# Self-Adaptive Clock Synchronization Based on Clock Precision Difference

**[1]Ying Zhao, [2]Wanlei Zhou[*], [2]Elicia. J. Lanham, [2]Shui Yu, [2]Mingjun Lan**

[1]International Network Centre
Beijing University of Chemical Technology
15 Beisanhuan East Road, Chaoyang District, Beijing, 100029, P. R. China
zhaoy@mail.buct.edu.cn

[2]School of Information Technology
Deakin University
221 Burwood HWY, Burwood, VIC 3125, Australia
{ wanlei, lanham, syu, mlan }@deakin.edu.au
* Correspondence author

## Abstract

This paper presents an innovative strategy to synchronize all virtual clocks in asynchronous Internet environments. Our model is based on the architecture of one reference clock and many slave clocks communicating with each other over the Internet. The paper makes three major contributions to this research area. Firstly, one-way information transmission is applied to reduce traffic overhead on the Internet for the purpose of clock synchronization. Secondly, the slave nodes use local virtual time and the arrival timestamp, from the reference node, to create linear mathematical trend models and to retrieve the clock precision differences between reference clock and slave clocks. Finally, a fault-tolerant and self-adaptive model executed by each slave node based on the above linear trend model is created in order to ensure that the virtual clock is running normally, even when the link between the reference node and this slave node has crashed. We also present detailed simulations of this strategy and mathematical analysis on real Internet environments.

*Keywords*: Clock Synchronization, Precision Difference, Asynchronous Environments, Self-Adaptive.

## 1 Introduction

Recently, network-connected clusters of PCs or workstations have evolved quickly and have become a widespread parallel computing platform because of their inexpensive prices. Unfortunately, these systems seldom have a global clock, even though it is invaluable for measuring connection latencies, evaluating performance and coordinating distributed applications.



The problem of Clock Synchronization is a classical issue in distributed system, where no two clocks are synchronized perfectly (Butler 1988, Cristian 1996, Sinha 1996). Generally, we consider two clocks are synchronized at a particular instance of time if the time difference of the two clocks is less than a specified constant $\delta$. Examples of distributed algorithms benefiting from clock synchronization can be found in (Liskov 1993).

Because there are different clock precisions of nodes, the issue of clock synchronization occurs. If an instantaneous time is applied to a synchronization process, the message delay must be considered seriously. However, questions such as: how to calculate this delay, and how to predict every delay in the system, are very complicated and difficult to solve. Some papers use round trip delay to estimate the transmission delay, but the irregular feature of networks frustrates this solution. In this paper, we first use one way timed transmission to estimate the precision difference, and then use this difference to tune the clock in a local node. This is a continuous process, not only does it compute the precision difference, but also provides a trend of data collecting.

This one-way transmission policy not only depends on the application of slave time or local time, but also tolerates the failures of communications. So we call this model a timed model. The detailed definition of timed asynchronous distributed system model is described in paper (Cristian, and Fetzer 1999). In our model, all transmission information-timestamps are timed, and the unreliable datagram services are also applied. Each slave node uses the local time and arrival timed timestamp to calculate its precision difference with master clock.

We hope to find some relation amongst these clocks, especially a relation between the salve clocks and master clock. If there is no delay of information transmission, then there is no problem, because the master clock can distribute its time value to slave clocks immediately. This is an ideal case, but it is not possible to achieve this situation. This means that we must consider the

transmission delay in asynchronous network environments.

Some papers (Mock, Frings, Nett, and Trikaliotis 2000, Lamport, and Melliar 1985, Srikanth, and Toueg 1987) provide a strategy of continuous synchronization to avoid these drawbacks by distributing the correction over the next synchronization period and applying it continuously. Unfortunately, these methods do not attempt to benefit from continuous clock adjustment to achieve a higher level of precision, to cope with variances in message delivery or even with message losses.

The goals of this paper are to: 1) propose a one way transmission method to measure the effect of network delay, 2) create a timed linear mathematical model to get the precision difference of slave clocks with a master clock and give a detailed mathematical analysis, 3) design a real simulation on Internet environments and collect important data results, graphics and tables, 4) provide a self adaptive model of slave nodes.

In the implementation of our model, we use a virtual clock concept rather than a physical clock. The advantages of a virtual clock based approach, as opposed to the system clock based approach, are summarized in paper (Ciuffoletti 1999). This makes the tuning of slave clocks more flexible and easy. In practice, it is very easy to keep a mapping relation between a virtual clock and a physical clock.

