БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ . BULGARIAN ACADEMY OF SCIENCES

КИБЕРНЕТИКА И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ • Том **2**, № 2
CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **2**, No 2

София . 2002 . Sofia

Clock Synchronization with Independent Adjustment in Distributed Systems

Krassimir Djambazov, Edita Djambazova

Institute of Computer and Communication Systems, 1113 Sofia
E-mails: kbd@iccs.bas.bg ead@iccs.bas.bg

Abstract: *The clock synchronization algorithm with independent adjustment is presented. It uses the cyclic nature of the TDMA strategy for continuous collection of time differences among the nodes' clocks and applies an independent adjustment. Nodes decide locally when to apply the adjustment. The correction term is calculated as in the fault-tolerant algorithms and is applied not periodically but upon exceeding one phase bit. Two variants for algorithms implementation are presented – fixed-point and checked adjustment.*

A classification of the clock synchronization algorithms is presented. Two main groups of algorithms are distinguished: with direct and with interactive synchronization. The fault tolerance support, referred to as convergent and consistent, is introduced as an upgrade of the basic algorithms. The proposed validity measure, local time rate of change, is used along with the skew distribution to estimate the influence of the failures and of the fault tolerance of the clock synchronization algorithms.

The algorithm is compared with other clock synchronization algorithms for distributed hard real-time control systems. A simulation model of clock synchronization algorithms that includes failure injection properties is used.

The reported results from the experiments with the simulator program give a base to make the conclusion that the proposed algorithm for clock synchronization with independent adjustment tolerates input omission, output omission, and bad clock failures to the same extent as the Fault-Tolerant Average algorithm.

Key words: *fault tolerance, clock synchronization, distributed systems, hard real time.*

1. Introduction

Keeping the clocks synchronous in distributed real-time process control systems is a crucial condition for their correct operation. It is even more important for distributed systems where the application poses hard real-time limits on the decision making process [10, 11].

A node of a distributed system is equipped with hardware (physical) clock and constructs some logical time units. The physical clock is composed of a pulse generator and counters. The content of the counters is readable and measures the time elapsed from a certain starting point. The value of the logical clock is visible for the other nodes only by passing its value in a message. The abstraction of logical clock is presented by a phase counter and a bit counter [18], forming microticks and macroticks, correspondingly. The logical clock value is displayed at bit level while the phase items are hidden.

Clock synchronization is a process of equalizing the logical clocks of all nodes to compensate the differences of their physical clock generators caused by the clock drifts. Some problems, especially in distributed systems, arise because of the uncertainty of the message delays in the system. Some other problems are caused by failures that could be exhibited either by the computers (including both physical and logical clocks) or by the communication channel.

The synchronization is usually performed periodically in resynchronization cycles (rounds) when every node reads the clocks of the others, calculates a target time and corrects its own logical clock. Between two resynchronizations the clocks are left to work with their specific frequencies and count factors.

Clock synchronization is made by adjustment of the clock value with a correction factor. Thus, the synchronization process could be divided into four phases: clocks values difference measurement, error handling, correction factor determination and correction factor application.

Two clocks are said to be synchronous if for any real time t their values differ by less than some predefined value Δ. A system is said to be synchronous if for any real time t no pair of clocks differ by more than some predefined value Δ .

The variety of clock synchronization algorithms and implementation protocols in many cases makes the design of such systems difficult, because of: (i) the gap between the theoretical results and implementation issues for most of the proposals and (ii) the different comparison bases used by the different authors and designers [20].

In an attempt to propose a solution of those problems a simulation model is developed [4]. The subjects of modeling are the synchronization algorithms themselves, the physical environment they are executed in, and the failures they are influenced by.

Section 2 of the paper presents a classification based on the most popular theoretical results and the corresponding synchronization algorithms, both fault sensitive and fault tolerant. The classification separates the task of synchronization and that of achieving fault tolerance. The algorithms are classified according to the way they determine the clock adjustment value, i.e., the correction factor.

Section 3 introduces a method for clock synchronization with independent adjustment that exploits the natural communication procedures instead of additional synchronization protocols in case of time-triggered distributed systems.

A simulation model, developed for estimation and design purposes, is presented in Section 4. Some results of modeling are given in Section 5 that demonstrate the fault-tolerant properties of the method proposed in comparison with some other methods.

1.1. Distributed clock synchronization

1.1.1. General description

The system under consideration is a fully connected network of nodes. The nodes execute local control tasks forming output signals to the local input stimuli, under the global system strategy. Each node measures the time in local time units – macroticks.

The hard real time requires the systems to be able to perform any control task in a limited period in spite of the number of tasks that could be active at the same time. This fact implies that the system is designed to meet peak load conditions without loss of timeliness [10, 19].

The described specifics led to the time-triggered approach [11, 12, 19] that obeys the hard real-time constraints via ensuring the worst-case communication traffic and the worst-case distributed computational load satisfies best the described specifics. In a time-triggered system, the elapsing of time intervals initializes all activities.

The time of a node is divided in cycles, each starting with polling of the inputs, going through calculation of a control output and finishing with an output reaction. Inside this control cycle a communication interval should be nested to meet the worst-case conditions. The communication interval is the only window where a node can transmit its data. Following this approach the system-level operation is organized in communication cycles. The communication cycle is divided into time-slots, each assigned to a node.

In the "synchrony by design" (time division multiple access [10]) the time points and the order of data transmissions are pre-scheduled for all nodes . This is used in the class of time-triggered protocols [10, 11, 12, 16, 19].

The common assumptions applied to the target system and its components are as follows.

The first assumption limits the drift rate of the physical clocks [1]. $H_i(t)$ represents the hardware clock of node i at real time t. The upper bound ρ defines the good and the bad clocks, the latter violate inequality:

$$\left|\frac{H_i(t_2)-H_i(t_1)}{t_2-t_1}-1\right| \le \rho \text{ for } t_2>t_1. \quad (1)$$

The second assumption requires monotonic time function of the local clocks [1]. $C_i^r(t)$ and $C_i^k(t)$ are the clock values of a node local clock i at real time t and in resynchronization cycles r and k, respectively:

$$C_i^r(t_1) < C_i^k(t_2) \text{ for } t_2>t_1 \text{ and } k>r. \quad (2)$$

Two properties define the synchronization:

- The agreement property of the synchronous clocks is defined by (3). The values of clocks i and j at real time t are denoted as $C_i^r(t)$ and $C_j^r(t)$, r is the resynchronization cycle number. The clock values difference Δ is often referred to as *skew*:

$$\left|C_i^r(t)-C_j^r(t)\right| \le \Delta. \quad (3)$$

- The following property is called *validity* and defines linear envelope of synchronized clock values; γ is an arbitrary small value:

(4) $$(1-\gamma)t \le C_i^r \le (1+\gamma)t\,.$$

Synchronization is achieved if both (3) and (4) are satisfied, given that (1) and (2) are met.

