ISSN:0975-9646

Clock Synchronization in Distributed System

Nikhil Khandare, Modraj Bhavsar , Prakash Kumare, Sowmiya Raksha

Veermata Jijabai Technological Institute (VJTI),
Matunga Mumbai-19

Abstract: Computer technology has advanced at a fast and steady rate during recent years. Improvements in VLSI technology and processor architecture have resulted in microprocessors with performance/cost ratios that are several orders of magnitude greater than those available a decade ago. During the same period, and motivated by these advances, parallel computing evolved to become the leading direction towards teraflop-level performance. This paper presents and analyzes a clock synchronization algorithm which is probabilistic that can guarantee a much smaller bound on the clock skew than most existing algorithms. We also discuss the basics of clock synchronization physical clock, logical clock and synchronization algorithms. A closed-form expression that relates the probability of invalidity to the clock skew and the number of synchronization messages is also derived.

Index Terms: *Clock synchronization, deterministic algorithm, distributed systems, master-slave scheme, probabilistic algorithm, probability of invalidity, time transmission protocol*

INTRODUCTION

A hardware clock which is fault free, even if initially synchronized with a standard time reference, tends to drift away from the standard over a period of time. As a result, an interval of time measured with such a clock tends to be in error. However, the rate at which the hardware clock deviates from the standard is bounded by a constant. This constant, known as the maximum drift rate of the clock, typically of the order of 1 microsecond per second A direct consequence of the phenomenon of clock drift is that clocks in a distributed system gradually deviate from each other over a period oft, time. Closely synchronized clocks are necessary in several, important distributed systems applications, including financial transactions, stock trading, airline reservations, hard real-time systems, distributed file systems, authentication, and performance evaluation. A clock synchronization algorithm is used in a distributed system to ensure that the skew that develops between clocks remains bounded. Several clock synchronization algorithms have been proposed in the literature. This paper proposes and analyzes a new clock synchronization algorithm based on a probabilistic approach. The proposed algorithm can guarantee a much smaller bound on the clock skew than most existing clock synchronization algorithms.

PHYSICAL CLOCKS

Most computers today keep track of the passage of time with a battery-backed up CMOS clock circuit, driven by a quartz resonator. This allows the timekeeping to take place even if the machine is powered off. When on, an operating system will generally program a timer circuit (a Programmable Interval Timer, or PIT, in older Intel architectures and Advanced Programmable Interrupt Controller, or APIC, in newer systems.) to generate an interrupt periodically (common times are 60 or 100 times per second). The interrupt service procedure simply adds one to a counter in memory. While the best quartz resonators can achieve an accuracy of one second in 10 years, they are sensitive to changes in temperature and acceleration and their resonating frequency can change as they age. Standard resonators are accurate to 6 parts per million at 31°C, which corresponds to ±½ second per day.

COMPENSATING FOR DRIFT

We can envision clock drift graphically by considering true (UTC) time flowing on the *x*-axis and the corresponding computer's clock reading on the *y*-axis. A perfectly accurate clock will exhibit a slope of one. A faster clock will create a slope greater than unity while a slower clock will create a slope less than unity. Suppose that we have a means of obtaining the true time. One easy (and frequently adopted) solution is to simply update the system time to the true time.
To complicate matters, one constraint that we'll impose is that it's not a good idea to set the clock back. The illusion of time moving backwards can confuse message ordering and software development environments.

LOGICAL CLOCKS

Let's again consider cases that involve assigning sequence numbers ("timestamps") to events upon which all cooperating processes can agree. What matters in these cases is not the time of day at which the event occurred but that all processes can agree on the *order* in which related events occur. Our interest is in getting event sequence numbers that make sense system-wide. These clocks are called *logical clocks*. If we can do this across all events in the system, we have something called *total ordering*: every event is assigned a unique timestamp (number), every such timestamp is unique. However, we don't always need total ordering. If processes do not interact then we don't care when their events occur. If we only care about assigning timestamps to related (causal) events then we have something known as partial ordering. Leslie Lamport developed a "happens before" notation to express the relationship between events: a→b means that a happens before b. If a represents the timestamp of a message sent and b is the timestamp of that message being received,

then a→b must be true; a message cannot be received before it is sent.
This relationship is transitive. If a→b and b→c then a→c. If a and b are events that take place in the same process the a→b is true if a occurs before b. The importance of measuring logical time is in assigning a time value to each event such that everyone will agree on the final order of events. That is, if a→b then clock(a) < clock(b) since the clock (our timestamp generator) must never run backwards. If a and b occur on different processes that do not exchange messages (even through third parties) then a→b is not true. These events are said to be concurrent: there is no way that a could have influenced b.

