Hindawi Publishing Corporation
Journal of Computer Systems, Networks, and Communications
Volume 2008, Article ID 583162, 7 pages
doi:10.1155/2008/583162

Research Article

Internal Clock Drift Estimation in Computer Clusters

Hicham Marouani and Michel R. Dagenais

Department of Computer and Software Engineering, Ecole Polytechnique, P.O. Box 6079, Downtown, Montreal, QC, Canada H3C 3A7

Correspondence should be addressed to Michel R. Dagenais, michel.dagenais@polymtl.ca

Received 22 August 2007; Revised 12 February 2008; Accepted 4 May 2008

Recommended by Wael Badawy

Most computers have several high-resolution timing sources, from the programmable interrupt timer to the cycle counter. Yet, even at a precision of one cycle in ten millions, clocks may drift significantly in a single second at a clock frequency of several GHz. When tracing the low-level system events in computer clusters, such as packet sending or reception, each computer system records its own events using an internal clock. In order to properly understand the global system behavior and performance, as reported by the events recorded on each computer, it is important to estimate precisely the clock differences and drift between the different computers in the system. This article studies the clock precision and stability of several computer systems, with different architectures. It also studies the typical network delay characteristics, since time synchronization algorithms rely on the exchange of network packets and are dependent on the symmetry of the delays. A very precise clock, based on the atomic time provided by the GPS satellite network, was used as a reference to measure clock drifts and network delays. The results obtained are of immediate use to all applications which depend on computer clocks or network time synchronization accuracy.

1. INTRODUCTION

Complex distributed computer systems are increasingly used to offer a growing array of online services from search engines to groupware and eCommerce. Each computer will typically contain several processors and possibly many disks. A request served by such a cluster may go through a load balancing frontend to one of several web servers, which will in turn make requests to authentication servers, database servers, and file servers before returning the answer.

When the answer is incorrect, or even more if the performance is below expectations, it may be extremely difficult to understand and diagnose the problem [1]. This is where detailed system tracing can provide the needed information by instrumenting the servers. Limited tracing tools were available with closed source operating systems. More recently, with the increasing popularity of open source operating systems, new tracing tools have appeared. Some tracers like DTrace [2] and SystemTap [3] allow the dynamic insertion of tracepoints, at some cost in performance. DTrace is offered by Sun for Solaris (now OpenSolaris) while SystemTap is an ongoing project of Red Hat, IBM, Intel, and Hitachi. Other tracers rely on static instrumentation, needing a kernel recompilation but minimizing the overhead and thus the system disturbance. The Linux Trace Toolkit, LTTng [4–6], is probably the best known system in this category. SystemTap and LTTng share some of the underlying trace recording technology, Relay, and it should be relatively easy in the future to combine events recorded from these two sources in a single trace.

Each CPU in a multiprocessor computer may record events. It typically uses its cycle counter to timestamp each event before recording it in a per CPU buffer. A separate tracing flow is thus obtained for each CPU. In some architectures, all cycle counters start with the same count, at the same time, and are connected to a shared clock signal. However, it is more typical to have independent clocks, possibly derived from a common clock signal. In that case, the different CPUs may not power up and start counting exactly at the same time, and they may even use different clock scaling values, for instance depending on their internal temperature. Different computers in a cluster will not only have independent clocks but may also exhibit more differences in terms of technology and architecture [7].

The objective is to estimate the clock differences over time with respect to a time reference to be used in the trace viewer [8]. The time reference may or may not correspond to a real clock but is the reference used to present a global time view of the events, irrespective of their CPU of origin. Often, a CPU in a central server, or one connected to a very precise clock source, will be a convenient choice as time reference. Each event must thus have its locally generated timestamp converted to the time reference, using the estimation of the clock differences over time.

The clock synchronization, computing the difference between clocks on two CPUs, is performed either by connecting to a shared clock signal (e.g., the output from an atomic clock) or by sending messages back and forth with the current time on each system and compensating as much as possible for the network delay. The synchronization may be performed live, adjusting the current time values to reduce any clock difference found. It may also be performed a posteriori, computing at trace analysis time the clock differences over time, and using these values to convert the local timestamps to the time reference.

