International Journal of Database Theory and Application
Vol. 2, No. 2, June 2009

Network Issues in Clock Synchronization on Distributed Database

Rumpa Hazra[1], Debnath Bhattacharyya[1], Shouvik Dey[2], Sattarova Feruza Y.[3], and Tadjibayev Furkhat A.[4]

[1]*Heritage Institute of Technology, Kolkata-700107, India*
[2]*IBM India Pvt. Ltd., Kolkata, India*
send2shouvik@gmail.com
[3]*Hannam University, Daejeon, Korea*
[4]*Lindenwood University, St. Charles, MO, USA*
{send2rumpa,debnathb}@gmail.com[1], {mymail6585, furkhat}@gmail.com[3,4]

Abstract

Synchronization of clocks in distributed system has been an important area of research over the last decade. There are various methods of achieving clock synchronization depending on the requirements of the situation. A clock synchronization service ensures that spatially dispersed and heterogeneous processors in a distributed system share a common notion of time. In order to behave as a single, unified computing resource, distributed systems have need for a synchronization of drifting clocks and several algorithms have been proposed on this topic. Our paper highlights total network connection between different processors (alias nodes) of the system,

We provide a simple, efficient, and unified algorithm for clock synchronization that can minimize the total network connection of the system.

Keywords: *Database, clock, distributed, operating system.*

1. Introduction

A distributed system consists of a set of processors that communicate by message transmission and that do not have access to a central clock. Nonetheless, it is frequently necessary for the processors to obtain some common notion of time, where “time" can mean either an approximation to real time or simply an integer-valued counter. The technique that is used to coordinate the notion of time is known as clock synchronization.

Synchronized clocks are useful for many reasons. Often a distributed system is designed to realize some synchronized behavior, especially in real-time processing in factories, aircraft, space vehicles, and military applications. If clocks are synchronized, algorithms can proceed in “rounds" and algorithms that are designed for a synchronous system can be employed. Synchronization of individual clocks also becomes the more important in the case of hard real – time systems, where predictable performance is the foremost concern and one need to preserve a total logical/temporal ordering of tasks in the system.

Each processor (alias node) in a distributed system has its own hardware clock. Physical clocks normally drift due to temperature changes etc. For example, if 2 processor nodes have hardware clocks having a stability of ±5 ppm (parts per million), they can drift by as much as 10 μs in one second of real time. Now let us consider the major problems that can arise, if the clocks are not synchronized:

a. In a distributed system, the sensor data acquisition, which is carried out in one processor must always maintain a fixed timing relationship with the algorithms, which process this sensor data. These algorithms may typically be performed in a set of different processors. This fixed timing relationship can not be maintained unless all clocks of the processors partitioning in the execution of the algorithm are synchronized. Similar arguments hold good for the processors, which are entrusted with the job of data distribution to the launch vehicle control system actuators.

b. There may exist a precedence relationship among tasks in different processors. Say, task A in one processor can start execution only upon the completion of Task B in another processor. Since the pre-run time scheduling algorithms [11], which are commonly used to schedule tasks in different processors use the logical clocks maintained by each individual processor, the only way to guarantee the precedence relationship among tasks distributed across processors is to maintain a good clock synchronization among processors.

Clock synchronization is achieved normally by two distinct methods, viz., external synchronization and internal synchronization. External synchronization tries to maintain processor clock within a specified deviation from an externally maintained time reference, using phase locked oscillators [8]. For this purpose, each processor should be connected to a stable clock through an external links, to provide nanosecond level accuracy to physical clocks. One needs to execute one pass of a synchronization algorithm initially or whenever a new processor joins in. But Cesium or Rubidium based stable automatic clocks are expensive and to carry them on board satellite launch vehicles makes the overall cost prohibitive. These issues are also highly relevant in the case of automotive industry where expensive hardware clocks in each node may not be cost-effective. More over, in a distributed system, it may well be near impossible to use external clocks, as the cables interconnecting these clocks introduce distortions, which could be higher than the inherent stability of the external clock. Internal clock synchronization addresses these problems by using software algorithms which ensures that the logical clocks used by the processors in different nodes are consistent within limits, irrespective of the drift in the physical clocks.

2. Previous works

Lamport and Melliar-Smith [5] have presented algorithms, which maintain clock synchronization under the assumption that clocks are initially synchronized. In the Interactive Convergence Algorithm a node sets its clock to the average of all clock values after discarding bad clock values (those which lie farther than a specified value from its own clock). The more complex Interactive Consistency Algorithms use median rather than mean, and require a great deal of message passing among the participating nodes. These algorithms achieve clock synchronization even under Byzantine faults and need 3m+1 processors and m+1 round of message passing to handle up to m faults.

Koptez and Oshenreiter [4] have proposed the ‘Mars approach towards clock synchronization.’ They use Fault-Tolerant Averaging (FTA) algorithm in which the highest as well as lowest f clock values are discarded and the remaining clock values are averaged to resynchronize. This method is able to tolerate up to f faulty clocks.

Christian [2] has suggested a clock synchronization algorithm, which is based on probabilistic arguments. This measures the round trip delay by sending a clock read request to a remote node. The remote clock value is then estimated based on this measured delay. This

algorithm claims to guarantee smaller clock synchronization errors as compared to deterministic algorithms.

Srikanth and Toueg [10] have presented a clock synchronization algorithm, which can tolerate different types of failures including arbitrary ones. This algorithm is given in two forms, viz., with and without message authentication. Synchronization is achieved by periodically adjusting the logical clocks forward by executing this algorithm. Here the values of the logical clocks can jump to a higher value and no logical clock value is moved backwards, thus avoiding the repetition of same logical clock values. The algorithm is optimal in the sense of accuracy as well as the number of faulty processors that can be tolerated and is useful in many situations. A transputer based fault-tolerant implementation and timing analysis of [9] is presented in Sinha et al [8].