DELIVERY TIME DELAY

As mentioned in the previous section, the other major problem to be faced in WSN clock synchronization, is the random delivery time of messages. In particular, it is possible to decompose the total delivery time into different parts, as thoroughly analyzed:

- *Send Time*, *Ts*: time needed to read the local clock, assemble the message, and do the send-request to the MAC layer on the transmitter side. Depending on the system call overhead of the OS and on the current processor load, the send time is non deterministic and can be as high as hundreds of milliseconds.
- *Access Time*, *Ta*: waiting time to access the channel until transmission begins. It depends on the traffic on the radio channel and the backoff time of the CDMA protocol implementation. It varies from milliseconds up to seconds depending on the current network traffic.
- *Transmission time*, *Tt*: time necessary for the sender to transmit the message. This time is in the order of tens of milliseconds depending on the length of the message and the speed of the radio.
- *Propagation time*, *Tp*: travel time of a message from sender to receiver. The propagation time is highly deterministic and it depends only on the distance between the two nodes. This time is less than one microsecond for node distances under 300 meters.
- *Reception time*, *Trp*: time necessary for the receiver to receive the message. It is the same as the transmission time, i.e. *Trp* = *Tt*.
- *Receive time*, *Trv*: time required to process the incoming message and to notify the reception to the application. It is similar to the send time.

The total delivery delay, *Td* is then given by:
Td = *Ts* + *Ta* + *Tp* + *Trp* + *Trv*

A. Deterministic Clock Synchronization:
Algorithms Most clock synchronization algorithms proposed in the literature try to guarantee an upper bound on the clock skew with certainty. However, a theoretical limit derived by Lundelius and Lynch limits the maximum clock skew that these deterministic algorithms can guarantee. It is shown that the upper bound on the clock skew that can be deterministically guaranteed by any clock synchronization algorithm can be no smaller than (dmax — dmin)(1 +1/n). Here N is the number of nodes in the system, and dm and dmin, respectively, denote the maximum and minimum values of message delays in the system.

B. Probabilistic Clock Synchronization Algorithms:
The theoretical limit established constrains only those algorithms that provide a deterministic guarantee on the maximum clock skew. Clock skews that are significantly smaller than this theoretical limit can be achieved if we are willing to relax the requirement of determinism and accept a probabilistic guarantee. A guarantee is said to be probabilistic if it fails to hold sometimes, but with a failure probability that can be determined or bounded. A clock synchronization algorithm that provides a probabilistic guarantee on the maximum clock skew, is referred to as a probabilistic clock synchronization algorithm. Note that the word "probabilistic," as used here, connotes the uncertainty in the guarantee offered by the algorithm, rather than any randomness in the actions of the algorithm,

Cristian's Probabilistic Algorithm:
The idea of probabilistic clock synchronization was proposed by Cristian. Cristian also proposed the first probabilistic clock synchronization algorithm, referred to as CRI. Cristian's algorithm is based on a remote clock reading (RCR). RCR is used by a node to read the clock at a remote node with a specified minimum accuracy. RCR involves querying a target node for the time on its clock. The querying node then estimates the time on the target node's clock from the response received. RCR guarantees that the maximum estimation error is approximately D — dmin, where D is half the response time and dmin is the minimum response time. CRI is a master-slave algorithm that makes use of RCR to achieve synchronization. One node in the system is designated as the master, and the remaining nodes are designated as slaves. Each slave periodically resynchronizes with the master by estimating the reading on the master's clock by using RCR and adjusting its own clock accordingly. However, resynchronization is not guaranteed, because RCR may fail to achieve communication or understanding. The probability that algorithm CRI fails to resynchronize can be determined analytically. Thus, CRI is a probabilistic clock synchronization algorithm, and is not subject to the limit established. As a result, CRI can guarantee much smaller maximum clock skews than deterministic algorithms.