In this article, several experiments were conducted to estimate the accuracy and stability of cycle counters and to measure the network delay values and variability. Several different units of the same model, of different models but same architecture, and of different architectures were tested while varying the system load and the temperature. Similarly, the network delays were measured using different network adapters and switching equipment. A very precise clock, generated by a GPS receiver optimized for time precision, was used as the reference clock. Thus, the main contribution of this study is the experimental results which provide a precise measurement of the characteristics of the computers and networking equipment commonly found in the distributed systems to be traced. A second contribution is the efficient experimental setups proposed to measure the relative clock accuracy and stability of several networked computers, using a GPS and a modified Linux kernel.

2. CPU CLOCKS

In earlier architectures, two hardware devices were traditionally used to update the operating system internal time [9]: the battery-backed real time clock (RTC) and the programmable interrupt timer (PIT). The real time clock is used to maintain the time even when the computer is off. In IBM PC compatible computers, the RTC circuit is the Motorola 146818, with a resolution of approximately one second and a significant drift, and the PIT circuit is the Intel 8254. The Linux operating system uses the RTC at boot time to initialize its internal time. It then updates its internal time by requesting regular interrupts from the PIT, typically every millisecond, and adding this value to its time variable. The Linux system call *gettimeofday()* retrieves this internal time.

The majority of recent microprocessors, starting with the Pentium in the i386 architecture, have a built-in clock cycle counter. This cycle counter, when available, is used by the Linux operating system to interpolate the time between PIT interrupts, thus increasing the resolution from 1 millisecond to 1 microsecond or better. In Intel Pentium processors, this register is named TSC (TimeStamp Counter) and is 64 bits long.

Microprocessor clock signals are typically generated with a circuit based on a quartz crystal. The clock precision [7], or drift rate, is measured as the offset between the clock and a very precise reference clock per unit of time, and is often expressed as a ratio in parts per million. Values of 1 ppm or 10^{-6} are typical for computer grade crystals (1 microsecond every second or 0.6 second per week). The circuit temperature directly affects the crystal frequency and consequently the drift.

3. EXPERIMENTAL SETUP

Each satellite in the Global Positionning System (GPS) contains a very accurate atomic clock. GPS receivers obtain time values and coordinates from several satellites and can accordingly compute their position and the time. The Motorola M12+ Timing Oncore GPS receiver [10] offers approximately 12 nanoseconds accuracy with respect to Universal Time Coordinated, and was used as a time reference to perform the different measurements. This GPS receiver produces a pulse per second (PPS) signal which was connected directly to a pin of the computer RS-232 serial port, from which an interrupt can be generated. For some of the experiments, the signal cable was split and connected to several computers using short wires of equal length.

The PPS signal reaches each CPU with a precision limited by the variability of the delay outside and inside the computer [11–13]. The delay outside the computer (i.e., speed of electric signal in copper at 5 ns/m) is fairly constant, and easily compensated, and thus has no effect on the drift measurement. The internal delay is variable and corresponds to the difference between the signal arrival time and the execution of the cycle counter read instruction in the serial port interrupt handler. The same study [11] reports that for a Pentium III at 860 MHz, the mean delay is 8.31 microseconds with a standard deviation of 0.36. This delay is the sum of the hardware interrupt controller and the operating system latency.

For our experiment, the lowest-level interrupt handler in Linux kernel version 2.4.26 was instrumented, as shown in Algorithm 1, to sample the cycle counter as soon as the PPS signal interrupt is notified [14, 15]. Thereafter, the difference between two successive readings is calculated, yielding the number of clock cycles per second, or clock frequency, for the CPU.