## 2 The System Model

Now we will use the external clock synchronization (Azevedo, and Blough 1998) case as an example to explain how to use the difference of clock precision to synchronize all local clocks. We call this external timeserver the master node. The nodes that need to be synchronized are called slave nodes.

The master node use multicast service to send timestamp messages to local nodes periodically. The local nodes will receive these timestamp messages, and can calculate the precision difference between the clock of the master node and the clock of the local node based on the received sampling data. As more timestamps are received, we can use statistical analysis to get a linear trend model to estimate the precision difference between two clocks.

For simplification, we assume the master node multicasts a timestamp per $T_{period}$ seconds. The local nodes or slave nodes will receive these timestamps periodically.

Because the precision difference between two nodes is applied to synchronize their clocks, these two clocks should be set at the same time before the commencement of the systems in order to obtain the best results possible. The system should therefore satisfy the follow constraint assumption.

Assumption 1: When a system begins to run, every pair consisting of a master clock and slave clocks (*m* and *s*) meets:

$$\left|LC_m(t_0) - LC_s(t_0)\right| \le \delta \qquad (1)$$

In this assumption, let $t_0$ denote the initial time of clock synchronization. Let  denote the time difference of two clocks *m* and *s*., and let $LC_x(t_0)$ denote the local time of *x* node at real time $t_0$.

Many existing methods (Azevedo, and Blough 1998, Bouzelat, and Mammeri 1997, Cristian 1991, Lamport, and Melliar 1985, Mock, Frings, Nett, and Trikaliotis 2000, Srikanth, and Toueg 1987) satisfy Assumption 1. Because the key topic of this paper is discussing how to use the precision difference to tune the clock, we assume that current systems meet Assumption 1.

Due to the imprecision of the oscillator, temperature changes, and aging, a hardware clock drifts apart from real-time. Intuitively, the drift rate of a hardware clock indicates how many microseconds a hardware clock drift from real-time per second. For example, if a clock drift rate is about 1 s/second, this means that a clock increases its value by 1second+1 s. To simplify the calculation, we redefine the clock drift rate as, how many seconds drift difference per second. For a 1 s/second drift rate, the new expression is $1*10^{-6}$.

Now we assume the existence of a constant maximum drift rate  <<1 that bounds the absolute value of the drift rate of a correct clock (Cristian, and Fetzer 1999). In addition, we assume the drift rate of a clock is a constant in order to simplify our analysis.

Assumption 2: Each slave clock in our system has a fixed drift rate and is bounded by a maximum drift rate  (absolute value). The relation between real-time *t* and clock time *T* can be expressed as follows:

$$-\rho \le \frac{dT}{dt} - 1 \le +\rho \qquad (2)$$

The basic architecture of this system is described in Figure 1.

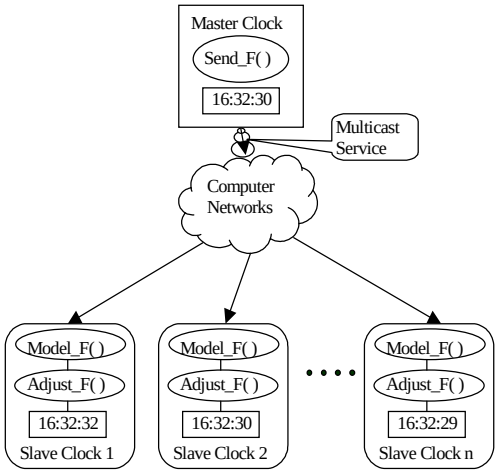


**Figure 1: The architecture of clock synchronization**

>From Figure 1, we know that there are n salve nodes that will synchronize their virtual clock time value with the master clock. Based on assumption 1, the initial state of the system should ensure that the absolute value of difference between the master clock time and each slave clock time is less than or equal to  . We assume that the

master mutilcasts its timestamp by function Send_F( ) per $T_{period}$ time. This is a one-way transmission of information, which avoids additional network traffics for clock synchronization. Each node on the slave side receives the timestamps from the master node, creates a linear trend model, and calculates the clock precision difference between its node and the master node. The slave then adjusts its clock by the demand of system. These functions are implemented by the functions: Model_F( ) and Adjust_F( ).

