Low-Cost Hybrid Internal Clock Synchronization Mechanism for COTS PC Cluster

Jorji Nonaka[1,⋆], Gerson H. Pfitscher[1], Katsumi Onisi[2], and Hideo Nakano[2]

[1] Department of Computer Science, University of Brasilia, Brasília-DF, Brazil
jorji@ais.sys.i.kyoto-u.ac.jp, gerson@unb.br
[2] Media Centre, Osaka-City University, Osaka, Japan
{onisi,nakano}@media.osaka-cu.ac.jp

Abstract. NTP is a well-known and widely used clock synchronization mechanism for PC cluster environments. However, like other software-based clock synchronization algorithms, its precision depends on the estimated accuracy of the network latency for synchronization messages. This paper presents a low-cost internal clock synchronization mechanism that uses a simple TTL signal distributor to support remote clock reading to obtain the time drift necessary to execute local clock adjustments. The objective is to improve precision, thus eliminating the need to estimate the network latency as in usual methods.

1 Introduction

Synchronized clocks are useful in distributed systems for performance measurements, auditing, event ordering and task scheduling, among other things. Typically, COTS (commodity off-the-shelf) PC clusters do not have access to a global clock or possess dedicated clock synchronization support. The goal of the internal clock synchronization is to minimize the maximum difference between any two clocks. Most internal clock synchronization algorithms, such as the *Network Time Protocol* (NTP)[1], are software-based. They are flexible and economical, but the performance is limited by the synchronization message transit delay. This paper presents a low-cost hybrid internal clock synchronization mechanism and its implementation on a COTS PC cluster running Linux. A simple signal distributor hardware and the parallel printer ports of the machines are used to support remote clock reading to improve the precision, thus eliminating the need to estimate the network latency.

Clock synchronization has been extensively studied for the last two decades and *hardware*, *software* and *hybrid* approaches have been proposed[2,3].*Hardware* approaches[2,3,4] achieve μs level precision through the use of dedicated hardware at each node and a separate network solely for the propagation of clock signals. Cost, however, has been a limiting factor. *Software* approaches do not need any extra hardware but the performance is limited by the synchronization message transit delay and provides a precision in the ms range[5,2]. Figure 1

⋆ Currently with Graduate School of Informatics, Kyoto University, Japan.

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 121–124.

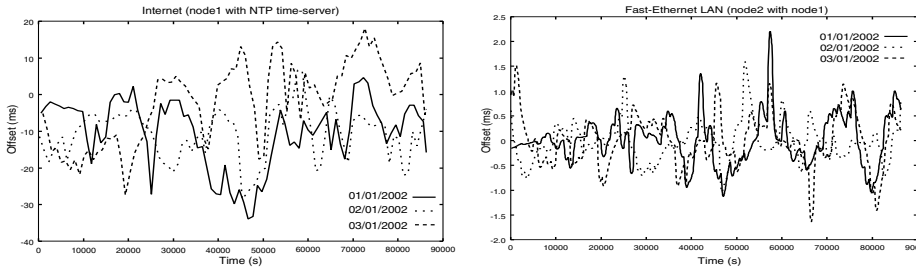

Fig. 1. External and Internal clock synchronization via NTP.

shows the interference of network latency on the clock synchronization precision obtained via NTP in two different network environments: Internet and Fast-Ethernet LAN.

Due to that limitation, a *hybrid* approach[3,5] has been proposed. This approach requires minimum extra hardware and uses the available network infrastructure for synchronization message exchanges. Although the extra hardware is considered to be minimum, these solutions usually require expensive apparatus such as a precise quartz crystal oscillator[3,5], a GPS signal receiver[5,6] or a custom LSI[4,5]. Even the most recent version of NTP, based on a *Nanokernel*[6]algorithm, needs a precise pulse per second (PPS) signal produced by an external device to obtain better performance. We refrained from using expensive hardware, such as those used in other methods, to match with the Beowulf-class PC cluster philosophy.

2 Clock Synchronization Mechanism

The on-going method uses the usually inactive PCs' I/O ports, such as serial or parallel ports, which can generate hardware interruptions. The TTL level signal distributor hardware is used to simultaneously signal the I/O ports to start a reading process of each local clock. After that, we obtain the instantaneous image of all clocks' value involved in the synchronization. The available network infrastructure is used to transmit the reference clock's value to the other nodes, to calculate the time offset in order to adjust each local clock in an appropriate way. This way, although the clock value is transmitted through the available network there is no need to calculate the message transmission latency as would be the case in a purely software based algorithm. Figure 2 shows a simplified scheme of the synchronization mechanism and the implemented hardware.