Dhruba Basu and Sasikumar Punnekkat [12] have suggested two simple clock synchronization algorithms, which can overcome some problems that arise out of the clock synchronization algorithms of Srikanth and Toueg[10]. This algorithm is given in two forms, viz., with and without fault. According to Srikanth and Toueg [9] no logical clock is ever adjusted backwards to prevent the occurrence of the same clock value twice. The main problem here is that logical clocks of slower processors are forced to jump forward. This could result in unfinished or unscheduled tasks in some processors with a very high utilization factor, which in turn could lead to unpredictable system behavior Srikanth and Toueg [10]. Lamport and Melliar-Smith [5] have briefly mentioned these problem in an earlier work. According to Dhruba Basu and Sasikumar Punnekkat [12] Synchronization is achieved with respect to the slowest among the correct clocks. This ensures that there is always a continuity of logical clock values and there is no need to cause the logical clock to be moved either forward or backward. This is an extremely important property for ensuring performance in most of the safety critical hard real-time systems. This also guarantees that no task release points are missed. The precedence constraints between tasks on different processors are preserved automatically.

The major argument against synchronizing with the slowest clock could be that, it necessitates logical clock of the processors with faster physical clocks to jump backwards, thus repeating the same logical time values again. This problem can be solved, by modifying processes, which manage the maintenance of logical clocks. Dhruba Basu and Sasikumar Punnekkat [12] have presented algorithm-A, which can overcome the above problem under the assumption of ‘no faults’. This algorithm works by stopping the ‘logical clock of a node (as if processor is in a wait state), as soon as it reaches its Ready Point, i.e., equal to an integral multiple of the clock synchronization interval (P).

3. Scope of works

This paper extends the work presented in [12] both on an algorithmic and on a theoretical level. We modify the algorithm proposed by Dhruba Basu and Sasikumar Punnekkat [12] and obtain better synchronization with less computation and less communication. In this paper, we present a simple clock synchronization algorithm, which reduces total network connections of the system mentioned in [12]. In this paper we synchronize the clocks with respect to the slowest among the correct clock. This newly proposed algorithm also ensures that there is always a continuity of logical clock values and there is no need to cause the logical clocks to be moved either forward or backward. We have developed a simulator in Java for the implementation of the proposed algorithm. We has also proposed two theorems based on the algorithm.

We consider a distributed system having n processors, which are referred to as *nodes.* Each node is assumed to have its own hardware (physical) clock. Also, each of these nodes has a logical clock, which is essentially a software counter.

We assume that each node has a *logical clock process,* which maintain a logical clock and synchronizes it with the other logical clocks periodically. All other processes executing on a particular node use this logical time obtained from the logical clock process of that node for scheduling, synchronization etc. In the context of this paper, we use the terms, node and *logical clock process* interchangeably.

As in most hard real-time systems, we assume that the tasks are periodic [2], where each task is characterized by its period p_i, release time r_i, and deadline d_i [6], [11]. All of these parameters are functions of the node's logical clock value.

We represent the real time (global time) with lower case letters such as t, and the logical time with upper case letters such as $C_i(t)$ which represent the value of the i^{th} logical clock (i=1,..,n) at real time t. We assume the ability to initially synchronize the clocks. The logical clocks are resynchronized periodically, and the *resynchronization interval* is represented by P.

Let $C_i^{k-1}(t)$ represent the logical clock value of the i^{th} node at real-time t, after the $(k-1)^{th}$ round of resynchronization has been performed. When $C_i^{k-1}(t)$ reaches the value KP, then the k^{th} round of resynchronization commences.

4. Assumptions reading clocks

We make the following assumptions, similar to those given in [10], regarding clocks:

a. We assume a constant Dmax ≥0, which represents the maximum permitted deviation of any correct logical clock from real time during a resynchronization interval. This can be stated as $|C_i^{k-1}(t)-t| \leq D_{max}$. This means that, the difference between logical clock values of the fastest and the slowest correct clocks can be at most $2D_{max}$ time units.
b. Rate of drift of correct or non-faulty physical clocks from real time is bounded by a known constant, p>0, which in turn determines D_{max}.
c. Clock drifts specified as ±x ppm are not considered but the actual clock drifts are measured apriori. This apriori measured clock drifts can never exceed the maximum drift i.e. ± x ppm.

The measured clock drifts do not change with time i.e. ignoring second order effects i.e. drift of drift is NIL.

5. Definitions

5.1. Ready Point

Similar to Dhruba Basu and Sasikumar Punnekkat[12], each logical clock process (except the slowest) stops logical clock incrementing process and put in 'wait' mode whenever its logical clock value reaches an integral (say, k^{th}) multiple of the resynchronization interval, P. These time points are referred to as Ready Points and they are {KP |¥K=1,2,…}. The real time corresponds to the Ready Point of the i^{th} process during the k^{th} round is denoted by $ready_i^k$.

5.2. Sequence of Synchronization

Different clocks may have different drift rates for which they drift apart from each other in course of time. To synchronize the system a pair wise synchronization has been achieved

between different drift rates of the clocks in the system. The sequence in which synchronization is taking place between different pairs is called the *Sequence of Synchronization*.

5.3. Reference Node

Different clocks may have different drift rates for which they drift apart from each other in course of time. To synchronize the system a pair wise synchronization has been achieved between different drift rates of the clocks in the system. For pair wise selection, each execution of the algorithm must be with respect to the one common node. That node is referred to as *Reference Node*.