## 3 Continuous Communication

All timestamps are transferred through networks and finally they arrive at the destination-slave nodes. Because of the intermittent asynchronous feature of networks, it is difficult to measure the accurate transmission delay. Figure 2 demonstrates this situation.

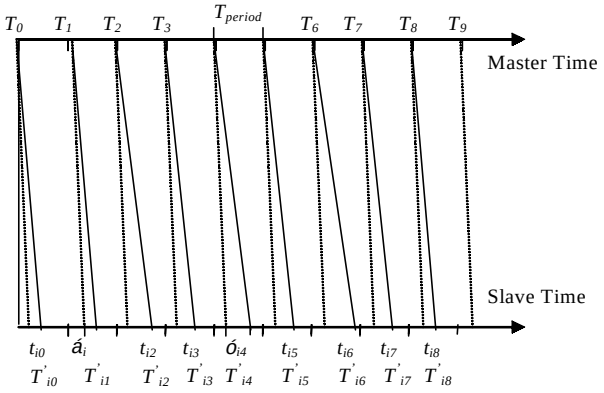


**Figure 2: The delay of message transmission**

Where:

$T_j$: Denotes the $j$th sending time value of the master clock. Usually we think of it as a standard time. This time value is contained in a timestamp, which is sent to slave nodes.

$t_{ij}$: Denotes the local time of slave node $i$ when the stamp containing $T_j$ from the master node arrives at.

$T'_{ij}$: Represents the standard time when the stamp containing $T_j$ from the master node arrives at the slave node $i$.

$T_{period}$: Represents the fixed synchronization period.

$_i$: Denotes the smallest transmission delay based on standard time.

$_{ij}$: Denotes the drift time difference from the smallest transmission delay based on standard time.

>From Figure 2, we can derive the following formula:

$$T'_{ij} = T_j + \alpha_i + \sigma_{ij} \qquad (3)$$

In this formula, $_i$ and $_{ij}$ denote the transmission delay of the timestamp from the master node. If we assume that no transmission delay happens, then we know:

$$\alpha_i = 0 \text{ and } \sigma_{ij} = 0.$$

Now, synchronising all the slave clocks using instantaneous clock time of the master clock can be done easily.

$$t_{ij} = T'_{ij} = T_j$$

This is the ideal case, however we cannot achieve it. In practice, both $_i$ and $_{ij}$ are not equal to zero. Normally, $_i$ is a constant, which we can calculate by the distance between the master node and the slave node and the transmission speed. But the value of $_{ij}$ is an unstable parameter which changes according to the network traffic. If network has a light load, the value of $_{ij}$ may be very small. However, if the network has a heavy load, then the value of $_{ij}$ will become very large. In this situation, the corresponding timestamp may be lost.

Now, we begin to consider the issue of precision difference between the master clock and the slave clock. If $_{ij}=0$ or $_{ij}=constant$, we can easily get the result of precision deviation. For instances,

If $t_{i8}-t_{i0}=T_8-T_0$, the master clock and the slave clock have equal precision.

If $t_{i8}-t_{i0}>T_8-T_0$, we conclude that the slave clock runs faster that the master clock.

If $t_{i8}-t_{i0}<T_8-T_0$, we conclude that the salve clock runs slower than the mater clock.

Because the timestamp arrives at the destination through the network, there is no reason to ensure $_{ij}=0$ or $_{ij}=constant$, as the value of $_{ij}$ is unstable and uncertain. It is difficult to only use a single value of $_{ij}$ to calculate the precision difference. In our research work, a master clock sends its timestamp per $T_{period}$ period. On the slave side, they expect that they can receive the timestamp from the master node per $T_{period}$ period.

If the precision of the master clock is same as the precision of the slave clock, we can defer the transmission delay of this timed-timestamp as follows:

$$\alpha_i + \sigma_{ij} = T'_{in} - T_n = t_{in} - T_n \qquad (4)$$

But if the precision of the master clock is not equal to the precision of the slave clock, then the value of $t_{in}$ is also not equal to the value of $T'_{in}$.

On the surface, the occurrence of this transmission delay is random, so it is difficult to measure it. In fact, if the previous timestamp arrives at destination late, then the next timestamp will have a tendency to come early. We can use the coming timestamps to get its trend. For example, we assume the timestamp containing $T_2$ arrives at the slave node late. Its delay is $_i+_{i2}$. Now we expect that the next timestamp containing $T_3$ can arrive at this node earlier.