To evaluate the validity property we introduce a new measure: local-time change rate distribution θ [4]. The rate distribution is defined as the average of the rates differences with the real time progress (5). Analyzing the rate distribution, the effect of introducing fault-tolerant methods could be better assessed than when using only the skew estimations:

(5) $$\theta = \frac{C_i(t_{r+1}) - C_i(t_r)}{t_{r+1} - t_r}.$$

1.1.2. Impediments to clock synchronization

The factors that affect synchronization are reading errors and failures demonstrated by the nodes, communication channels and clocks.

A full reading error ε is structured as follows (Fig. 1): (i) time interval α between placing the time stamp and the starting point of transmitting, (ii) physical communication delay δ, and (iii) interval τ in the receiver between the first bit receiving and determining of the time point.

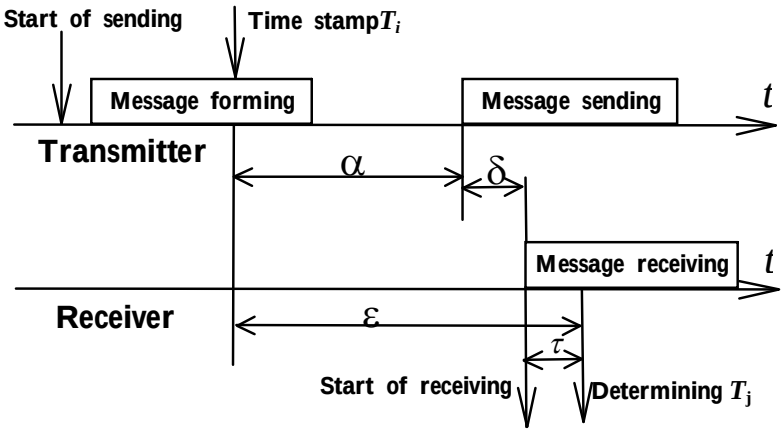


Fig. 1. Reading error

In general, the considerations are limited to partially synchronous systems, which implies the upper bounds of the delays to be known. This results in including the maximum reading error in the admissible clock difference Δ. Some more specific approaches are proposed for dealing with the different delays:

– The delay α could be reduced by a dedicated hardware that places the time stamp at the point of transmission [10].

– The delay δ could be directly measured by "round trip" method and therefore compensated for [2, 10, 16].

– The delay τ could be avoided by using a dedicated hardware determining the local clock value at the beginning of data reception.

1.1.3. Distributed measurement of time-related parameters

The measurement of time-related parameters is possible only on the basis of local clocks. We define the time-related measurements of any pair of nodes as comparison of two time periods: expected and observed. One and the same observed period differs in the different nodes.

We also assume that every non-faulty node follows its local time and starts message transmission exactly according to its time slot, thus announcing it to the others local time.

Another important feature is that the parameters are measured in pairs.

The parameters that could be measured are:

– Time difference between the expected time and the local point of receiving a message.

– Time interval between two successive received/transmitted messages from the same node.

One of the purposes of the measurement should be to distinguish between phase or clock differences and the message delay. Phase and clock differences could vary in different measurements while the message delay is constant and is always positive. It is also the same in both transmission directions of a pair of nodes.

Distributed control of time-related parameters

The parameters that could be locally controlled (observed) are listed bellow:

- Clock rate (frequency).

The clock rate could be controlled either continuously or digitally depending on the hardware used. The continuous control can be applied to the non-stabilized clock generators, where the phase difference is transformed into control voltage that specifies the frequency. A problem could arise because of the need to keep the last frequency during the pause between two successive messages.

- Time period (number of ticks). The time periods could be controlled by adding (subtracting) a corrective number of macroticksbits at given periods. The correction is asymmetric but this does not corrupt the proper adjustment. The length of controlled time periods could differ.

- Phase of the time period. The phase is controlled either immediately by presetting the phase counter, or continuously by suppression of the microticks. The immediate adjustment can be applied only if the correction value is less than one macrotick. The continuous adjustment prevents the clocks from being set back and therefore the correction value has no limits.

1.2. Failure model

1.2.1. Node failure modes

It is widely agreed that the failure model at system level comprises three kinds of failures of the nodes:

– *fail stop* (crash), when the node is silent in case of internal failure or because of local communication failure that prevents the node from access to the communication links;

– *omission*, when a message is either not received (input omission), or is not transmitted (output omission);

– arbitrary failure, when the faulty behavior of the node is not specified.

1.2.2. Clock failure modes [2]

- *Clock crash*, when a clock stops ticking or the counters do not count the clock ticks. In general, the clock crash cannot be associated with any specific system-level failure. The time-triggered systems, however, interpret the clock crash as fail stop.
- *Arbitrary failure*, when the faulty behavior of the clock is not specified. It could be non-linear counting or change of the drift rate of the physical clock, etc. This failure is regarded as bad clock in the paper.

1.2.3. Communication failure modes [2]

- *Late delivery* (performance error), when the message is delivered too late and its time stamp is not adequate to the receiving point because of bus access problems and traffic overload.
- *Arbitrary failure*, when physical noises prevent some receivers from correct reading of a message. An arbitrary failure is interpreted either as input omission failures if some nodes can receive and some cannot the corrupted message, or as an output omission failure if no node can receive the corrupted message.

Time-triggered systems are designed to operate correctly in the presence of arbitrary failures. For some studies, however, it could be more suitable to examine system behavior under specific operational conditions, with less complex failure types, such as input omission or output omission. This simplification does not reduce the range of the results, since the arbitrary behavior is decomposed into different failures.

1.2.4. Fault tolerance of clock adjustment algorithms

The fault-tolerance property could be regarded as independent of the synchronization algorithms. The main fault- tolerant algorithms are aimed at tolerating arbitrary failures. Most of the fault-tolerant algorithms discussed in the literature are aimed at tolerating a finite number of arbitrary faulty nodes. It is proved that at least $3m+1$ correct nodes are needed for m faults to be tolerated, if no authentication is applied, and $m+2$ correct nodes are needed otherwise [8].

Distributed fault tolerance is achieved by filtering the reported clock values in convergence methods and by data consistency in consistency methods (see Section 2).

The filtering could be independent or/and via co-operative agreement. The independent filtering is based on relative or absolute comparison of the incoming values.

- Relative comparison – the received time values are compared to each other, and:

(i) m largest and m smallest values are rejected, or

(ii)m values are rejected that are most distanced from the others (local time).

- Absolute comparison – values are rejected that lie out of some predefined bounds with respect to the value of the clock that makes the comparison. The bounds form an *acceptance window.*

It is proved that both methods are convergent and fault-tolerant [15, 16, 21].