In the networking experiments, the same PPS signal is connected to the two computers. In a first setup, the two computers exchange Ethernet packets in order to measure the network delays. In a second setup, a third computer broadcasts an Ethernet packet to the two PPS connected computers. The difference in broadcast packet arrival time can then be measured [16].

```
/* Pseudocode for modified low level interrupt handling routine */
do_IRQ ()
{
    /* Modification to read the cycle counter as early as possible */
    if ( interrupt is 4 (serial port) and clock experiment active ) {
        read TSC cycle counter and store in buffer;
        increment buffer pointer;
    }

    /* Normal content of do_IRQ in Linux */
    Call the specialized interrupt handler based on interrupt number;

    /* Modification for transferring the results to user space */
    if ( clock experiment active and buffer is full ) {
        check that the alternate buffer was read by the daemon;
        switch to the alternate buffer;
        signal the daemon to read the filled buffer;
    }
}
```

Algorithm 1: Pseudocode for the modified main interrupt request handler, used to sample the cycle counter at every GPS pulse per second in the Linux kernel.

Table 1: CPU clock frequency versus CPU temperature in Celsius.

Temperature (Celsius)	Frequency (Hz)	Standard deviation (Hz)
28	349205333	464.67
32	349204698	1686.58
36	349204068	1878.87
40	349203314	2299.72
44	349202799	1469.95
47.25	349202435	1929.43

Table 2: Measured clock frequency for light, moderate, and heavy system loads.

	Light load	Medium load	Heavy load
Frequency (Hz)	350797133	350797332	350797453
Temperature (Celsius)	34.5	34.5	34.5
Standard deviation (Hz)	149	331	426

4. RESULTS

4.1. Effect of temperature on clock frequency

For this experiment, the exact clock frequency (number of cycles between two PPS signals) is measured while the temperature varies from room temperature to the maximum rated temperature. An Intel Pentium II 350 MHz system was used with the CPU fan disconnected, and started at room temperature. The CPU temperature value is obtained through the Linux operating system from the health monitoring chip on the motherboard.

Table 1 illustrates that the CPU frequency diminishes at the rate of approximately 150 Hz per degree Celsius. This is consistent with other studies [17, 18] showing a linear relation between the frequency and the temperature, with a positive or negative slope depending on the specific circuit used as clock driver.

4.2. CPU load

In this experiment, the effect of system load on the frequency is characterized. The effect may be twofold. An increased load may affect power usage and heat dissipation. Indeed, when the processor is idle, several units (e.g., ALU) are inactive and do not consume power. Another possible effect of higher system load is an increase in interrupt latency variability, affecting the delay to read the cycle counter after the PPS signal, and thus the accuracy of the frequency measurement.

Again, an Intel Pentium II 350 MHz system was used for this experiment. It was tested with *no load*, only running the frequency data collection daemon, *moderate load* archiving the Linux Kernel source code with command *tar*, and *heavy load* where several processes are added to *tar* and perform intensive floating point computations on arrays. For each case, the frequency was measured at every second for one hour.

The load variation has little effect on either the frequency or the temperature, which are closely related as shown in Table 2. The CPU fan appears fairly efficient at keeping the CPU close to room temperature despite the power dissipation that could be caused by the system load variations. On the other hand, the standard deviation of the measured frequency increases significantly with the load. Indeed, the heavy load has a direct impact on the interrupt latency since

Table 3: Clock frequency for 5 1.3 and 3 1.5 GHz AMD microprocessors.

Model	Frequency (Hz)	99% interval (μs)
AMD 1.3 number 1	1 343 172 180	0.492
AMD 1.3 number 2	1 343 175 836	0.475
AMD 1.3 number 3	1 343 134 661	0.702
AMD 1.3 number 4	1 343 175 117	0.419
AMD 1.3 number 5	1 343 174 308	0.664
AMD 1.5 number 1	1 533 362 884	0.608
AMD 1.5 number 2	1 544 642 975	0.510
AMD 1.5 number 3	1 544 665 817	0.498

Table 4: Clock frequency for 8 Intel Pentium IV 2.4 GHz microprocessors.