From the perspective of the slave, they have their own clocks, and know the sending interval of $T_{period}$. If a slave clock has same precision as the master clock, the arrival interval calculated by the slave node will waver near the interval of $T_{period}$. If there is a long interval this time, a short interval shall be expected next time. Even though, in some cases, this next interval may be longer than

previous one. This also means that the next or future interval shall be expected shorter than the interval we are currently expecting. In the long run, its trend line should be a horizontal line.

If a slave clock runs slower than the master clock, the interval trend line measured by the local clock should be a diagonal line below the horizontal line. If a slave clock runs faster than the master clock, this interval trend line should be a diagonal line above the horizontal line. These situations are illustrated in Figure 3.

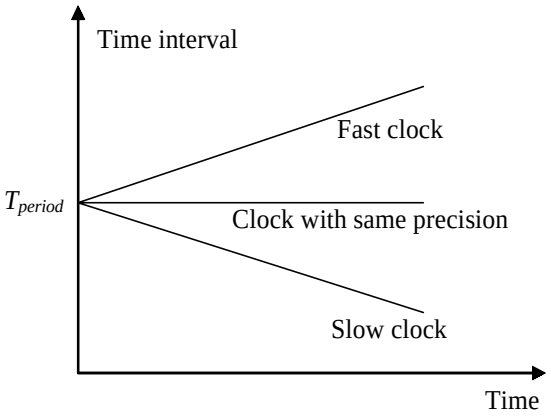


**Figure 3: The clock running trend analysis**

Actually, if slave clocks meet assumption 2, then the time interval trends from the slave clocks should be a straight line shown in the Figure 3. If a slave clock runs slower than the master clock or standard clock, the interval measured by slave nodes will be smaller and smaller as time flies. If a slave clock runs faster than the master clock, the interval measured by slave nodes will get larger as time continues.

As Figure 3 shows, these trend lines reflect the drift rate of slave clocks directly. Based on the knowledge of the line equation, the slope of a line equation presents the direction of a trend line. For the application of clock synchronization, this slope reflects the deviation of the slave clock from the master clock. We usually call this deviation the drift rate.

Because of the uncertainty of transmission over networks, we can not use an instantaneous value to evaluate this drift rate. A trend line can reflect a continuous changeable status of variables, and its slope presents speed of deviation. In Figure 3, if this slave clock has same precision as the master clock, the trend line of the slave node calculated should be a horizontal straight line, where its slope is 0. This means that its drift rate also is 0. If the value of slope is less than 0, this means that the straight trend line is a diagonal line below the horizontal line. If the value of slops is greater than 0, this would mean that the straight trend line is a diagonal line above the horizontal line.

Based on above analysis, we can create the following theorem.

**Theorem**: We assume that a master node sends a timestamp per $T_{period}$ interval, and the drift rate of the slave clock meets assumption 2. Based on the samples of the slave nodes collected, we can achieve a linear trend formula as follows:

$$y = b_0 + b_1 x$$

Here, b1, as a slope, denotes the drift speed of the slave clock, and has a tight relationship with drift rate $_i$ of this clock. As the number of samples increases, the following formula holds:

$$\tau_i = \lim_{\eta \to \infty}(b_1 / T_{period})$$

Based on above theorem and analysis, a conclusion is given as follows:

**Conclusion**: If we have calculated the slope $b_1$ of linear equation from the sampling data, the following statements proves correct:

If $b_1<0$, the *i*th slave clock runs slower that the master clock.

If $b_1=0$, the *i*th slave clock run same speed as the master clock.

If $b_1>0$, the *i*th slave clock run faster than the master clock.

In order to prove this reasoning, we have designed a simulation environment in the next section.

## 4 Simulation

Our simulation is based on the Internet environment. Its topology is shown in Figure 4.