.

6. Design methodology of our Algorithm

In this section, we present a simple clock synchronization algorithm, which reduces total network connections of the system mentioned in [12]. The fundamental concept of our proposed algorithm is to perform pair wise synchronizations. Slowest node forms a pair with every node. Every pair is synchronized with respect to the slowest among the correct clocks. This algorithm also ensures that there is always a continuity of logical clock values and there is no need to cause the logical clocks to be moved either forward or backward. This eliminates the problems associated with adjusting the clock backwards. This is an extremely important property for ensuring performance in most of the safety critical hard real-time systems. This also guarantees that no task release points are missed. The precedence constraints between tasks on different processors are preserved automatically.

It may be noted that, theoretically it is possible to have more than one clock with exactly the same drift rate, so that there is no unique clock, which is 'the slowest'. However, this does not have any impact on our algorithm, except that one should consider the term *slowest clock* as either referring to the group of slowest clocks or to any one among them.

The major argument against synchronizing with the slowest clock could be that, it necessitates logical clock of the processors with faster physical clocks to jump backwards, thus repeating the same logical time values again. This problem can be solved, by modifying processes, which manage the maintenance of logical clocks. We now present an algorithm, which can overcome the above problem under the assumption of 'no faults'.

The proposed algorithm works for *pair wise* nodes, by stopping the logical clock of the node i (as if processor is in a wait state), as soon as it reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P). This node i stay in wait state until the slowest clock reaches Ready Point of that node, i.e., i_{KP} value. In this algorithm, we first synchronize the fastest first and the slowest last. Next synchronization is taken place between the 2^{nd} fastest and the slowest last and so on. When the synchronization between the 2^{nd} slowest and the slowest last is completed, one cycle of synchronization is completed. In the next cycle of synchronization same algorithm will be repeated, i.e., in the next cycle, first synchronization is taken place between the fastest first and the slowest last. So the *maximum drift* between the fastest and the slowest correct clocks will be *bounded.* Here $C_i^{k-1}(t)$ represents the logical clock value of the i^{th} node (except the slowest) at real-time t, after the $(k-1)^{th}$ round of resynchronization has been performed and $C_n^{k-1}(t)$ represents the logical clock value of the slowest node at real-time t, after the $(k-1)^{th}$ round of resynchronization has been performed It may be noted that there exist a separate process for incrementing the logical clock.

The following algorithm needs to be executed by every clock process i (except the slowest), during k^{th} round of resynchronization.

6.1. Proposed Algorithm

begin

for i=1 to n-1 **do**

if $C_i^{k-1}(t)$= iKP **then** // k^{th} Ready Point has reached

Suspend logical clock incrementing process

// stop logical clock incrementing process and put in 'wait' mode

if $C_n^{k-1}(t) \neq$ iKP **then wait till** ($C_n^{k-1}(t)$= iKP)

//wait till the slowest node reaches the Ready Point of clock process i

$C_i^k(t)$= iKP and restart the logical clock incrementing process

// k^{th} round of resynchronization is completed for clock process i

endif

end for

end

This algorithm is simpler compared to the previous algorithm [12] and it is developed without any requirement of [READY K] and [SYNC K] messages [12]. The logical clock do not experience any jumps (either forward or backward). The logical time always changes in a non-descending manner.

The observations given below follow directly from our definitions and are noteworthy:

a. In the previous algorithm, we get a *fully connected* system. No. of network connection will be ${}^nC_2$, i.e. **n(n-1)/2** (where n is the no. of processors).
In the new algorithm, no. of network connection will be **(n-1)** (where n is the no. of processors).So in the new algorithm, we can reduce the total no. of network connection.
In the following diagram, we take n=5,so no. of network connection will be (n-1)=4, which is represented by the no. of edges of the following diagram.

b. If clock synchronization interval will be 10,000, then according to Dhruba Basu and Sasikumar Punnekkat[12] maximum drift would be 10 (between slowest and fastest).
In new algorithm, this maximum drift is ***bounded*** but has higher value, i.e.,90.

c. Since the slowest clock is the reference clock we assume its drift to be zero and drifts of all other clocks are measured with respect to this slowest clock.
Let these drifts be ($\rho_0, \rho_1, \rho_2, \rho_{3,..}, \rho_{n-1}$) all in ppm. (Where ρ_0=0.)

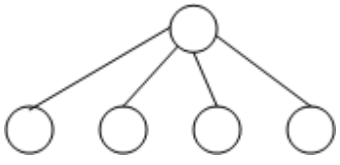

Figure 1. System with 5 processors having 4 network connections.

If the periodicity of pair wise synchronization is T_{pair}, then at the time of first synchronization (the fastest and the slowest) their clocks would be drifted by + ρ_{n-1} ppm. And the absolute maximum drift before the next synchronization between these pair would be $(\rho_{n-1} - \rho_0)$ (n-1) $T/10^6$.

7. Case study

In this section, we take 10 clocks (i.e., processors) of different drift rates. We want to synchronize these clocks according to our newly proposed algorithm.

Let us assume clock synchronization interval P to be 10,000, so that KP values (i.e., Ready Points) are 10,000, 20,000, 30,000, and so on.

In a 10 processors system, let these drifts be +500, +400, +300, +200, +100, -100, -200, -300, -400, -500 (all in ppm) shown in Table 1.

Table 1. 10 processors system with different drift values.