Model	Frequency (Hz)	99% interval (μs)
Intel 2.4 number 1	2 393 902 454	2.023
Intel 2.4 number 2	2 393 896 497	2.357
Intel 2.4 number 3	2 393 925 792	1.977
Intel 2.4 number 4	2 393 882 326	1.861
Intel 2.4 number 5	2 393 889 589	2.008
Intel 2.4 number 6	2 393 857 038	2.128
Intel 2.4 number 7	2 393 886 405	2.217
Intel 2.4 number 8	2 393 882 430	1.901

a loaded system spends more time in code sections where interrupts are temporarily disabled. The interrupt latency does not affect the clock frequency but rather the accuracy of its measurement, since the reading of the cycle counter in the interrupt following a PPS signal may be delayed.

The effect of interrupt latency is relatively easy to mitigate in a posteriori analysis over a long period of time. By measuring longer intervals, the relative effect of interrupt latency can be decreased. Furthermore, by excluding values too far from the average measurement, it is possible to take out the cases where interrupts were significantly delayed.

5. CLOCK ACCURACY FOR DIFFERENT COMPUTER MODELS

The clock frequency accuracy and stability was measured for computer motherboards from several manufacturers: 5 units of AMD 1.3 GHz ASUS A7A (AMD1.3), 3 units of AMD 1.5 GHz ASUS A7A (AMD1.5), 8 units of Intel Pentium IV 2.4 GHz Intel D865PES (Intel2.4), 7 units of Intel Pentium II 266 MHz (Intel266), 1 unit of VIA 600 MHz EPIA ME6000 (VIA600), and 1 unit of Intel Itanium 1.4 GHz biprocessor server SR870BH2 (Itanium1.4). In each case, the average frequency was measured over a one-hour period, and the frequency interval within which fall 99% of the samples was computed.

The results obtained are shown in Tables 3, 4, 5, and 6. The AMD boards tested have the smallest 99% interval. It may be caused by a simpler and/or better interrupt controller or a more precise clock generation circuit.

Table 5: Clock frequency for 7 Intel Pentium II 266 MHz microprocessors.

Model	Frequency (Hz)	99% interval (μs)
Intel 266 number 1	266 317 841	1.057
Intel 266 number 2	266 314 530	1.137
Intel 266 number 3	266 314 969	1.168
Intel 266 number 4	266 318 621	1.544
Intel 266 number 5	266 313 279	1.074
Intel 266 number 6	266 316 786	2.641
Intel 266 number 7	266 312 792	1.263

Table 6: Clock frequency for different computer models. The frequency and interval of the 5 AMD 1.3 GHz, 3 AMD 1.5 GHz, 8 Intel 2.4 GHz, and 7 Intel 266 MHz have been averaged.

Model	Frequency (Hz)	99% interval (μs)
AMD1.3	1 343 166 500	0.550
AMD1.5	1 540 890 559	0.539
Intel2.4	2 393 890 316	2.085
Intel266	266 315 545	1.412
VIA600	599 924 976	1.617
Itanium1.4	1 396 283 528	2.584

6. CLOCK STABILITY

The short-term clock variations presented in Table 6 may in large part be caused by variable interrupt latencies. Interrupt latency causes a clock frequency measuring error but, unlike frequency drift, does not accumulate over time. Thus, assuming that the CPU temperature does not change much (since the load has little effect on the temperature as shown in Table 1) and that aging is a very long-term factor, the clock frequency of a system may remain fairly stable over a tracing session (which may last anything from a few seconds to a few hours).

A convenient way to estimate the clock stability is to calculate the Allan variance, or its square root, the associated Allan deviation [19, 20]. The Allan variance is based on the difference between successive readings of the frequency difference from the reference clock, sampled at a fixed interval. Its advantage is that it converges over time for most typically encountered types of noise. The traditional variance uses the difference between each sample and the average.

The clock frequency of 8 AMD processors was measured over a period of several hours. The Allan deviation was plotted over time as more measurements were added. The Allan deviation rapidly diminishes and converges to a steady state, as shown in Figure 1. If the clock frequencies drifted in a measurable way, the values would systematically vary over time and the Allan deviation would tend to increase with time.