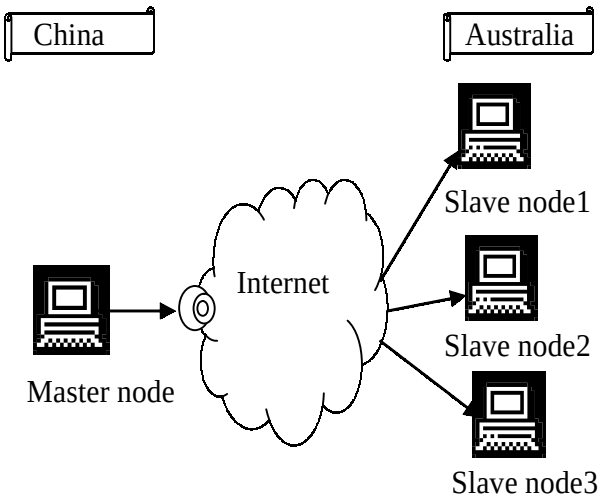


**Figure 4: The simulation environment of clock synchronization**

In this simulation environment, one master node is located in Beijing, China, and three slave nodes are located in Melbourne, Australia. These two sites are connected by Internet. Our simulation uses UDP protocol to communicate with each other.

In this case the virtual clock in the master node is a standard clock. The sending function running in the

master node takes charge of sending standard timestamp included in UDP packages to slave nodes. In our simulation, this node sends a timestamp message per 30 seconds. For simplification, we only show two simulation results from slave node 1 and slave node 2.

For slave node 1, its clock runs normally and the precision difference between the master clock and this slave clock is very small. This node collects timestamps from the master node per 60 seconds.

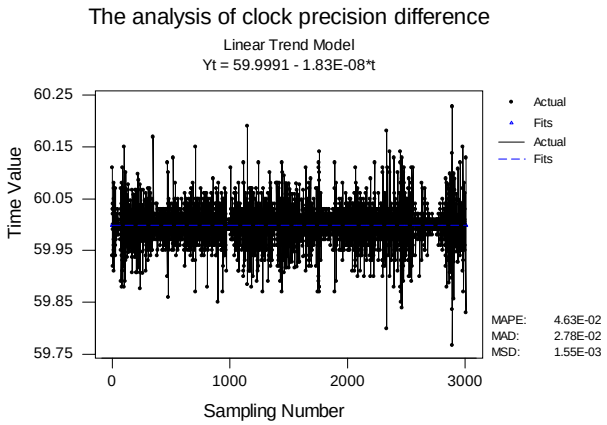


**Figure 5: The simulation of clock precision difference using 3000 samples for salve node 1**

>From figure 5, we can see that the trend line is near a horizontal line even though there are many uncertain samples. The slope of this trend line equation is -1.83E-8. This means that every 60s, the slave clock lag down 1.183E-8s. The real drift rate is expressed by following formula:

$$Drift\_rate = \frac{-1.83\times10-8}{60} = -3.05E-10$$

This means the clock in slave node 1 lag down 30.5 ns per 1 s. In this case, we only show that their precisions are very similar. Normally, the maximum hardware clocks drift rate in computers is of the $10^{-4}$ to $10^{-6}$.

For slave node 2, in order to get results quickly, we use a program to slow down the speed of the slave clock. Every minute that the slave clock runs, it slows down by 4 ms. On the slave side, we use 800 sampling results to demonstrate our idea. The simulation result is showed in figure 6.

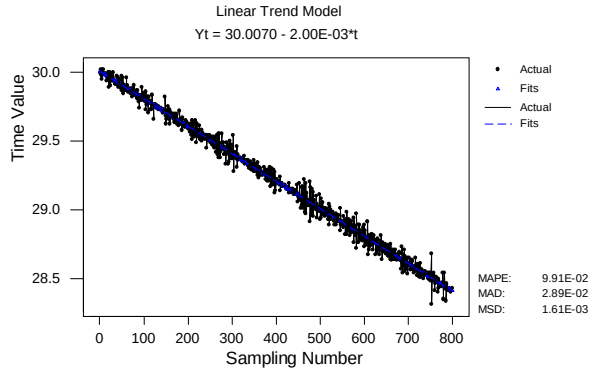


**Figure 6: The simulation with clock precision difference using 800 samples for slave node 2**

In this simulation we have only used 800 samples to achieve a trend line. From figure 6 we can easily conclude that the slave clock runs slower than the master clock considerably as time goes by.