FAULT TOLERANCE

Tradeoff between fault tolerance and communication cost (m = 2, N = 256)

Array	Fmax	Remote clock reading per round (N*(N1*N2-2))
16*16	5	7680
8*32	10	9728
4*64	21	16896
2*128	42	32768
1*256	85	65280

Moreover, each of the a-clock readings gathered by *p* during that step originates from a node at a distinct (*m* - 1)th-step group. It may seem counter-intuitive that the fault tolerance of *m*-ICV depends only on the size of dimension *m* and is independent
of the other dimensions of the array. Thus, the number of faults tolerated can be increased simply by increasing the size of dimension *m*. However, increasing the degree of fault tolerance in this way does not come without cost. Increasing the size of dimension *m* while holding the number of nodes fixed forces either some dimensions to be eliminated or the numbers of nodes in other dimensions to be reduced.

CLOCKS AND REAL TIME

Real time and clock time quantities are, respectively, measured in seconds and SECONDS and are defined as follows:
DEFINITION 1. Time that is measured in an assumed Newtonian time frame (which is not directly observable) is referred to as real time and is denoted by t, u, or v.

DEFINITION 2. Time that is directly observable in a TSP's hardware clock is referred to as hardware clock time. We denote the value displayed by TSP p's hardware clock at real time t by Hp(t).

DEFINITION 3. TSP p's hardware clock is correct in a real-time interval [t1, t2] if, for all intervals [u1, u2] µ [t1, t2], the bounded drift condition holds:

|(Hp(u2) - Hp(u1)) - (u2 -u1)| . r (u2 -u1) + G,

where r and G are, respectively, the maximum drift rate and the granularity of a hardware clock.

Definition 3 states that a correct hardware clock measures the duration of a real-time interval [u1, u2] with an error of at most r (u2 - u1) + G. Hardware clocks are often implemented with a quartz oscillator and a counter, giving typical values of r on the order of 10^(-7) to 10^(-5). This technology can also produce high-resolution hardware clocks describes a clock synchronization unit for which G = 1 ms).

CLOCK TOPOLOGIES

The topology property of a clock defines how events are ordered in relation to each other and the interpretation of that ordering we can show how a clock topology can produce different event orderings from the same event set

- A linear graph ordering may be appropriate, if the topology does not characterize concurrency.
- A tree if the topology characterizes concurrency but not the synchronization between objects.
- A directed, acyclic graph if concurrency and synchronization between objects are characterized.

In all of these examples, the interpretation of the topology could follow that of potential causality: If a path exists between any two nodes, then the earlier node on the path may be a cause of the later node, otherwise the events may have occurred concurrently. The metrication property of a logical clock is a data structure containing counters (usually called timestamps), rules for advancing the counters, and rules for interpreting a timestamp to order the events or to identify if events may have occurred concurrently. The metrication must be consistent with the topology, so the topology expresses requirements on the metrication. For example, a vector time logical clock FL has a metrication of a vector of integers that can be used with a partial ordering relation to reconstruct a directed acyclic graph with a causal interpretation. Other metrication % for vector time have been proposed and are reported. A clock topology and metrication does not need to form a causal graph of events, although this is an intuitive characterization. For example, KF,hemkalyani exhaustively examined causal relationships between discrete and dense intervals of events, cataloguing 16 types of interactions between two intervals.

TOTAL ORDERING

Note that it is very possible for multiple non-causal (concurrent) events to share identical Lamport timestamps. This may cause confusion if multiple processes need to make a decision based on the timestamps of two events. The selection of a specific event may not matter if the events are concurrent but we want all the processes to be able to make the same decision. This is difficult if the timestamps are identical. Fortunately, there's an easy remedy. We can create a total order on events by further qualifying them with identities of processes. We define a global logical timestamp (Ti,i) where Ti represents the local Lamport timestamp and i represents the process ID (in some globally unique way: for example, a concatenation of host address and process ID).