Similarly, the clock frequency of an Intel Pentium II 350 MHz was monitored for longer periods of several days [14]. The clock frequency still exhibited excellent stability over these longer periods. However, it is interesting to note that a cyclic daily variation clearly appeared, from a high of 350 797 150 to a low of 350 796 890 Hz, for a difference of

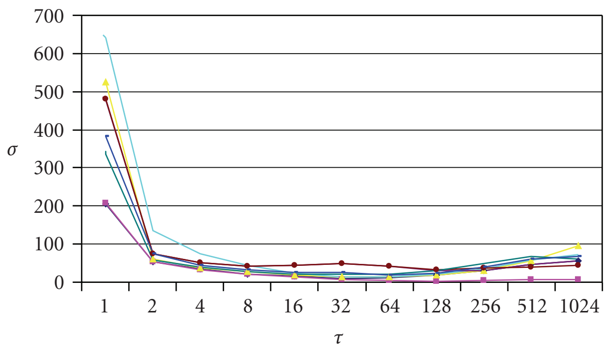

Figure 1: Allan deviation for 8 AMD processors.

260 cycles. The frequency maintains a close to average value between 10h00 and 23h00, rises from 23h00 to 6h30 in the morning and then drops sharply from 6h30 to 8h30, before coming back gradually to the average value around 10h00. These small variations of plus or minus 130 cycles per second are likely caused by variations in the electrical supply and in the ventilation system. The same variations were found when connecting the systems to either a GPS clock or to an independent high-precision clock.

Table 7: Query and response times for 3600 NTP requests between two computers connected by an unmanaged Ethernet switch. All times are in microseconds.

Direction	Average (μs)	Standard deviation (μs)	Minimum (μs)	Maximum (μs)
Request	121.03	1.90	113.64	173.55
Reply	110.53	1.60	102.77	117.30

Table 8: Difference between query and response times for NTP requests on a free versus a busy network, in microseconds.

Network	Average difference (μs)
Network with no traffic	10.493
Busy network	10.805

Table 9: Difference between query and response times for different networking equipment, in microseconds.

Network	Average difference (μs)
Crossover cable	11.000
Managed Ethernet switch	10.161
Simple Ethernet switch	10.493

7. NETWORK TIME SYNCHRONIZATION ACCURACY

The network synchronization protocols are dependent on the packets delivery time. A constant, symmetric, delay between two computers is the ideal case where the send and receive times perfectly compensate for each other. The Network Time Protocol (NTP) [7, 21, 22] is the most popular of such protocols. The same type of clock synchronization algorithm is used a posteriori during trace analysis to align the traced events on a common time base in LTTng [8]. The upper bound on clock difference inaccuracy is the sum of the NTP request and response packets transmission time (physical network delay, network interface card processing time, and operating system latency). The actual error is caused by the time difference between both directions in the NTP request-response roundtrip. The achievable accuracy on a 100 Mb/s Ethernet network is reported to be about 1 millisecond [23]. A tighter bound on synchronization was obtained in [24], 1.45 ± 1.26 microseconds, but using a higher performance Myrinet network and special network interface cards which processed in firmware the synchronization messages.

Two networked Pentium II 350 MHz computers were connected to the same GPS pulse per second signal. Three different network topologies were tested: a direct crossover cable, a simple 8 ports DLINK switch, and an institutional managed switch. In each case, the delays for the NTP like time request and answer packets were measured, thanks to the common GPS time reference.

The first test, in Table 7, measures the request and response times between a client and a server. Both computers contain 350 MHz Intel Pentium II processors but they differ otherwise in terms of motherboard chipset and network adapter. When the server and client roles were exchanged, the send and receive timings were almost reversed, indicating that the hardware configuration and not the client or server role is responsible for the delay difference. The network switching equipment is symmetrical and thus should not contribute to a systematic difference in one way or another. The delay is from the packet send to the packet receive functions in the kernel, which were instrumented with LTTng. The delay values do not vary much in general (small standard deviation) but are sometimes slightly smaller (minimum value) or quite larger (maximum value). Thus, most packets take the same, close to minimal, time, while a few packets may be delayed significantly.