Node i	$C_i^k(t)$ when real time t=10,000
A	10,000+5
B	10,000+4
C	10,000+3
D	10,000+2
E	10,000+1
F	10,000-1
G	10,000-2
H	10,000-3
I	10,000-4
J	10,000-5

According to our proposed algorithm, slowest node forms a pair with every node. Every pair is synchronized with respect to the slowest among the correct clocks. Here slowest node is J and the fastest node is A. Pair is formed between the nodes A and J, B and J, C and J, and D and J, E and J, F and J, G and J, H and J, I and J. In this algorithm, we first synchronize the fastest first and the slowest last, i.e., between the nodes A and J .Next synchronization is taken place between the 2nd fastest and the slowest, i.e., between the nodes B and J and so on. One round is completed when the last pair I and J is synchronized.

1st synchronization is taken place between the nodes A and J at 10,000, by stopping the clock of node A when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =10,000).The node A stays at 10,000 until the node J reaches 10,000. So, A has to wait 10 time unit.

2nd synchronization is taken place between the nodes B and J at 20,000, by stopping the clock of node B when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =20,000).The node B stays at 20,000 until node J reaches 20,000. So, B has to wait 18 time unit.

3rd synchronization is taken place between the nodes C and J at 30,000, by stopping the clock of node C when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral

multiple of the clock synchronization interval (P) (in this case i_{KP} =30,000).The node C stays at 20,000 until node J reaches 30,000. So, C has to wait 24 time unit.

4th synchronization is taken place between the nodes D and J at 40,000, by stopping the clock of node D when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =40,000).The node D stays at 40,000 until node J reaches 40,000. So, D has to wait 28 time unit.

5th synchronization is taken place between the nodes E and J at 50,000, by stopping the clock of node E when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =50,000).The node E stays at 50,000 until node J reaches 50,000. So, E has to wait 30 time unit.

6th synchronization is taken place between the nodes F and J at 60,000, by stopping the clock of node F when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =60,000).The node F stays at 60,000 until node J reaches 60,000. So, F has to wait 24 time unit.

7th synchronization is taken place between the nodes G and J at 70,000, by stopping the clock of node G when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =70,000).The node G stays at 70,000 until node J reaches 70,000. So, G has to wait 21 time unit.

8th synchronization is taken place between the nodes H and J at 80,000, by stopping the clock of node H when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =80,000).The node H stays at 80,000 until node J reaches 80,000. So, H has to wait 16 time unit.

9th synchronization is taken place between the nodes I and J at 90,000, by stopping the clock of node I when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =90,000).The node I stays at 90,000 until node J reaches 90,000. So, I has to wait 9 time unit.

To prove the maximum drift (i.e., the drift between the slowest and the fastest) is bounded after the completion of each round, we found out the distance between the fastest node (A) and the slowest node (J) when the pair (I,J) is synchronized.

a. Pair (H,J) is synchronized 10,000 time unit before the synchronization of pair (I,J). After 10,000 drift between the H and J is 2*1=2.
b. Pair (G,J) is synchronized 20,000 time unit before the synchronization of pair (I,J). After 20,000 drift between the G and J is 3*2=6.
c. Pair (F,J) is synchronized 30,000 time unit before the synchronization of pair (I,J). After 30,000 drift between the F and J is 4*3=12.
d. Pair (E,J) is synchronized 40,000 time unit before the synchronization of pair (I,J). After 40,000 drift between the E and J is 6*4=24.
e. Pair (D,J) is synchronized 50,000 time unit before the synchronization of pair (I,J). After 50,000 drift between the D and J is 7*5=35.
f. Pair (C,J) is synchronized 60,000 time unit before the synchronization of pair (I,J). After 60,000 drift between the C and J is 8*6=48.
g. Pair (B,J) is synchronized 70,000 time unit before the synchronization of pair (I,J). After 70,000 drift between the B and J is 9*7=63.
h. Pair (A,J) is synchronized 80,000 time unit before the synchronization of pair (I,J). After 80,000 drift between the A and J is 8*10=80.

Table 2 shows drift of other clocks when the pair (I,J) is synchronized at 90,000 and Fig. 2 shows the time graph at that time.

Table 2. Drift of other clocks when the pair (I,J) is synchronized at 90,000.

Clock	Drift
H	2
G	6
F	12
E	24
D	35
C	48
B	63
A	80

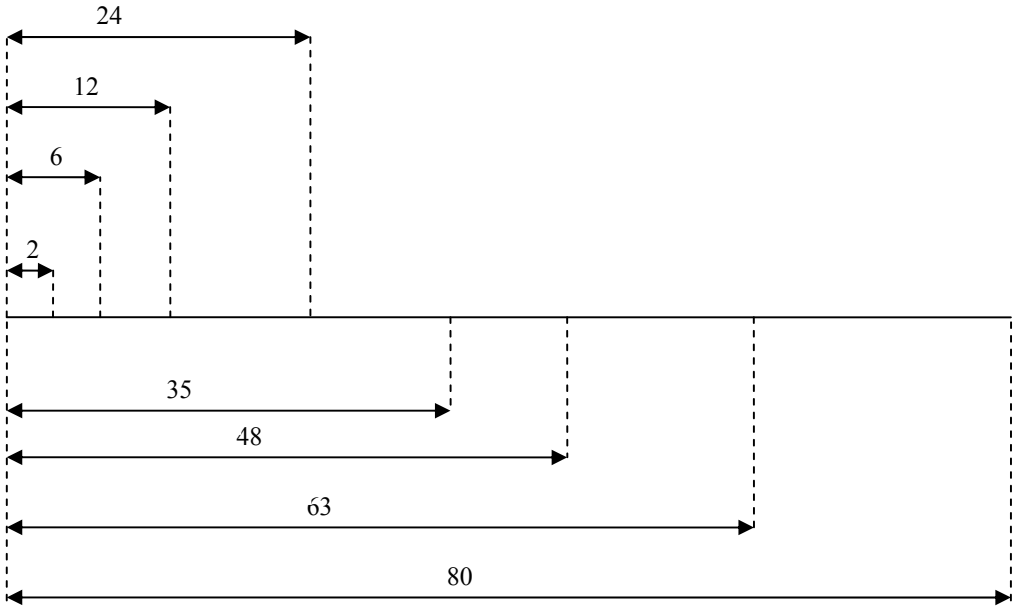

Figure 2. Time graph when (I,J) was synchronized.