The rejected values could be either replaced by the own value or ignored. It makes difference which rejection method is applied: in the first case the number of items is kept n, while in the second case the number of items is reduced by the number of rejected values.

2. Distributed clock adjustment – classification and methods

2.1. Clock adjustment principles (synchronization algorithms)

According to the methods of adjustment, the clock synchronization algorithms could be classified as: methods/algorithms with direct synchronization and algorithms with co-operative (interactive) synchronization [3].

2.1.1. Direct adjustment algorithms

The algorithms with selective direct adjustment discussed in [9, 14] use time stamped messages. The receiver of a message adjusts its local clock to the time stamp of the incoming message and if it is greater than its local clock, correction of the reading error is provided. The selective function ensures that: (i) no clock is set back than its current time, and (ii) the time of the fastest clock will be accepted by all local clocks as system time. Most of the algorithms in that class work as every node upon receiving a resynchronization message relays it to the other nodes, thus initializing their resynchronization.

The algorithms with unconditional direct adjustment synchronization [18] suppose that the receiver of a message adjusts its local clock to the time stamp of the incoming message. The algorithm is correct while the time difference is less than one macrotick communication bit.

2.1.2. Co-operative (interactive) adjustment algorithms

The algorithms with co-operative adjustment synchronization [13, 15, 17, 21] use time information collected from other system node members during the resynchronization phase. The algorithms should provide convergence of the synchronization targets. Two convergence methods are known for determination of the local clock correction on the basis of collected times:

- Average value [13, 15]. In the average value algorithms each node calculates the average of the collected differences and uses the result as a correction factor of its clock.
- Midpoint (median) value [21]. In the midpoint algorithm each node calculates the median of the collected differences and uses the result as a correction factor of its clock.

The consistent algorithms with co-operative adjustment apply an agreement protocol on the collected data to obtain consistency [15].

2.2. Background

The time-triggered distributed systems are of special interest for our study and the presented review is limited to their specifics. They operate under hard real-time restrictions. Message exchange, synchronization, etc., events in the system occur in pre-defined time points. All delays are limited and have known upper limit. This allows the system to operate according to a pre-defined schedule that assures its predictable behavior. From synchronization viewpoint, the known time points for message exchange and the constant delays in the communication channel ease the measurement of the time differences between the local clocks values and the application of algorithms for synchronization.

Fault-tolerant clock synchronization algorithms are convergence algorithms with co-operative adjustment by average value. They can be used in the process control applications where systems operate in a periodic manner. The control cycle could be combined with the communication cycle. The strategy “time-division multiple-access” (TDMA) [11] eases the communication. Nodes broadcast messages according to a pre-defined schedule. Each node has a time slot attached and can send messages only when its slot comes. During the resynchronization interval a node collects all time differences measured between the incoming messages and the expected arrival time points of the messages. In every resynchronization interval the fault-tolerant synchronization algorithm is applied.

The distributed system DACAPO [16, 18] also uses TDMA strategy. In DACAPO, however, a different approach is used for clock synchronization. The clock synchronization algorithm applies unconditional direct adjustment. It is not organized in rounds and does not require measuring and collection of the time differences of the nodes, and calculation of the adjustment. The clock adjustment is unconditional and direct. The receiving nodes synchronize their clocks to the node that is currently sending. This Daisy-Chain method is not fault-tolerant. A reception window is introduced to discard the messages that deviate more than the pre-defined size of that window.

The synchronization scheme in DACAPO is simple and does not require a complex hardware implementation. The clocks are synchronized on every message frame and oscillators with lower precision can be used.

The fault-tolerant synchronization algorithms are more complex, use frequency sources with high precision, but guarantee a tight synchronism among the nodes even in the presence of failures.

3. Clock synchronization with independent adjustment

The approach *clock synchronization with independent adjustment* [5] is not organized in resynchronization intervals. It uses the periodicity of the TDMA strategy for continuous collection of time differences among the clocks but the adjustment is independent. Nodes decide locally when to apply the adjustment. The resynchronization interval is individual for each node and is minimum one communication cycle in size. The correction factor is calculated as in the fault-tolerant algorithms but is applied when it exceeds one microtick (phase bit) and at least one communication cycle has passed since the last adjustment.

3.1. Presentation of the approach

The organization of time-triggered system operation is shown in Fig. 2. A communication cycle, CCi, is formed from the time slots of *N* nodes, each slot attached to a node in cyclic order. The access to the communication channel is based on the TDMA strategy. The figure shows a case where the data transmission (communication) phase is placed at the end of node calculation interval, when its internal state is just actualized. Nodes send only one message in their dedicated slots according to a pre-defined schedule. Thus, the time difference between the expected and the observed message arrival time is a measure of the skew between clock values of the sender and the receiver.

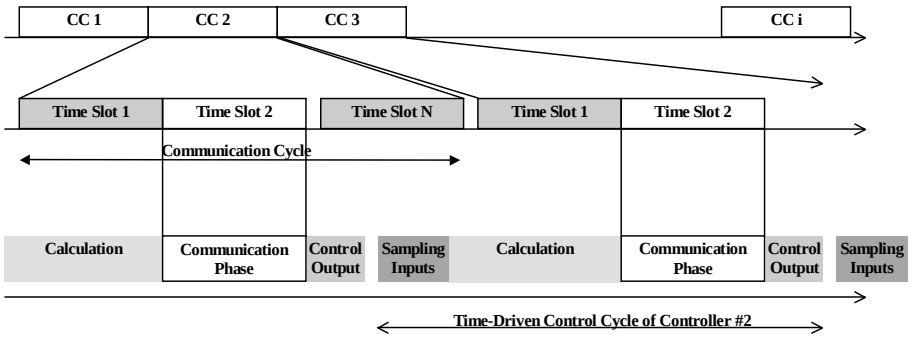


Fig. 2. Organization of system operation

The physical clock of each node produces a continuous sequence of pulses called phase bits. The actual time units in the distributed system are named information bits and consist of *x* phase bits (e.g. 16). The synchronization is purposed at keeping the phase difference between information bits of the nodes as small as possible, limited to ±1 phase bit.

The independent clock adjustment is based on three principles:

– continuous measurement of the differences between local clock value and clock values of the nodes;

– continuous calculation of the correction factor;

– independent and almost immediate adjustment if the correction factor is greater than one phase bit.

The continuous measurement and correction factor calculation do not violate the real-time constraints. These operations are included in the control cycle of time-triggered systems although they are executed only for resynchronization.

The continuous measurement means that every node compares its current clock value with the starting point of transmission of the other nodes. It is assumed that the starting point of transmission corresponds to the beginning of a slot. The difference between the expected and the actual starting point gives the difference (in phase bits) between the clock values of the transmitting and the receiving node. These differences are collected and used for calculation of the correction factor. The function of calculation could be average value [13] or midpoint [21]. Every time a node receives a message, it calculates the correction factor and if it exceeds one phase bit the adjustment is applied depending on some conditions.