The second test, shown in Table 8, examines the effect of network congestion. The switch appears to be effective as the traffic added to other ports has very little impact on the network delay between the two computers exchanging time synchronization packets. The last test, in Table 9, compares different network switching solutions. The results are very similar. The networking technology (10 Mb/s, 100 Mb/s, Myrinet) and the hardware (network adapter and corresponding operating system driver) appear to have much more influence than the type of switch or the other traffic on the switches.

A different clock synchronization strategy, using broadcast packets, was also tested. A clock server broadcasts the time to several computers connected to the same local area network. Two computers connected to this network and receiving these broadcast packets were at the same time also connected to the common GPS time base.

While broadcast packets should be received almost simultaneously by all computers on the local area network, delays in the switch, network interface card, and operating system increase the time difference. Nevertheless, broadcast packets can achieve an accuracy of a few microseconds,

Table 10: Broadcast time difference, in microseconds, for different networking equipment.

Network	Average difference (μs)
Managed switch	2.411
Simple switch	1.405

which is several times better than what is achievable with the NTP request response packets.

8. CONCLUSION

The main contribution of this experimental study is to provide data about the accuracy and stability of computer clocks and local area network delays, with emphasis on the various parameters that could affect the precision of event timestamps when tracing distributed systems. Several different computers of the same model and of different models and several networking switches were tested.

This study shows that the microprocessor clock cycle counter can be used as a high resolution, high accuracy, low drift, timing system. In the experiments presented, the variations from one measurement to the other were mostly due to the interrupt latency (short-term noise), when processing the pulse per second signal, and not to variations in the clock frequency (long-term stability). Furthermore, the clock frequency exhibited an excellent stability even over several days. The clock frequency does vary with temperature and line voltage but these parameters are not expected to change too much in a server room during regular hours, for example for the duration of a tracing session. While it is possible to vary the temperature by disconnecting the fan, little variation is obtained, for instance by varying the computing load, when the fan is running.

The clock offset between networked computers, with 100 Mb/s Ethernet, can be computed from packet send and arrival time with an accuracy of approximately 10 microseconds. An even better accuracy, around 2 microseconds, would be possible when broadcast packets are received simultaneously by several computers. The variations in network delay time caused by variable interrupt latency can easily be removed when performing a posteriori analysis on a trace containing several message exchanges. Indeed, the excellent clock stability can be used to detect and ignore the few outliers. These normally correspond to the reception of a network packet, and the network adapter raising an interrupt, but this interrupt being delayed because the operating system was in a critical section, with interrupts disabled.

Such a precision of approximately 10 microseconds in the computation of the clock offset to a common time base is more than adequate for presenting a merged view of several traces collected from networked computers. Indeed, if some kernel events are a few microseconds apart (e.g., syscall entry versus syscall exit), all packets exchanges (DSN, NTP or HTTP requests, remote procedure calls, etc.) have durations in the hundreds of microseconds or in milliseconds.

Another interesting contribution of this work is to propose efficient experimental setups to measure the relative clock accuracy and stability of several networked computers. A new, affordable, and extremely precise GPS-based time source was used to perform the experiments, along with a modified Linux kernel to intercept and time external events (pulse per second signal and packet arrival) as early as possible.

ACKNOWLEDGMENTS

Several researchers, Benoit des Ligneris and Mathieu Desnoyers from Ecole Polytechnique, Karim Yaghmour from Opersys, and Tom Zanussi, Robert Wisniewski, and Richard Moore from IBM, contributed to the experimentation through stimulating discussions. The financial support of the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

REFERENCES

[1] M. Bligh, M. Desnoyers, and R. Schultz, "Linux Kernel Debugging on Google-sized clusters," in *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada, June 2007.