8. Theorems

8.1. Theorem-1

If the pair wise synchronization is carried out by selecting pairs, such that they have the minimum drift between them and the pairs are exclusive for one complete round of synchronization then the maximum drift is unbounded.

PROOF: Let us assume clock synchronization interval P to be 10,000 , so that KP values (i.e., Ready Points) are 10,000, 20,000, 30,000, and so on.

In a 8 processors system, let these drifts be +400,+300,+200,+100,-100, -200,-300.-400 (all in ppm) shown in Table 3.

Table 3. 8 processors system with different drift values.

Node i	$C_i^k(t)$ when real time t=10,000
A	10,000+4
B	10,000+3
C	10,000+2
D	10,000+1
E	10,000-1
F	10,000-2
G	10,000-3
H	10,000-4

Here we form pairs exclusively and then we implement our newly proposed algorithm on those pairs.

Here pair is formed between nodes A and B, C and D, E and F, G and H as they have minimum drift between them. Synchronization is carried out in each pair just like the following way.

When the node having the faster clock in a pair reaches the Ready Point = KP, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case synchronization interval is 10,000), it stays in wait state until the node having the slower clock reaches Ready Point =KP. In this process, first synchronization is carried out between the fastest first and the fastest second, i.e., between the nodes A and B .Next synchronization is taken place between the 3rd fastest and the 4th fastest i.e., between the nodes C and D and so on.

One round is completed when the last pair G and H is synchronized.

As the pairs are exclusive and no synchronization is carried out between the individual pairs, so the maximum drift (i.e., the drift between the slowest and the fastest) will be increased after each round of synchronization is completed.

8.2. Theorem 2

Table 4. 10 Processor System with different drift values

Node i	$C_i^k(t)$ when real time t=10,000
A	10,000+5
B	10,000+4
C	10,000+3
D	10,000+2
E	10,000+1
F	10,000-1
G	10,000-2
H	10,000-3

For a given periodicity of pair wise synchronization the maximum drift (i.e., the drift between the slowest and the fastest) would be bounded in all cases of inclusive selection of the sequence of pair wise synchronization. This bound would be the smallest when the sequence of synchronization is the fastest first to the slowest last and the largest when the sequence of synchronization is the slowest first to the fastest last.

PROOF: Let us assume clock synchronization interval P to be 10,000, so that KP values (i.e., Ready Points) are 10,000, 20,000, 30,000, and so on.

In a 10 processors system, let these drifts be +500, +400, +300, +200, +100,-100, -200,-300, -400, -500 (all in ppm) shown in Table 4.

Synchronization is achieved with respect to the slowest among the correct clock. The slowest node forms a pair with every node. Every pair is synchronized with respect to the slowest among the correct clocks. Here slowest node is J and the fastest node is A. Pair is formed between the nodes A and J, B and J, C and J, and D and J, E and J, F and J, G and J, H and J, I and J. Let the first pair is formed between the slowest first to the fastest last nodes, i.e., between nodes I and J. Next synchronization is taken place between the 3rd slowest and the slowest, i.e., between the nodes H and J and so on. One round is completed when the last pair A and J is synchronized. Synchronization is carried out in each pair just like the following way.

First synchronization is taken place between the nodes I and J at 10,000, by stopping the clock of node I when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =10,000).The node I stays at 10,000 until the node J reaches 10,000. So, I has to wait 1 time unit.

2nd synchronization is taken place between the nodes H and J at 20,000, by stopping the clock of node H when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =20,000).The node H stays at 20,000 until node J reaches 20,000. So, H has to wait 4 time unit.

3rd synchronization is taken place between the nodes G and J at 30,000, by stopping the clock of node G when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =30,000).The node G stays at 30,000 until node J reaches 30,000. So, G has to wait 9 time unit.

4th synchronization is taken place between the nodes F and J at 40,000, by stopping the clock of node F when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =40,000).The node F stays at 40,000 until node J reaches 40,000. So, F has to wait 16 time unit.

5th synchronization is taken place between the nodes E and J at 50,000, by stopping the clock of node E when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =50,000).The node E stays at 50,000 until node J reaches 50,000. So, E has to wait 30 time unit.

6th synchronization is taken place between the nodes D and J at 60,000, by stopping the clock of node D when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =60,000).The node D stays at 60,000 until node J reaches 60,000. So, D has to wait 42 time unit.

7th synchronization is taken place between the nodes C and J at 70,000, by stopping the clock of node C when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =70,000).The node C stays at 70,000 until node J reaches 70,000. So, C has to wait 56 time unit.

8th synchronization is taken place between the nodes B and J at 80,000, by stopping the clock of node B when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral

multiple of the clock synchronization interval (P) (in this case i_{KP} =80,000).The node B stays at 80,000 until node J reaches 80,000. So, B has to wait 72 time unit.

9^{th} synchronization is taken place between the nodes A and J at 90,000, by stopping the clock of node A when its logical clock reaches its Ready Point= i_{KP}, i.e., equal to an integral multiple of the clock synchronization interval (P) (in this case i_{KP} =90,000).The node A stays at 90,000 until node J reaches 90,000. So, A has to wait 90 time unit.