Independent adjustment approach differs from the approach used in [12] by the following characteristics:

- Nodes apply the adjustment independently of each other. The existence of the necessary conditions is decided on locally.
- The independent adjustment is based on the individual (local) view of the nodes for the correction of the clocks' values. The process of measurement, computation and application of the correction factor is fully distributed.
- The resyncronization interval is shorter (the shortest possible for co-operative adjustment) than that in [11]. This allows the use of crystal oscillators with lower precision.
- Shorter period of adjustment keeps the clocks in tight synchronism.

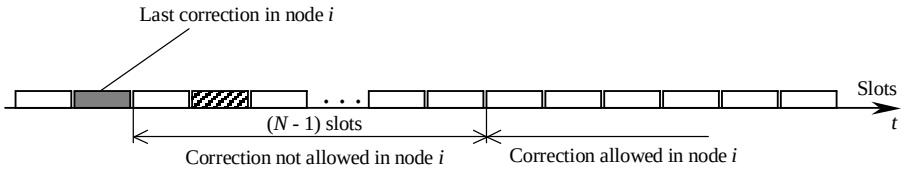
• The independent adjustment does not require a scheme for continuous resynchronization and/or a scheme for compensation of the fractional part of the correction factor [11].

3.1.1. Fault-tolerant independent adjustment

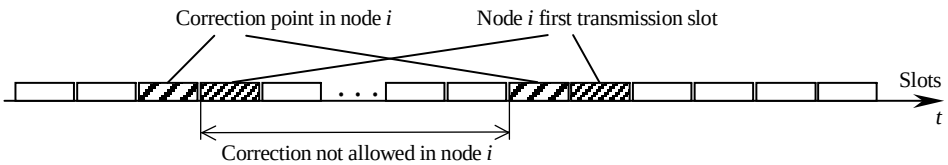
The fault tolerance is achieved via convergence algorithms with co-operative adjustment. In these algorithms, *m* highest and *m* lowest values are discarded from the sorted list of collected differences *m* faults to be tolerated. The rejection of the end values is based on the assumption that the faulty nodes deviate greatly from the ensemble and, thus, their values cannot influence the synchronization.

3.1.2. Implementation of the independent adjustment

Since the independent adjustment uses the collected differences, it could not be applied before all the differences are actualized, i.e., after one communication cycle. Each node measures the difference with every other once in a communication cycle. If a node changes its phase, the next cycle of measurements will be completed after $N-1$ time slots, with N nodes in the system. During this period the measured data will be referred to the previous phase or to a mixture of old and new measurements. Following the technique of independent adjustment, it could happen that the same correction factor is applied incorrectly more than once. To avoid this effect we propose two schemes for adjustment application:



a. Checked independent adjustment



b. Fixed-point independent adjustment

Fig. 3. Independent adjustment

• The adjustment is allowed when N new differences after the last adjustment are collected (checked adjustment) (Fig. 3a).

• The adjustment is allowed if the node is the next to transmit and if the transmission is its first in the communication cycle (fixed-point adjustment) (Fig. 3b). The second condition concerns cases when a node has more than one transmission in a system communication cycle.

3.2. Fault tolerance of the approach

To demonstrate the fault tolerance of the clock synchronization with independent adjustment it for one failure type is studied [3, 4]. The average skew between the local clocks values is examined for transient bad clock failures. Transient bad clock failure is a single non-linear change of clock value.

3.2.1. Assumptions and conditions

A system of N nodes tolerating m faults is considered. The time difference between two clocks in phase bits (the skew between their clock values) is Δ. The average skew is determined as a function of N, m, and Δ. The analysis is carried out for the case of a failure in one node clock.

The algorithms with checked independent adjustment, with fixed-point independent adjustment, and with resynchronization [11] are analyzed.

The effect of the transient bad clock failure lasts for maximum two communication cycles, $2N$ slots. The average skew is derived under the following conditions:

- Clocks drifts are excluded in order to study the pure effect of the failure.
- The bad clock failure is observable at system level in the next slot after its occurrence.
- The faulty node measures time difference Δ with all other nodes.
- Since only one faulty node is considered, the other nodes are synchronized and measure $\Delta \neq 0$ only with the faulty node. The faulty node collects non-zero differences and can adjust its clock not earlier than m slots after the failure because of the fault-tolerant algorithm adopted.
- The correction factor is greater than one phase bit during the time interval of $2N$ slots.

The average skew shows how big is the time difference of the failed node clock from the ensemble.

3.2.2. Average algorithm for clock adjustment

Checked independent adjustment. In the checked adjustment, the correction is made when N new differences after the last adjustment are collected and the correction factor is greater than one phase bit. During the first m slots the correction factor is zero because the highest m and lowest m values are rejected from the sorted list of time differences. The first non-zero calculation is made in the $(m+1)$-st slot after the failure occurrence. If the faulty node is the one to transmit in one of the first m slots, it does not measure any difference in its attached slot. Thus, it collects $m+1$ time differences in $m+2$ slots. The first two terms of expression (6) reflect this consideration. When the faulty node applies the adjustment in $(m+1)$-st slot, it differs from the others with $\left(1-\frac{1}{N-2m}\right)\Delta$ during the next communication cycle, i.e., for N slots. The average skew for checked independent adjustment is expressed as follows:

$$\Delta_{av} = \frac{1}{2N}\left[\frac{(N-m)}{N}(m+1)+\frac{m}{N}(m+2)+N\left(1-\frac{1}{N-2m}\right)\right]\Delta. \quad (6)$$

Fixed-point independent adjustment. In the fixed-point adjustment, the correction is made if the node is the next to transmit and if the transmission is its first in the

communication cycle. The interval from correction to correction is always *N* slots despite of the faulty node number. The value of the correction factor, however, depends on how many slots before the node slot the failure has occurred. The first term in expression (7) reflects the time difference of the failed node at its first adjustment. The second term shows the skew at the second adjustment:

$$\Delta_{av} = \frac{1}{2N^2}\left[\sum_{i=1}^{N}(N-i) + \sum_{i=m}^{N-m-1}\left(1-\frac{N-m-i}{N-2m}\right) + N + m + 1\right]\Delta. \quad (7)$$

Fault-tolerant average algorithm with resynchronization. The clock synchronization is made in resynchronization intervals, common for all nodes. The value of the correction factor depends on the number of the faulty node. This changes only the second term in expression (7) and the average skew is

$$\Delta_{av} = \frac{1}{2N^2}\left\{\sum_{i=1}^{N}(N-i) + \sum_{i=m}^{N-m-1}\left[i\left(1-\frac{N-m-i}{N-2m}\right) + (N-i)\left(1-\frac{N-m-i-1}{N-2m}\right)\right] + N + m + 1\right\}\Delta.. \quad (8)$$