[2] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," in *Proceedings of the USENIX Annual Technical Conference*, pp. 15–28, Boston, Mass, USA, June-July 2004.

[3] V. Prasad, W. Cohen, F. C. Eigler, M. Hunt, J. Keniston, and B. Chen, "Locating system problems using dynamic instrumentation," in *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada, July 2005.

[4] K. Yaghmour and M. R. Dagenais, "Measuring and characterizing system behavior using kernel-level event logging," in *Proceedings of the USENIX Annual Technical Conference*, pp. 13–26, San Diego, Calif, USA, June 2000.

[5] M. Desnoyers and M. R. Dagenais, "The LTTng tracer: a low impact performance and behavior monitor for GNU/Linux," in *Proceedings of the Linux Symposium*, pp. 209–224, Ottawa, Ontario, Canada, July 2006.

[6] Linux Trace Toollkit, July 2007, http://ltt.polymtl.ca/.

[7] G. Coulouris, J. Dollimore, and T. Kindberg, "Time and global state," in *Distributed Systems Concepts and Design*, pp. 385–400, Addison-Wesley, Reading, Mass, USA, 2001.

[8] E. Clement, "Synchronisation de traces dans un réseau distribué," Mémoire de maîtrise, Ecole Polytechnique de Montréal, Québec, Canada, August 2006.

[9] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, O'Reilly, Sebastopol, Calif, USA, 2nd edition, 2002.

[10] Motorola GPS, September 2004, http://www.motorola.com/ies/GPS/productstiming.html.

[11] V. Smotlacha, "Measurement of time servers," Tech. Rep. 18/2001, CESNET Association, Prague, Czech Republic, December 2001.

[12] J. R. Ring, C. P. Allen, and S. Snyder, "Adjusting processor clock information using a clock drift estimate," US patent 20060208941, September 2006.

[13] W. Lewandowski, J. Azoubib, and W. J. Klepczynski, "GPS: primary tool for time transfer," *Proceedings of the IEEE*, vol. 87, no. 1, pp. 163–172, 1999.

[14] H. Marouani, "Mesure de la précision des compteurs de cycle et horloge interne des ordinateurs," Mémoire de maîtrise,

Ecole Polytechnique de Montréal, Québec, Canada, October 2004.

[15] H. Marouani and M. R. Dagenais, "Comparing high resolution timestamps in computer clusters," in *Proceedings of the 18th Annual Canadian Conference on Electrical and Computer Engineering (CCEC '05)*, pp. 400–403, Saskatoon, Canada, May 2005.

[16] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. S1, pp. 147–163, 2002.

[17] R. K. Karlquist, L. S. Cutler, E. M. Ingman, J. L. Johnson, and T. Parisek, "A low-profile high-performance crystal oscillator for timekeeping applications," in *Proceedings of the IEEE International Frequency Control Symposium*, pp. 873–884, Orlando, Fla, USA, May 1997.

[18] H. Kawashima and K. Sunaga, "Temperature compensated crystal oscillator employing new shape GT cut quartz crystal resonator," in *Proceedings of the 45th Annual Symposium on Frequency Control*, pp. 410–417, Los Angeles, Calif, USA, May 1991.

[19] D. W. Allan, "The Allan Variance," September 2004, http://www.allanstime.com/AllanVariance/.

[20] D. W. Allan, "Should the classical variance be used as a basic measure in standards metrology?" *IEEE Transactions on Instrumentation and Measurement*, vol. IM-36, no. 2, pp. 646–654, 1987.

[21] D. L. Mills, "Improved algorithms for synchronizing computer network clocks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 245–254, 1995.

[22] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, 2004.

[23] A. Pásztor and D. Veitch, "PC based precision timing without GPS," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 1–10, 2002.

[24] C. Liao, M. Martonosi, and D. W. Clark, "Experience with an adaptive globally-synchronizing clock algorithm," in *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, pp. 106–114, Saint Malo, France, June 1999.