CONCLUSION & FUTURE WORK

In this paper we have studied the various strategies of synchronization in Distributed system. In the domain of synchronous timeout approach are worth mentioning. The former is an asymmetric protocol and assumes, like us, an authenticated Byzantine model within a TMR system; further, it assumes every client to be a TMR system as well and solves the problem of message ordering together with majority voting of inputs. AMp was developed with commercial applications in mind, and provides the same message ordering guarantees as our protocol in a general n-processor system but assumes a benign fault model where processors either crash or occasionally omit to produce responses. Our assumption of authenticated Byzantine faults is weaker and, as argued, any further weakening of our fault model makes the desired form of message ordering impossible in a three-processor system. In the asynchronous model, the processing, the scheduling, and communication delays are only known to be finite, but their (upper) bounds cannot be known with certainty. Consequently, no deterministic message ordering protocol can be guaranteed to terminate even if one processor can crash [30]. This impossibility stems from the inherent difficulty in determining whether a remote processor has crashed or is only very slow. That is, since the asynchronous model permits any prior estimates of bounds to be violated, a fault-tolerant deterministic protocol cannot be guaranteed to terminate. It can only guarantee correctness without liveliness: If nonfaulty processes order a given message, they do so identically.

However, extensive work is still necessary to compare the performance of our proposed approach relative to FTSP and other protocols over large scale multi-hop sensor network and over longer periods. Moreover, some of the the parameters have not been optimized to cope with the fact that the clock skews change over time and that there are small measurement time delays We are currently analyzing these effects to compute estimates of the expected synchronization errors as a function of the number of nodes and the communication topology.

REFERENCES:

[1] C. Guillemot et al., "Transparent optical packet switching: the European ACTS KEOPS project approach," J. Lightwave Technol., vol. 16, pp.2117–2134, Dec. 1998.

[2] D. J. Blumenthal et al., "Optical signal processing for optical packet switching networks," IEEE Commun. Mag., vol. 41, pp. S23–S29, Feb. 2003.

[3] C. Su, L.-K. Chen, and K.-W. Cheung, "Theory of burst-mode receiver and its applications in optical multiaccess networks," J. Lightwave Technol., vol. 15, pp. 590–606, Apr. 1997.

[4] G. Georgiou et al., "Clock and data recovery IC for 40 Gb/s fiber optic receiver," IEEE J. Solid-State Circuits, vol. 37, pp. 1120–1125, Sept. 2002

[5] Y. Ota et al., "High-speed, burst-mode, packet-capable optical receiver and instantaneous clock recovery for optical bus operation," J. Lightwave Technol., vol. 12, pp. 325–331, Feb. 1994

[6] D. Wonglumsom, I. M. White, S. M. Gemelos, K. Shrikhande, and L. G. Kazovsky, "HORNET—a packet-switched WDM network: optical packet transmission and recovery," IEEE Photon. Technol. Lett., vol. 11, pp. 1692–1694, Dec. 1999.

[7] H. Nishizawa, Y. Yamada, K. Habara, and T. Ohyama, "Design of a 10-Gb/s burst-mode optical packet receiver module and its demonstration in a WDM optical switching network," J. Lightwave Technol., vol. 20, pp. 1078–1083, July 2002

[8] C. Bintjas et al., "Clock recovery circuit for optical packets," IEEE Photon. Technol. Lett., vol. 14, pp. 1363–1365, Sept. 2002.

[9] D. Chiaroni et al., "All-optical clock recovery from 10 Gbit/s asynchronous data packets," in Proc. Eur. Conf. Optical Communication, vol.4, 2000, pp. 69–70.

[10] B. Sartorius, C. Bornholdt, S. Bauer, and M. Mohrle, "40 GHz optical clock recovery for application in asynchronous networks," in Proc. Eur.Conf. Optical Communication, 2001, Paper We.P.32, pp. 442–443.

[11] K. L. Hall and K. A. Rauschenbach, "100 Gbit/s bitwise logic," Opt. Lett., vol. 23, pp. 1271–1273, Aug. 1998.

[12] R. J. Manning and G. Sherlock, "Recovery of a _ phase shift in _12.5 ps in a semiconductor laser amplifier," Electron. Lett., vol. 31, no. 4, pp. 307–308, 1995.

[13] C. Bornholdt, J. Slovak, M. Moehrle, and B. Sartorius, "Application of 80 GHz all-optical clock in a 160 km transmission experiment," in Proc. Optical Fiber Communication Conf., 2002, pp. 87–89.