3.2.3. Midpoint algorithm for clock adjustment

In the midpoint algorithm for clock adjustment, the correction factor does not depend on the node number, nor on the number of the collected time differences. The expressions for the average skew with midpoint algorithm applied are:

checked independent adjustment –

$$\Delta_{av} = \frac{1}{2N}\left[\frac{(N-m)}{N}(m+1) + \frac{m}{N}(m+2) + \frac{N}{2}\right]\Delta; \quad (9)$$

fixed-point independent adjustment –

$$\Delta_{av} = \frac{1}{2N^2}\left[\sum_{i=1}^{N}(N-i) + \frac{(N-2m)}{2} + N + m + 1\right]\Delta; \quad (10)$$

fault-tolerant average algorithm with resynchronization –

$$\Delta_{av} = \frac{1}{2N^2}\left[\sum_{i=1}^{N}(N-i) + N\frac{(N-2m)}{2} + N + m + 1\right]\Delta. \quad (11)$$

Results based on expressions (6)–(11) are presented in Figs. 4 and 5 for the Fault-Tolerant Average (FTA) algorithm and in Figs. 6 and 7 for the Midpoint (MP) algorithm for clock adjustment. The average skew is depicted as a fraction of the skew on the *Y* axis.

It can be seen from Figs. 4 and 5 that the algorithm with fixed-point independent adjustment has the smallest average skew of the algorithms under consideration when FTA algorithm for clock adjustment is applied. The algorithm with checked independent adjustment has greater skew than the algorithm with resynchronization. For small values of the number of tolerated failures (Fig. 4), however, the average skew of the algorithm with checked independent adjustment is close to that of the algorithm with resynchronization.

For the MP algorithm for clock adjustment there are cases in which the algorithm with checked independent adjustment has better average skew than that with resynchronization (Fig. 6). This is especially true for a small number of tolerated failures (Fig. 7).

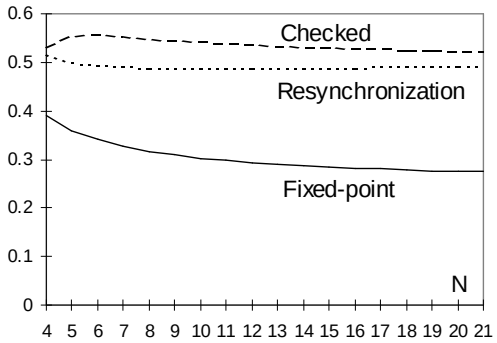


Fig. 4. Average skew as a function of m, FTA, N =1

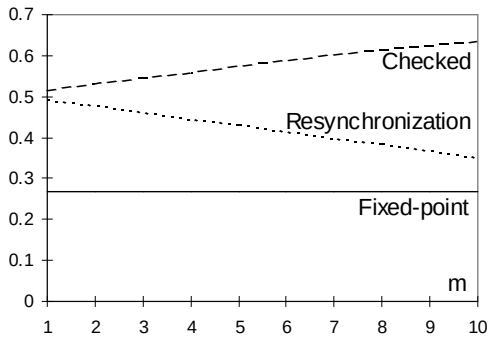


Fig.5. Average skew as a function of m, FTA, N=31

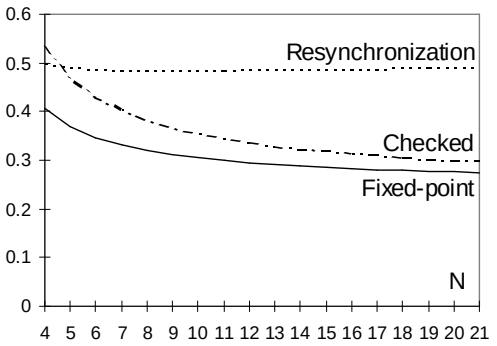


Fig. 6. Average skew as a function of N, MP, m=1

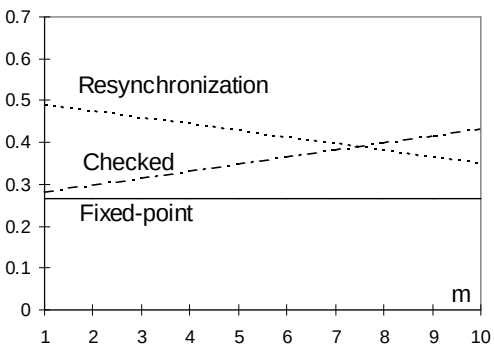


Fig. 7. Average skew as a function of m, MP, N=31

The algorithm with fixed-point independent adjustment is not influenced by the number of tolerated failures m (Figs. 5 and 7). The algorithm with checked independent adjustment, however, shows a skew increasing with m, despite of the algorithm for clock adjustment (Figs. 5 and 7).

4. Simulation

4.1. Simulator

The simulator program used for the experiments includes a number of built-in algorithms. It comprises several models incorporated in a common environment:

– time presentation model – models the progress in real and local time units on the basis of given clock drifts;

– events generator – generates and orders the events in real time and initializes their modeling;

– error model – generates error demonstrations;

– synchronization algorithms model – models different kinds of synchronization algorithms.

4.2. Simulated system

A system including six nodes is modeled. It is fully connected through LAN, i.e., all nodes can read a message sent by one of them. The drift rate of the clocks is in the range

ρ = + 5.10^{-5} for permanent failures, ρ = + 5.10^{-4} for transient failures. The system tolerates one failure, i.e., m = 1. The failure model includes full input omission, output omission, and bad clock failure.

The following algorithm cases are included: direct synchronization, interactive convergence with average value, and interactive convergence with midpoint value. The fault tolerance is achieved by highest and lowest values rejection.

The studied failure cases include: no failures, permanent bad clock, transient bad clock, permanent full input omission, transient full input omission, permanent output omission, transient output omission.

The clock synchronization algorithms are compared by their skew and validity properties.

5. Results

5.1. Experimental results for permanent failures

The presented two variants for independent adjustment are combined with the fault-tolerant algorithms with average value and midpoint value. The results are compared to the time-triggered system with resynchronization [11].

Figs. 8 – 10 present the simulation results of the skew distribution in percentages. The skew scale is in phase bits. The diagrams shown are only for the Fault-Tolerant Average (FTA) algorithm. The results are similar when using the Midpoint (MP) algorithm.

Bad clock is modeled as clock whose drift rate is out of the range of normal clocks drifts.

In the figures, diagram S1 represents the skew of the clock synchronization algorithm with fixed-point independent adjustment. Diagram S3 is for the algorithm with checked independent adjustment. Diagram S2 demonstrates the case of time-triggered approach with resynchronization interval equal to the communication cycle. The coordinate system in the figures for skew distribution depicts the number of phase bits on the X axis and their percentage on the Y axis.

