 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.

[Che92] Wing-Hong Cheung. Improvements on Fidge logical clocks. In *TENCON '92. Technology Enabling Tomorrow 1992 IEEE Region 10 International Conference. Computers, Communications and Automation towards the 21st Century, Melbourne, Nov. 11-13, 1992*, pages 136–140, vol. I. IEEE, New York, 1992.

[CHK92] Janice E. Cuny, Alfred A. Hough, and Joydip 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.

[DB93] Rogério Drummond and Özalp Babaoğlu. Low-cost clock synchronization. *Distributed Computing*, 6(4):193–203, July 1993.

[DHHB87] 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.

[Dun91] Thomas H. Dunigan. Hypercube clock synchronization. Technical Report ORNL TM-11744, Oak Ridge National Laboratory, TN, February 1991.

[Dun94] Thomas H. Dunigan. Hypercube clock synchronization. ORNL TM-11744 (updated), September 1994. http://www.epm.ornl.gov/~dunigan/clock.ps.

[DW91] G.J.W. van Dijk and A.J. van der 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.

[EK94] Dennis Edwards and Phil 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.

[Fid88] Colin J. Fidge. Timestamps in message-passing systems that preserve partial ordering. In *Proceedings of 11th Australian Computer Science Conference*, pages 56–66, February 1988.

[Fid89] Colin J. Fidge. Partial orders for parallel debugging. *ACM SIGPLAN Notices*, 24(1):183–194, January 1989.

[Hof93a] Richard 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.

[Hof93b] Richard 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.

[HW88] Dieter Haban and Wolfgang 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.

[Jez89a] Jean-Marc 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.

[Jez89b] Jean-Marc Jézéquel. *Outils pour l'expérimentation d'algorithmes distribués sur machines parallèles.* PhD thesis, Université de Rennes, 1. Oct. 1989.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[LK93] William S. Lloyd and Phil Kearns. Tracing the execution of distributed programs. *Journal of Systems and Software*, 21(3):201–214, June 1993.

[Mat89] Friedemann 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.

[Mil92] David L. Mills. Network time protocol (version 3), specification, implementation and analysis. RFC 1305, Request for Comments, March 1992.

[MPI95] MPI Forum. *MPI: A Message-Passing Interface Standard*, June 12, 1995. http://www.mcs.anl.gov/mpi/.

[MSV91] Sigurd Meldal, Sriram Sankar, and James Vera. Exploiting locality in maintaining potential causality. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 231–239, 1991. Also Technical Report CSL-TR-91-446, Stanford University, April 1991.

[MT95] Eric Maillet and Cécile Tron. On efficiently implementing global time for performance evaluation on multiprocessor systems. *Journal of Parallel and Distributed Computing*, 28:84–93, 1995.

[PYS92] Robert L. Probert, Hualong Yu, and Kassem 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.

[Rab96] Rolf Rabenseifner. Die geregelte logical clock – definition, simulation und anwendungparalleler anwendungen in RUS - 30, Rechenzentrum der Universität Stuttgart, Mai 1996.
http://www.uni-stuttgart.de/People/rabenseifner/log_clock_rus30.html.

[Rab97] Rolf Rabenseifner. The controlled logical clock – a global time for trace based software monitoring of parallel applications in workstation clusters. In *Proceedings of the 5th Euromicro Workshop on Parallel and Distributed Processing – PDP '97*, pages 477–484. IEEE CS, Jan. 1997.
http://www.uni-stuttgart.de/People/rabenseifner/log_clock_pdp97.html.

[Ray87] Michel Raynal. A distributed algorithm to prevent mutual drift between n logical clocks. *Information Processing Letters*, 24:199–202, 1987.

[RS95] Michel Raynal and Mukesh Singhal. Logical time: A way to capture causality in distributed systems. Technical Report OSU-CISRC-4/94-TR13, Ohio State University, Computer and Information Science Research Center, 1995. //ftp://ftp.cis.ohio-state.edu/pub/tech-report/1995/TR13.ps.gz.

[RTV93] Maurice van Riek, Bernard Tourancheau, and Xavier-Francois Vigouroux. Monitoring of distributed memory multicomputer programs (A general approach to the monitoring of distributed memory MIMD multicomputers). Technical Report CS-93-204, University of Tennessee, 1993. //http://www.netlib.org/tennessee/ut-cs-93-204.ps.

[SC90] Frank Schmuck and Flaviu Cristian. Continuous clock amortization need not affect the precision of a clock synchronization algorithm. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, Quebec city, Canada, 22-24 August, 1990*, pages 133–143. ACM, 1990. Also as Technical Report RJ7290, IBM, January 1 1990.

[Sch87] Fred 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.

[SES89] André Schiper, Jorge Eggli, and Alain Sandoz. A new algorithm to implement causal ordering. In J.-C. Bermond and M. Raynal, editors, *Proceedings of the 3rd International Workshop on Distributed Algorithms,* LNCS 392, pages 219–232. Springer-Verlag, 1989.

[SK90] Mukesh Singhal and Ajay Kshemkalyani. An efficient implementation of vector clocks. Technical Research Report OSU-CISRC-11/90-TR34, Computer and Information Science Research Center, Ohio State University, 2036 Neil Avenue Mall, Columbus, Ohio 43210, 1990.

[SM94] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: in search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[YM93] Zhonghua Yang and Anthony Marsland, T.˙ Annotated bibliography on global states and times in distributed systems. *Operating Systems Review*, 27(3):55–74, July 1993.

[ZEA93] M. Zaki, M.Y. El-Nahas, and H.A. Allam. DPDP: an interactive debugger for parallel and distributed processing. *Journal of Systems and Software*, 22(1):45–61, July 1993.