[14] T. Akiyama et al., "Nonlinear gain dynamics in quantum-dot optical amplifiers and its application to optical communication devices," IEEE J. Quantum Electron., vol. 37, pp. 1059–1065, Aug. 2001

[15] M. Usami et al., "Mechanism for reducing recovery time of optical nonlinearity in semiconductor laser amplifier," Appl. Phys. Lett., vol. 72, no. 21, pp. 2657–2659, 1998.

[16] G. T. Kanellos, L. Stampoulidis, N. Pleros, T. Houbavlis, D. Tsiokos, E. Kehayas, H. Avramopoulos, Member, IEEE, and G. Guekos, Member, IEEE Clock and Data Recovery Circuit for 10-Gb/s Asynchronous Optical Packets

[17] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," J. ACM, vol. 27, no. 2, pp. 228-234, Apr. 1980.

[18] D. Powell, P. Verissimo, G. Bonn, F. Waeselynck, and D. Seaton, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems," Digest of Papers, FTCS-18, Tokyo, pp. 246- 251, June 1988.

[19] F.B. Schneider, "Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial," ACM Computing Surveys, vol. 22, no. 4, pp. 299-319, Dec. 1990.

[20] L. Lamport, "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," ACM Trans. Programming Languages and Systems, vol. 6, no. 2, pp. 254-280, Apr. 1984.

[21] N. Vasanthavada and P.N. Marinos, "Synchronisation of Fault- Tolerant Clocks in the Presence of Malicious Failures," IEEE Trans. Computers, vol. 37, no. 4, pp. 440-448, Apr. 1988.

[22] P. Verissimo, L. Rodrigues, and A. Casimoro, "Cesium Spray: A Precise and Accurate Global Clock Service of Large Scale Systems," J. Real Time Systems, vol. 11, no. 3, 1997.

[23] D. Dolev, J. Halpern, and H.R. Strong, "On the Possibility and Impossibility of Achieving Clock Synchronisation," Proc. 16th Ann. ACM STOC, pp. 504-511, Apr. 1984.

[24] F. Schmuck and F. Cristian, "Continuous Clock Amortization Need Not Affect the Precision of a Clock Synchronisation Algorithm," Proc. Ninth ACM Symp. Principles of Distributed Computing, pp. 133-141, Aug. 1990.

[25] K. Echtle, "Fault Masking and Sequence Agreement by a Voting Protocol with Low Message Number," Proc. Sixth Symp. Reliability in Distributed Software and Database Systems, pp. 149-160, Mar. 1987.

[26] P. Verissimo, L. Rodrigues, and J. Rufino, "The Atomic Multicast Protocol (AMp)," Delta-4: A Generic Architecture for Dependable Distributed Computing, D. Powell, ed., pp. 267-294, ESPRIT Research Papers, Springer-Verlag, 1991.

[27] P. Verissimo, "Causal Delivery Protocols in Real-Time Systems: A Generic Model," J. Real Time Systems, vol. 10, no. 1, pp. 45-73, 1996.

[28] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," Comm. ACM, vol. 31, no. 2, pp. 120-126, Feb. 1978.

[29] P.D. Ezhilchelvan, "Early Stopping Algorithms for Distributed Agreement under Fail-Stop, Omission, and Timing Fault Types," Proc. Sixth Symp. Reliability in Distributed Software and Database Systems, pp. 201-212, Mar. 1987.

[30] D. Dolev, R. Reischuk, and H.R. Strong, "Early Stopping in Byzantine Agreement," J. ACM, vol. 37, no. 4, pp. 720-741, Oct. 1990.

[31]Paul D. Ezhilchelvan, Francisco V. Brasileiro, Member, IEEE Computer Society, and Neil A. Speirs A Timeout-Based Message Ordering Protocolfor a Lightweight Software Implementation of TMR Systems

[32]Paul Krzyzanowski- Lectures on distributed systems Clock Synchronization

[33]Luca Schenato, Giovanni Gamba A distributed consensus protocol for clock synchronization in wireless sensor network

[34]Bong Jun Choi, Student Member, IEEE, Hao Liang, Student Member, IEEE, Xuemin (Sherman) Shen, Fellow, IEEE, and Weihua Zhuang, Fellow, IEEE DCS: Distributed Asynchronous Clock Synchronization in Delay Tolerant Networks.