It can be seen from the figures that all algorithms tolerate the examined failure types.

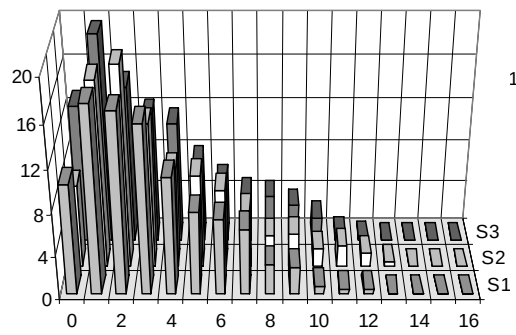


Fig. 8. Skew distribution upon input omission failure

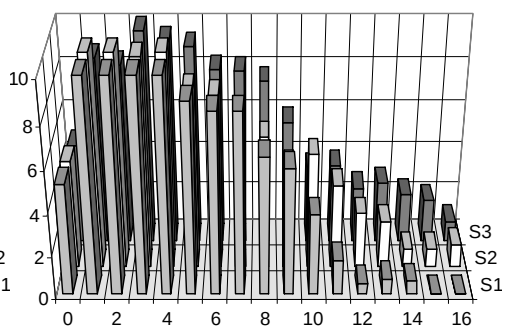


Fig. 9. Skew distribution upon output omission failure

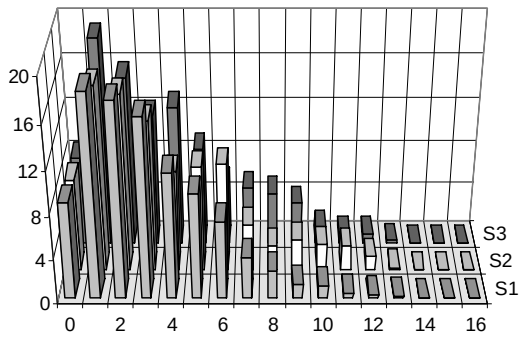


Fig. 10. Skew distribution upon bad clock failure

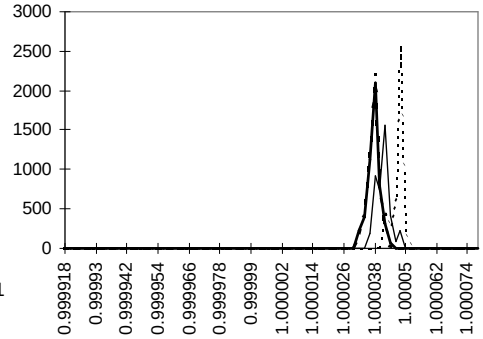


Fig. 11. Local-time change rate for time-triggered system with resynchronization

The output omission failure causes bigger deviation of the clocks (Fig. 9) because the faulty node applies the adjustment but the other nodes do not measure a difference with it and do not take into account its clock value.

Figs. 11–13 show the results for the rate of changing of the local clocks, θ [4], during synchronization under bad clock failure. The rate of local clock change θ represents the ratio between local and global (real) time.

In Figs. 11–13, the diagrams representing the case without failures are plotted with fat line and those for the case with bad clock failure are plotted with thin line. A distinction is made between the cases with and without fault tolerance. The results are for the FTA. When the highest and lowest values are rejected the charts are plotted with flat lines and they are plotted with dotted lines otherwise.

When there are no failures in the time-triggered system the application of fault tolerance does not change the local time (fat curves in Fig. 11 are very close). Failures do not increase the local clock rate of change, they only make the distance from real time bigger.

The rate of local clock change in the no failure case increases when fault tolerance is applied through the algorithms with independent adjustment (fat curves in Figs. 12 and 13). In case of a failure, however, the application of fault tolerance not only decreases the rate of local clock change but it keeps the values close to those for the case without failures (thin curves in Figs. 12 and 13).

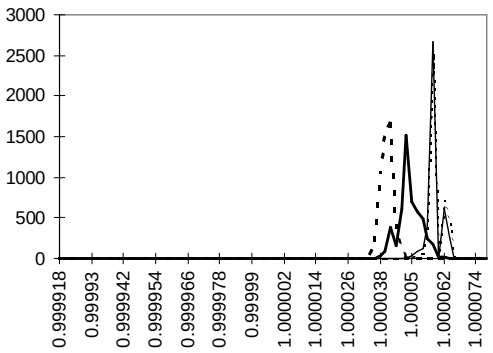


Fig. 12. Local-time change rate for checked independent adjustment

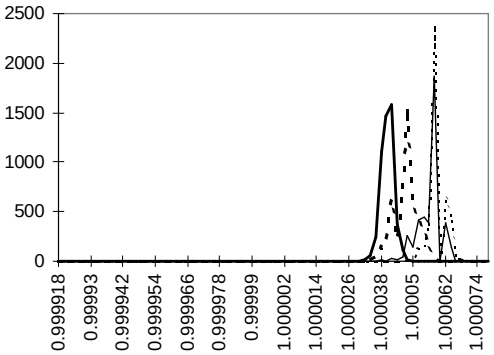


Fig. 13. Local-time change rate for fixed-point independent adjustment

5.2. Experimental results for transient failures

The basic concepts of simulation for transient failures are:

- Transient failures are simulated as a Poison process.
- The duration of a transient failure in slots is defined as one-sided normal distribution with mean and standard deviation equal to the failure length.
- The number of simultaneously active transient failures is limited to *m*, the number of tolerated failures.
- It is assumed that a failure does not affect directly the communications among the non-faulty nodes.

Transient input and output omission failures are studied for duration more than one slot, since failure with duration one slot does not affect significantly the synchronization. Transient bad clock failure is defined as a single non-linear change of clock value.

The presented experimental results are obtained under the following conditions:

– Time slot of 100 bits (1 bit = 16 phase bits);

– Failure rate $\lambda = 10^2$ s^{-1};

– Period of simulation 10^6 slots;

– Simulation in 10 runs with different drift rates.

Transient bad clock failures are examined for failure values 100, 400, and 800 phase bits.

The presented two variants for independent adjustment are combined with the fault-tolerant algorithms with average value and midpoint value. The results are compared to the FTA algorithm with resynchronization [11] and resynchronization interval equal to one communication cycle.

Figs. 14 and 15 show the diagrams for skew distribution in case of input and output omission, respectively, when an average function is applied for correction factor calculation. The results are very close when a midpoint function is applied to calculate the adjustment. Figs. 14 and 15 show that input and output omission failures are tolerated similarly by the studied algorithms. The two variants of the algorithm with independent adjustment show slightly better skew distribution than the FTA algorithm with resynchronization.