Actually, the precision difference between the master clock and the slave clock has a direct relation with the slope of this equation. The slope reflects the deviation of the slave clock from the master clock. Now we use different number of samples to observe the slope and intercepts of trend equation.

| Number of samples | Intercept | slope |
|---|---|---|
| 7950 | 30.0075 | -0.002 |
| 7500 | 30.0075 | -0.002 |
| 7000 | 30.0075 | -0.002 |
| 6000 | 30.0075 | -0.002 |
| 5000 | 30.0075 | -0.002 |
| 4500 | 30.0072 | -0.002 |
| 4000 | 30.0073 | -0.002 |
| 3500 | 30.0072 | -0.002 |
| 3000 | 30.0073 | -0.002 |
| 2500 | 30.0072 | -0.002 |
| 2000 | 30.0072 | -0.002 |
| 1500 | 30.0072 | -0.002 |
| 1000 | 30.0072 | -0.002 |
| 800 | 30.0070 | -0.002 |
| 600 | 30.0068 | -0.002 |
| 500 | 30.0063 | -0.002 |
| 400 | 30.0067 | -0.002 |
| 300 | 30.0063 | -0.00199 |
| 250 | 30.0057 | -0.00199 |
| 200 | 30.0050 | -0.00198 |
| 160 | 30.007 | -0.00201 |
| 130 | 30.0082 | -0.00204 |
| 100 | 30.0074 | -0.00202 |
| 70 | 30.0061 | -0.00197 |
| 50 | 30.0055 | -0.00195 |
| 40 | 30.0038 | -0.00185 |
| 30 | 30.0012 | -0.00158 |
| 20 | 30.0101 | -0.00263 |
| 10 | 29.9973 | -0.00515 |

**Table 1: The relation between the number of samples and trend equations**

In Table 1, when the algorithm receives fewer samples, the trend equation is unstable. This means that values of intercept and slop changed frequently. As the number of samples increases, especially when the number of samples is greater than 400, we find that the slop of the trend equation is very stable. This is so even though there is little change in the values of the intercept. The change of intercept demonstrates the initial transmission delay. As the number of samples increases, so does the stability.

However, its stable process is slower than the stable process of the slope.

From these simulations, we can conclude that even though the transmission delay exists on the networks, nodes on the slave side also can calculate their precision difference to the master clock by statistical methods. If every slave clock knows its clock precision difference to the master clock, the slave node can easily tune its clock fast or slow only if its initial time is correct.

From the above figures we observe, as a sample, that the transmission delay is uncertain. When we increase the number of samples, we find a strong linear relation between the running of the master clock and running of the slave clock. In this relation, the slope delegates the running speed of the slave clock.

Based on our simulation and analysis, we can achieve our theorem and the conclusion.

## 5 Self-Adaptive Architecture

Based on the model we discussed previously, the local nodes clock state is easy to know. Each node can use this model to locally adjust itself by the different demands of their deviations. When a local node can receive timestamps from the master node normally, its model created before can tune itself based on the arrival data. This kind of model can be called the self-adaptive model.

We assume the difference in time between the master clock and the *i*th slave clock should be less than a threshold $\varepsilon_i$ and the drift rate of *i*th slave clock is $\tau_i$, the maximum time needed by *i*th slave node to tune its clock is:

$$T_{i\max} = \frac{\varepsilon_i}{|\tau_i|} + \frac{\varepsilon_i \tau_i}{|\tau_i|} = \frac{\varepsilon_i (1+\tau_i)}{|\tau_i|} \qquad if \quad \tau_i \neq 0$$

Because $\tau_i$ is calculated by scope *b1* and $T_{period}$, $\tau_i$ may be a positive value, a negative value or zero according to the *b1* value. This value also abides by the conclusion.

When the *i*th slave clock runs and arrives at time value $T_{imax}$, its clock should be adjusted. The new value $T_{inew}$ of its clock should be:

$$T_{inew} = T_{i\max} - \frac{\varepsilon_i \tau_i}{|\tau_i|} \qquad if \quad \tau_i \neq 0$$

The formula above is the simplest method of tuning the local clock. If the value of $\tau_i$ is negative, this means that the slave clock runs slower than the master clock. In this case, we know that $T_{imax}$ is the local time of this slave clock, and its value is less than the real time value. For our simulation, as the slave clock runs it slows down 2 ms per 30 s. We assume that the initial times of both the slave clock and the master clock are equal to 0. This period of 30 s is a standard or real time. After the master node sends 500 timestamps, the real running interval is 30*500=15000s. The local time of this slave clock is 30*500-0.002*500=14999s. Based on the statistical analysis and formula, we can calculate the drift rate of this slave clock as follow:

$$\tau_i = -0.002/30$$

We also assume that when the slave clock lags 1s ($\varepsilon_i$=1s), the clock needs to be adjusted. Now we need to know when the slave clock is adjusted. Based on the formula, we can deduce this time:

$$T_{i\max} = \frac{\varepsilon_i (1+\tau_i)}{|\tau_i|} = \frac{1(1-0.002/30)}{|-0.002/30|} = 14999s$$