3 Implementation Issue and Results

The PC cluster on which we have implemented our model consists of 8 Pentium II 350 MHz with 64MB RAM interconnected by a Fast-Ethernet network. We used kernel version 2.2.16 of the linux operating system and NTP version

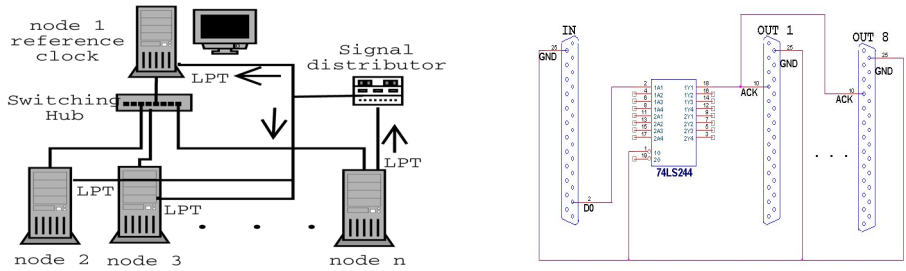

Fig. 2. Hybrid internal clock synchronization mechanism.

4.0.99k. The NTP was used for external clock synchronization of the reference node's clock. We have elaborated a simple synchronization signal distributor hardware, a character oriented device driver for parallel ports, and the master and slave processes. The choice of the parallel printer port was only due to its easy manipulation and programming.

The synchronization signal distributor hardware distributes a TTL level signal to the *Acknowledge* (ACK) parallel port pin to generate hardware interruption *IRQ7* which causes the device driver to read the local clock through the kernel function *do_gettimeofday* with 1 μs resolution and makes its value available in the *"/dev/sincro"* device. Then it wakes up the slave processes to begin the resynchronization by requesting the master process for the reference clock value to adjust each local clock. This is done through the system call *adjtimex* derived from the NTP clock discipline algorithm[1,6], which has been used by Linux since 1992.

During the experiments, the processes running on the cluster were reduced to a minimum level and we observed the clock drift behavior of all node clocks (*node2* to *7*) in relation to the reference clock (*node1*)without any local clock adjustments. The result can be viewed in figure 3. We used several resynchronization interval periods to observe the clock synchronization behavior and obtained hundreds, tens and few microseconds precision using, respectively, 30s, 3s and 1s as the pulse intervals. Figure 3 for 3s pulse interval, unlike figure 1 shows only the clock offset before each resynchronization process instead of the real-time behavior. We did not work on the clock adjustment algorithm and only used available function *adjtimex* to cancel out the time-offset. Before each resynchronization process, each clock drifts according to its own drifting rate.

In comparison to the NTP we have obtained better and more homogeneous results in the time-offset variation as time passes even using 30s as the resynchronization interval. Using smaller intervals the processing and communication overheads increase. It is therefore necessary to choose an optimum interval matching the application's requirements. The computational cost of the slave process that consumes most processing time is less than 0.03% of the CPU time. During the experiments, we observed some interferences that appear in the graph as spikes. One of them is the *reading error* that produces data inconsistency and the other

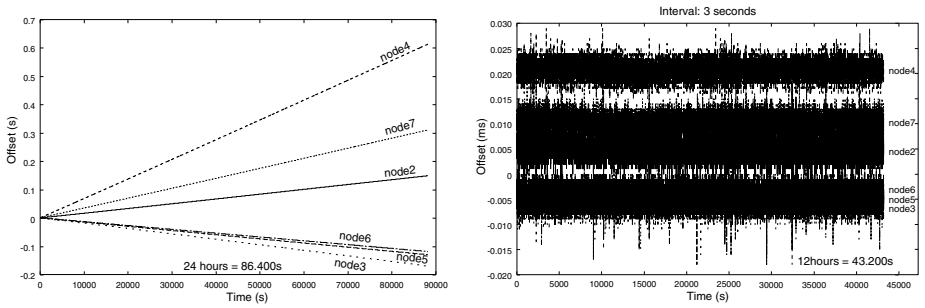

Fig. 3. Clock drift and internal synchronization using an interval of 3s.

is the noise generated by external devices, such as monitors and transformers. These values are ignored by the data filtering subroutine.

4 Conclusions

This paper has described an overview of a low-cost hybrid internal clock synchronization mechanism. The performance evaluation in a small size COTS PC cluster running Linux has shown that this method can be a good alternative to improve internal clock synchronization without using any expensive extra hardware. However, the implemented model still requires improvements to be suitable for the real world. This model has great scalability in systems with the condition that the nodes are placed physically close. A next step might be to analyze the dependence between the precision and the signal generation interval, making it possible to construct an adaptive clock synchronization mechanism.

References

1. Mills, D. L.: Internet Time Synchronization: The Network Time Protocol. *IEEE Micro Trans. Communications*, 39(1):1482-1493, January 1991.
2. Anceaume, E., Puaut, I.: Performance Evaluation of Clock Synchronization Algorithms. *Technical report 3526*, INRIA, France, October 1998.
3. Ramanathan, P., Kandlur, Dilip D., Shin, Kang G.: Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems. *IEEE Transactions on Computers*, 39(4):514-524, April 1990.
4. Kopetz, H., Ochsenreiter W.: Clock Synchronization in Distributed Real-Time Computer Systems. *IEEE Transactions on Computers*, C-36(8):933-940, August 1987.
5. Horauer, M.: Hardware Support for Clock Synchronization in Distributed Systems. *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Gōteborg, Sweden, 2001.
6. Mills, D.L.,Kamp, P.: The Nanokernel. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting*, Reston VA, USA, 2000.