Figs. 16–18 show the skew distribution for the Midpoint (MP) algorithm for clock adjustment in case of bad clock (BC) failure. Three failure values are examined: 100 phase bits (Fig. 16), 400 phase bits (Fig. 17), and 800 phase bits (Fig. 18).

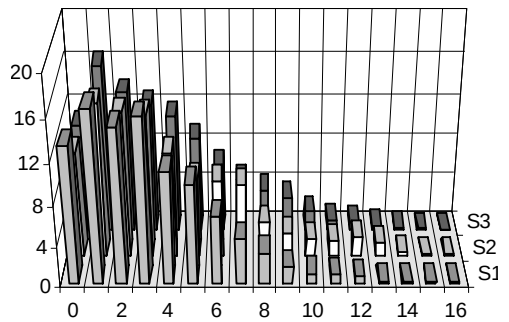


Fig. 14. Input omission in more than one slot

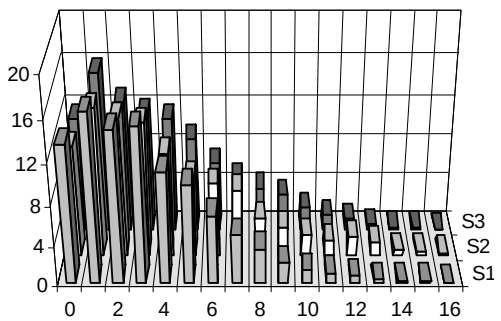


Fig. 15. Output omission in more than one slot

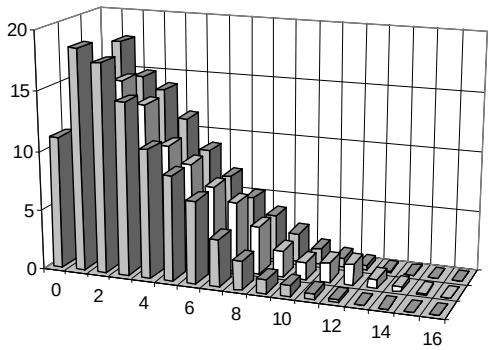


Fig. 16. Skew distribution for BC failure value 100, MP

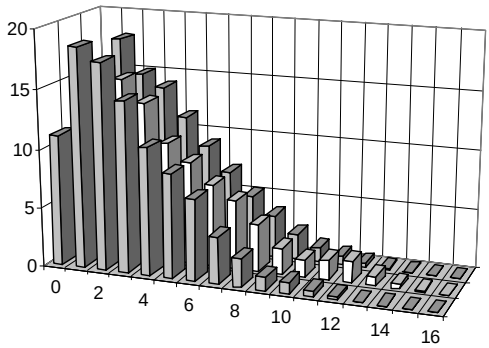


Fig. 17. Skew distribution for BC failure value 400, MP

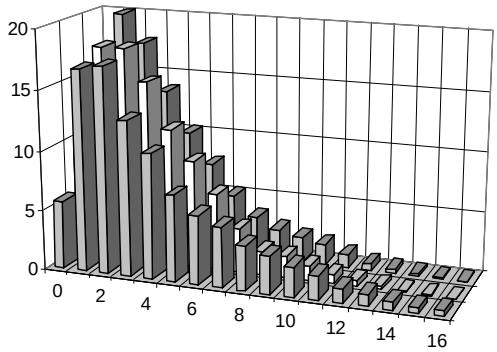


Fig. 18. Skew distribution for BC failure value 800, MP

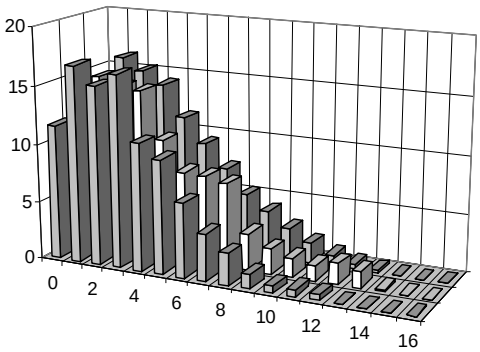


Fig. 19. Skew distribution for BC failure value 100, FTA

Transient bad clock failures have slight influence on the skew distribution when MP algorithm for clock adjustment is applied (Figs. 16–18). Only when the failure value is big (Fig. 18) the skew distribution shifts to the low values of the skew. The distribution for the algorithm with independent adjustment is better than that for the algorithm with resynchronization, except for the value of 800 phase bits.

The results for the Average (FTA) algorithm for clock adjustment are presented in Figs. 19–21 for the same failure values. As for the MP algorithm, the skew distribution

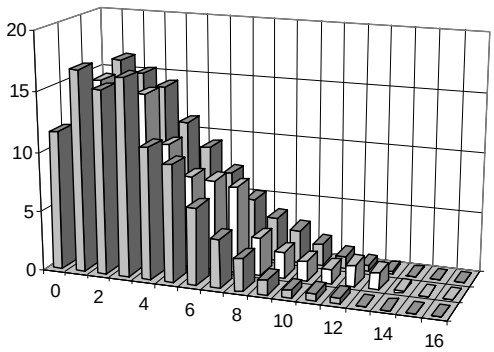


Fig. 20. Skew distribution for BC failure value 400, FTA

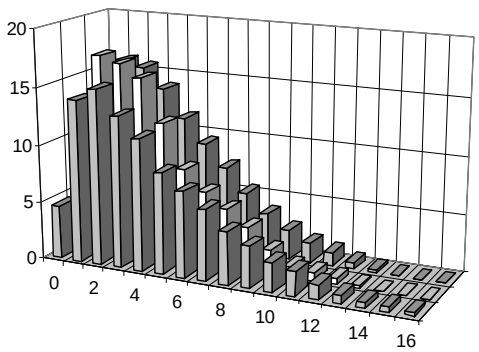


Fig. 21. Skew distribution for BC failure value 800, FTA

is affected only for failure value of 800 phase bits but even in that case it is less influenced than for the MP algorithm.

The slight effect of the transient bad clock failure on the skew distribution is due to the small value of the ratio of the time for resynchronization of the failed node and the time between failures.

A generalization of the skew as a measure for clock synchronization algorithms in the presence of bad clock failures is its mathematical expectation (Table 1).

Table 1

Failure value	0			100			400			800		
	checked	fixed-point	TTP	checked	fixed-point	TTP	checked	fixed-point	TTP	checked	fixed-point	TTP
FTA	3.63	3.18	3.95	5.51	4.68	5.91	11.46	9.39	12.13	19.35	16.78	19.75
MP	3.31	3.09	3.90	5.31	4.59	5.83	11.04	9.32	12.02	18.56	16.63	19.52

The results in the table show that the two algorithms studied with independent adjustment have better mathematical expectation of the skew than the FTA algorithm for system with resynchronization for all failure values. It can also be seen that the average and the midpoint algorithms for clock adjustment show close values of the expectation, with the MP algorithm demonstrating lower skews.