If we use the simplest method to tune this clock, the new value of this clock is shown as follows:

$$T_{inew} = T_{i\max} - \frac{\varepsilon_i \tau_i}{|\tau_i|}$$

$$= 14999 - \frac{1*(-0.002/30)}{|-0.002/30|} = 15000s$$

From the two calculations above, when we know the threshold $\varepsilon_i$ and drift rate $\tau_i$, we can easily get the value of $T_{imax}$, and set the current time of the slave clock to real time or standard time $T_{inew}$.

However, this method can cause a time discontinuity problem, which inturn may confuse the run-time system into making an incorrect judgment on real-time tasks. The solution to this discontinuity problem can be found in (Ryu, Park, and Hong 1999). This is also a complicated issue.

## 6 Conclusion

This paper presents the concept of using continuous time intervals to synchronize all clocks of computers in asynchronous environments. In the process of clock synchronization, the difference of time precision is applied, rather than the instantaneous time value. This difference not only reduces the effect of network delay, but also provides important information for local clocks. Based on this information, a local clock cannot only predict the active-synchronization time, but can also apply a self-adaptive state when its connection crashes.

From a long-term viewpoint, the precision of synchronization will be improved gradually in our model. More information of precision differences means more accurate clock synchronization. Therefore, this method can lead to a higher level of clock precision in a distributed system.

The authors would like to thank Dr. Fuchun Huang and Mr. Gang Li, who provided us with relevant suggestions about our mathematical model.

## 7 References

AZEVEDO, M.M. and BLOUGH, D.M. (1998): Multistep Interactive Convergence: An Efficient Approach to the Fault-Tolerant Clock Synchronization of Large Multicomputers. *IEEE Transactions on Parallel and Distributed System*, Vol. 9, No. 12, December 1998:1195-1212.

BOUZELAT, A. and MAMMERI, Z. (1997): Simple Reading, Implicit Rejection and Average Function for Fault-Tolerant Physical Clock Synchronization. *The 23rd EUROMICRO Conference'97 New Frontiers of Information Technology*: 524-530.

BUTLER, R.W. (1988): A Survey of Provably Correct Fault-Tolerant Clock Synchronization Techniques. *NASA Technical Memorandum 100553*, Langley Research Centre, Feb. 1988.

CIUFFOLETTI, A. (1999): Uniform Timing of a Multi-cast Service. *Proc. of 19th IEEE International Conference on Distributed Computing Systems*, Austin, Texas-USA, May, 1999: 478-487.

CRISTIAN, F. (1996): Synchronous and Asynchronous Group Communication. *Comm. ACM*, Vol.39, No. 4, Apr. 1996:88-97.

CRISTIAN, F. (1991): Reaching Agreement on Processor-Group Membership in Synchronous Distributed System. *Distributed Computing*, Vol. 4, 1991: 175-187.

CRISTIAN, F. and FETZER, C. (1999): The Timed Asynchronous Distributed System Model. *IEEE Transactions on Parallel and Distributed System*, Vol.10, No.6, June 1999: 642-657.

LAMPORT, L. and MELLIAR, S.P. (1985): Synchronizing Clocks in the Presence of Faults. *Journal of the ACM 32*(1), January 1985: 52-78.

LISKOV, B. (1993): Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing*, Vol. 6, 1993: 211-219.

MOCK, M., FRINGS, R., NETT, E. and TRIKALIOTIS, S. (2000): Continuous Clock Synchronization in Wireless Real-Time Applications. *The 19th IEEE Symposium on Reliable Distributed Systems*, 2000: 125-132.

RYU, M., PARK, J. and HONG, S. (1999): Timing Constraint Remapping to Avoid Time Discontinuities in Distributed Real-Time System. *The Fifth IEEE Real-Time Technology and Applications Symposium* 2 - 4 June, 1999:89-98.

SINHA, P.K. (1996): Distributed Operating Systems. *IEEE Communications Society, IEEE Press*: 286-300.

SRIKANTH, T.K. and TOUEG, S. (1987): Optimal Clock Synchronization. Journal of the ACM 34(3), July 1987:626-645.