One cycle is completed when the last pair A and J is synchronized.

After the completion of one cycle the maximum drift (i.e., the drift between the slowest and the fastest) is 90.

But if we first synchronize the fastest first and the slowest last, i.e., between the nodes A and J and next synchronization is taken place between the 2^{nd} fastest and the slowest, i.e., between the nodes B and J and so on. After the completion of one cycle the maximum drift (i.e., the drift between the slowest and the fastest) would be 80, which is already shown in the algorithm design section.

9. Result

We have developed a simulator in Java for the implementation of the proposed algorithm. The outcome of the simulator output is given below:

```
Enter no. of Clocks:
3

Enter 0-th Clock's drift value:
100

Enter 1-th Clock's drift value:
500

Enter 2-th Clock's drift value:
200
```

Clock synchronization interval (P): 10000

```
Clock Value having drift ----- 500: 10000
Starts waiting -------

Next Synchronization at: 30000
Synchronization done at ---------- 10000
Synchronization done between clocks having drift 500 and 100
Clock Value having drift ----- 200: 20000
Starts waiting -------
Next Synchronization at  : 40000
Synchronization done at ---------- 20000
Synchronization done between clocks having drift 200 and 100
```

One Cycle Completed

```
Clock Value of the Fastest Clock: 28160
Clock Value of the slowest Clock: 20000
Difference between the two clocks:8160
```

Clock Value having drift ----- 500 : 30000
Starts waiting -------
Next Synchronization at : 50000
Synchronization done at ---------- 30000
Synchronization done between clocks having drift 500 and 100
Clock Value having drift ----- 200 : 40000
Starts waiting -------
Next Synchronization at : 60000
Synchronization done at ---------- 40000
Synchronization done between clocks having drift 200 and 100
One Cycle Completed

Clock Value of the Fastest Clock: 48210
Clock Value of the slowest Clock: 40000
Difference between the two clocks:8210
Clock Value having drift ----- 500 : 50000
Starts waiting -------
Next Synchronization at : 70000
Synchronization done at ---------- 50000
Synchronization done between clocks having drift 500 and 100
Clock Value having drift ----- 200 : 60000
Starts waiting -------
Next Synchronization at : 80000
Synchronization done at ---------- 60000
Synchronization done between clocks having drift 200 and 100
One Cycle Completed

Clock Value of the Fastest Clock: 68210
Clock Value of the slowest Clock: 60000
Difference between the two clocks:8210
Clock Value having drift ----- 500 : 70000
Starts waiting -------
Next Synchronization at : 90000
Synchronization done at ---------- 70000

Synchronization done between clocks having drift 500 and 100
Clock Value having drift ----- 200 : 80000
Starts waiting -------
Next Synchronization at : 100000
Synchronization done at ---------- 80000
Synchronization done between clocks having drift 200 and 100
One Cycle Completed

Clock Value of the Fastest Clock: 88210
Clock Value of the slowest Clock: 80000
Difference between the two clocks:8210

10. Analysis

Table 5. Drift of the fastest clock (500 ppm) with respect to different time interval.

Real Time	Drift
1000	0.5
2000	1
3000	1.5
4000	2
5000	2.5
6000	3
7000	3.5
8000	4
9000	4.5
10,000	5

Table 6. drift of the slowest clock (100 ppm) with respect to different time interval.

Real Time	Drift
1000	0.1
2000	0.2
3000	0.3
4000	0.4
5000	0.5
6000	0.6
7000	0.7
8000	0.8
9000	0.9
10,000	1

To represent graphically proposed Clock synchronization algorithm, we take 3 clocks having drift 500 ppm, 200 ppm, 100 ppm.

We choose resynchronization interval (P) = 10,000.

According to our proposed algorithm, clock having drift 500 ppm is first synchronized with the clock having drift 100 ppm at 10,000.

Next synchronization is taken place between the clock having drift 200 ppm and the clock having drift 100 ppm at 20,000.

Figure 3.shows the drift of the fastest (500 ppm) and the slowest clock (100 ppm) before the synchronization is taking place between them.

At 10,000 drift of the fastest clock (500 ppm) is 5.

At 10,000 drift of the slowest clock (100 ppm) is 1.

At 10,000, synchronization is taking place between these two clocks.

So, drift between them before the synchronization at 10,000 will be (5-1) =4.

These drift (i.e., 4) will be minimized, if we reduce the synchronization interval (P).

Table 7. Drift of the 2nd fastest clock (200 ppm) with respect to different time interval

Real Time	Drift
2000	0.4
4000	0.8
6000	1.2
8000	1.6
10,000	2
12,000	2.4
14,000	2.8
16,000	3.2
18,000	3.6
20,000	4

The table 7 shows drift of the 2nd fastest clock (200 ppm) with respect to different time interval.

Figure 3. shows the drift of the 2nd fastest (200 ppm) and the slowest clock (100 ppm) before the synchronization is taking place between them.

At 20,000 they are synchronized.

At 20,000 drift of the 2nd fastest clock (200 ppm) is 4.

At 20,000 drift of the slowest clock (100 ppm) is 2.

So, drift between them before the synchronization at 20,000 will be (4-2) =2.

These drift (i.e., 2) will be minimized, if we reduce the synchronization interval (P).