6. Conclusion

Algorithms for clock synchronization in distributed time-triggered control systems are studied. They are classified according to their adjustment principles. Two main groups of algorithms are distinguished: with direct and with interactive synchronization. The fault tolerance support, referred to as convergent and consistent, is introduced as an upgrade of the basic algorithms. A validity measure is proposed – local time rate of change. Along with the skew distribution it is used to study the influence of the failures and that of the fault tolerance of the clock synchronization algorithms.

The algorithms are studied with a simulation model that includes failure injection properties.

A clock synchronization algorithm with independent adjustment is presented. It uses the periodicity of the TDMA strategy for continuous collection of time differences among the nodes' clocks and their independent adjustment. Nodes decide locally when to apply the adjustment. Two variants for algorithm implementation are proposed: fixed-point and checked adjustment. The presented algorithms are studied formally.

The experimental results with the simulation model showed that the clock synchronization algorithm with independent adjustment tolerates the studied failure types at least to the same extent as the fault-tolerant average algorithm for time-triggered systems. The implementation of the new approach implies simpler solutions, preserving the effective fault tolerance.

References

1. Babaoglu, O., R. Drummond. (Almost) No cost clock synchronization. – In: Proc. FTCS-17, 1987, 42-47.
2. Cristian, F., C. Fetzer. Probabilistic Internal Clock Synchronization. University of California, San Diego, Tech. Report CS94-367, June 1996.
3. Djambazov, K., E. Djambazova. Distributed clock adjustment in real-time systems. – In: Proc. of National Conference "Automatics and Informatics '98", Sofia, Oct. 1998, 33-36.
4. Djambazova, E., K. Djambazov. Clock synchronization in distributed systems – simulation results. – In: Proc. of National Conference "Automatics and Informatics '98", Sofia, Oct. 1998, 29-32.
5. Djambazova, E., K. Djambazov. Clock synchronization with independent adjustment in distributed time-triggered systems. – In: Proc. of National Conference "Automatics and Informatics '99", Sofia, Oct. 1999, 60-64.
6. Djambazova, E., K. Djambazov. Clock synchronization with independent adjustment in distributed systems: Simulation results. – In: Proc. of the International Scientific Conference "Communication, Electronic, and Computer Systems '2000", May 2000, 96-101.
7. Djambazov, K., E. Djambazova. Fault tolerance of clock synchronization algorithms. – In: Proc. of CompSysTech '2000, June 2000, II.2-1-II.2-5.
8. Dolev, D., J. Halpern, H. R. Strong. On the possibility and impossibility of achieving clock synchronization. – Journal of Computer and System Sciences, **32**, 1986, No 2, 230-250.
9. Halpern, J., B. Simons, R. Strong, D. Dolev. Fault-tolerant clock synchronization. – In: Proc. 3rd Annual ACM Symp. on Principles of Distributed Computing, August 1984, 89-102.
10. Kopetz, H. et al. Distributed fault-tolerant real-time systems: The MARS approach. – IEEE Micro, February 1989, 25-40.
11. Kopetz, H., A. Kruger, D. Millinger, A. Schedl. A synchronization strategy for a time-triggered multicluster real-time system. – In: Proc. 14th IEEE Symp. on Reliable Distributed Systems, September 1995.
12. Kopetz, H., G. Grunsteidl. TTP – A Time-triggered protocol for fault-tolerant real-time systems. – In: Proc. FTCS-23, 1993, 524-533.
13. Kopetz, H., W. Ochsenreiter. Clock synchronization in distributed real-time systems. – IEEE Trans. on Computers, **C-36**, August 1987, No 8, 933-940.
14. Lamport, L. Time, clocks, and the ordering of events in a distributed real-time systems. – Communications of the ACM, **21**, July 1978, No 7, 558-565.
15. Lamport, L., P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. – Journal of the ACM, **32**, January 1985, No 1, 52-78.
16. Lonn, H., R. Snedsbol. Synchronization in safety-critical distributed control systems. – In: Proc. IEEE Int. Conf. On Algorithms and Architectures for Parallel Processing, Brisbane, Australia, 1995.
17. Mahaney, S., F. Schneider. Inexact agreement: Accuracy, precision and graceful degradation. In: – Proc. 4th Annual ACM Symp. on Principles of Distributed Computing, August 1985, 237-249.
18. Rostamzadeh, B. et al. DACAPO: A distributed architecture for safety-critical control applications. – In: IEEE Int. Symp. on Intelligent Vehicles, Detroit, USA, 1995.
19. Rushby, R. Systematic formal verification for fault-tolerant time-triggered algorithms. – Extended version of a paper from the 6th Working Conf. On Dependable Computing for Critical Applications, March 1997, 203-222.
20. Simons, B., J. Lundelius Welch, N. Lynch. An overview of clock synchronization. – In: Fault-Tolerant Distributed Systems (B. Simons and A. Spector, Eds.), LNCS, No 448, 1990, 84-96.
21. Lundelius Welch, J., N. Lynch. A new fault-tolerant algorithm for clock synchronization. – Information and Computation, **77**, April, No 1, 1988, 1-36.

Синхронизация на часовници с независима корекция в разпределени системи

Красимир Джамбазов, Едита Джамбазова

Институт по компютърни и комуникационни системи, 1113 София

(Р е з ю м е)

Представен е алгоритъм за синхронизация с независима корекция. Той използва цикличността на стратегията с времеделене и множествен достъп до магистралата за непрекъснато събиране на времеви разлики между часовниците на възлите в системата и прилага независима корекция. Възлите решават локално кога да приложат корекцията. Коригиращият фактор се изчислява както при отказоустойчивите алгоритми, но се прилага, когато стойността му надвиши един фазов бит. Представени са два варианта за приложение на алгоритъма – фиксирана и отложена корекция.

Представена е и класификация на алгоритмите за синхронизация. Разграничени са две основни групи алгоритми: с директна и със съвместна корекция. Поддържането на отказоустойчивостта, разграничено като конвергентно и консистентно, е представено като надстройка на базовите алгоритми. Предложената мярка на валидност (скорост на изменение на локалното време), се използва заедно с разпределението на отместването за оценка на влиянието на отказите и на отказоустойчивостта на синхронизационните алгоритми.

Представеният алгоритъм е сравнен с други алгоритми за синхронизация в разпределени системи за управление с твърди времеви ограничения. Използван е симулационен модел на алгоритмите за синхронизация, който включва възможности за инжектиране на откази.

Показаните резултати от експериментите със симулаторната програма дават основание да се направи изводът, че представеният алгоритъм за синхронизация с независима корекция толерира откази от типа пропускане по вход, пропускане по изход и неизправен часовник поне в същата степен като отказоустойчивия алгоритъм с осредняване.