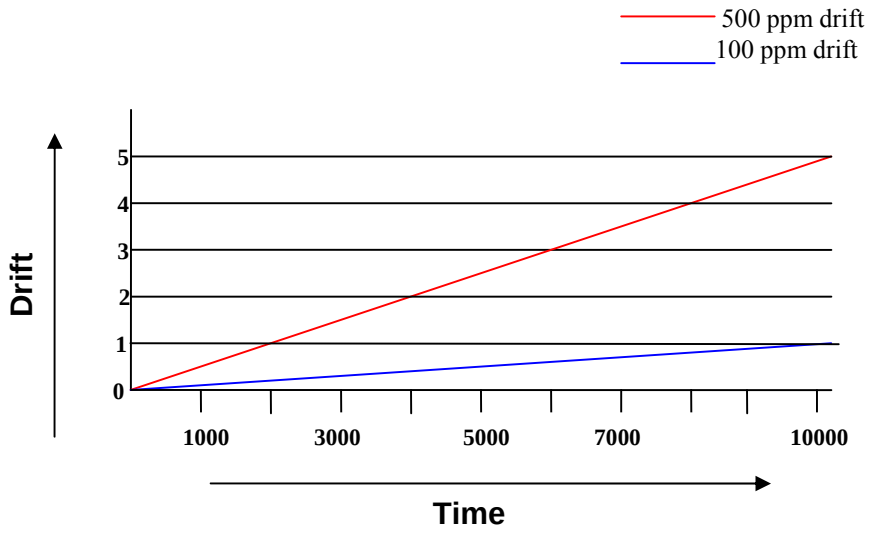

Figure 3. Drift of the fastest (500 ppm) and the slowest clock (100 ppm) before synchronization.

200 ppm drift
100 ppm drift

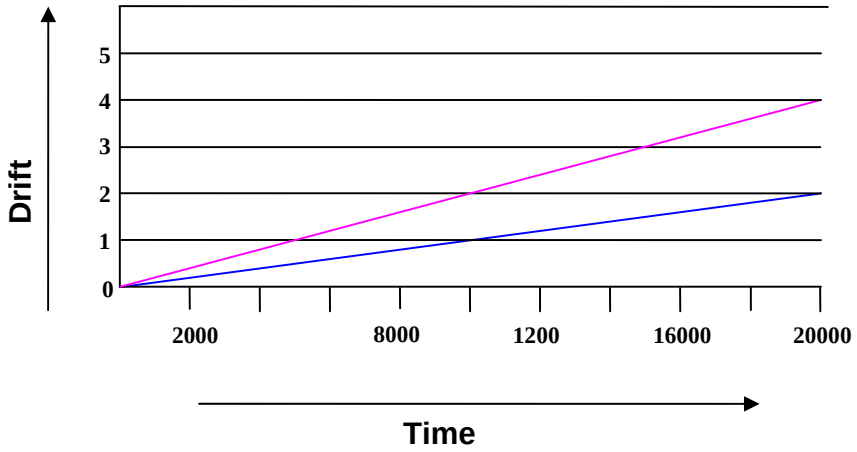

Figure 4. Drift of the 2nd fastest (200 ppm) and the slowest clock (100 ppm) before synchronization.

Figure 5, shows the clock synchronization between 3 clocks (having drift 500 ppm, 200 ppm, 100 ppm) according to our proposed algorithm.

The following table 8 shows drift of the fastest clock (500 ppm) and the slowest clock (100 ppm) after the first round of synchronization (at 10,000).

Table 8. Drift of the fastest clock and the slowest clock at 10,000 (At 30,000 these two clocks are again synchronized).

Fastest clock (500 ppm)		Slowest clock (100 ppm)	
Real Time	Drift	Real Time	Drift
10,000	1	10,000	1
15,000	3.5	15,000	1.5
20,000	6	20,000	2
25,000	8.5	25,000	2.5
30,000	11	30,000	3

11. Proof of concept demonstration

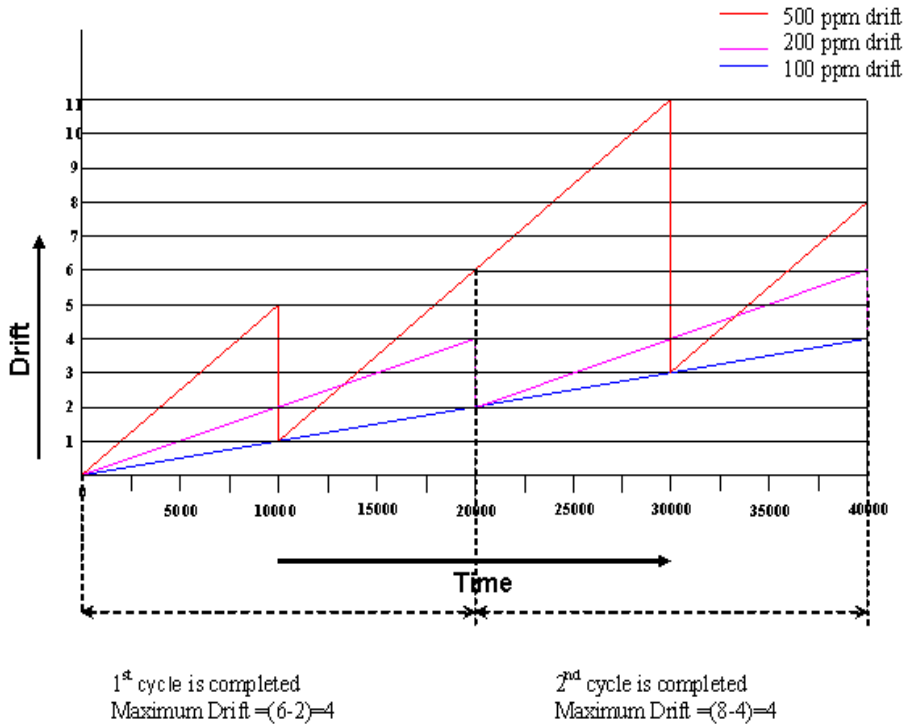

Figure 5. Graphical Representation of Clock Synchronization Including 3 clocks.

Table 9 shows drift of the fastest clock (500 ppm) and the slowest clock (100 ppm) after the second round of synchronization (at 30,000) between them.

Table 9. Drift of the fastest clock and slowest clock at 30,000.

Fastest clock (500 ppm)		Slowest clock (100 ppm)	
Real Time	Drift	Real Time	Drift
30,000	3	30,000	3
35,000	5.5	35,000	3.5
40,000	8	40,000	4

At 20,000 synchronization is taken place between the 2nd fastest clock (200 ppm) and the slowest clock (100 ppm).1st cycle of synchronization is completed at 20,000. At that time, maximum drift (i.e., the drift between the slowest and the fastest) will be 4.

The following table 10 shows drift of the 2nd fastest clock (200 ppm) and the slowest clock (100 ppm) after the synchronization at 20,000.

Table 10. Drift of the 2nd fastest clock and the slowest clock at 20,000.

2nd fastest clock(200 ppm)		Slowest clock (100 ppm)	
Real Time	Drift	Real Time	Drift
20,000	2	20,000	2
25,000	3	25,000	2.5
30,000	4	30,000	3
35,000	5	35,000	3.5
40,000	6	40,000	4

At 40,000 these two clocks are again synchronized.

2nd cycle of synchronization is completed at 40,000. At that time maximum drift will be (i.e., the drift between the slowest and the fastest) 4. So, the maximum drift will be bounded.

In theorem section, we provide simulation output. In simulation output, maximum drift (i.e., the drift between the slowest and the fastest) is computed after the completion of each cycle. Maximum drift is almost equal at the end of each cycle. So, we can conclude that the maximum drift is bounded from the simulation output.

12. Conclusion

In this paper, we have highlighted a practical problem, which can arise, in the case of the well-known clock synchronization algorithms by Dhruba Basu and Sasikumar Punnekkat [12]. According to Dhruba Basu and Sasikumar Punnekkat[12], in a n processor system, we get a *fully connected* system. No. of network connection will be ${}^{n}C_{2}$, i.e. n(n-1)/2 (where n is the no. of processors). Complexity of the algorithm proposed by them is $0(n^2)$. We have proposed a new algorithm for clock synchronization, which overcome this problem by reducing total network connections of the system. In our proposed algorithm, no. of network connection will be (n-1) (where n is the no. of processors). Complexity of the newly proposed algorithm will be 0(n). The fundamental concept of our proposed algorithm is to perform pair wise synchronizations. Slowest node forms a pair with every node. Every pair is synchronized with respect to the slowest among the correct clocks. A minor criticism against our algorithm could be that, since faster clocks are made to wait the overall utilization of the system decreases. Clearly there is a tradeoff between predictability and optimality of utilization/accuracy, but in the class of system we are dealing with, the thrust is on predictable solutions rather than optimal utilization, thus favoring our new algorithm. Presently, we are working on algorithm that tries to reduce the maximum drift (i.e., the drift between the slowest and the fastest) and will give better synchronization.

Acknowledgement

This work was supported by the Security Engineering Research Center, granted by the Korea Ministry of Knowledge Economy.

References

[1] Thomas Anderson and John Knight, "A Framework for Software Fault Tolerance in Real-Time Systems," IEEE Transactions on Software Engineering, Vol. SE-9, No.3, 355-364, 1983.

[2] Flaviu Cristian, "Probabilistic Clock Synchronization," Distributed Computing, No.3, pp 146-158, Springer-Verlag, 1989.

[3] H. Kopetz and G. Grünsteidl, "TTP- A Protocol for Fault Tolerant Real-Time Systems," IEEE Computer, January 1994.

[4] H. Kopetz and W. Oshsenreiter, "Clock Synchronization in Real-Time Systems," IEEE Transactions on Computers ,Vol. 36, No.8 pp 933-940, August 1987.

[5] Leslie Lamport and P.M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," Journal of the Association for Computing Machinery, Vol. 32, No.1, pp 52-78, January 1985.

[6] C.L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming 1973.

[7] Brian Randell, "System Structure for Software Fault Tolerance," IEEE Transactions on Software Engineering, Vol. SE-1, No.2, pp 220-232, 1975.

[8] K.G. Shin and P. Ramamathan, "Transmission Delay in Hardware Clock Synchronization," IEEE Transactions on Computers ,Vol. 37, No.11, pp 1465-1467, November 1988.

[9] Anupam Sinha, Pradip K. Das, and Dhruba Basu, "Implementation and Timing Analysis of Clock Synchronization based on a transputer-based replicated system ," Information and Software Technology, Vol. 40, No.5-6, pp 291-301, July 1998.

[10] T.K. Srikanth and Sam Toueg, "Optimal Clock Synchronization," Journal of the Association for Computing Machinery, Vol. 34, No.3 pp 626-645, July 1987.

[11] Jia Xu and David L. Parnas, "On Satisfying Timing Constraints in Hard Real-Time Systems," IEEE Transactions on Software Engineering, Vol. 19, No.1, pp 70-84, January 1993.

[12] Dhruba Basu and Sasikumar Punnekkat, "Clock Synchronization Algorithms and Scheduling Issues," at 5th International Workshop Distributed Computing, Kolkata, India, Dec 27-30, 2003. Proceeding LNCS 2918, Springer 2003.

[13] Barbara Simons, "an Overview of Clock Synchronization," IBM Research Jennifer Lundelius Welch, GTE Laboratories Incorporated Nancy Lynch MIT, August 23, 1988.