the Availability Digest

Time Synchronization for Distributed Systems – Part 1

November 2007

In many distributed systems, it is imperative that each node march to the same time. In fact, this requirement is often extended to include client systems that access nodes in the distributed system.

Time synchronization is a complex problem across distributed systems. Each node in a distributed system necessarily has its own clock. Communication between nodes can take tens of milliseconds and, especially over the Internet, can be quite variable and unpredictable. Consequently, tight (e.g., submillisecond) time synchronization cannot be achieved simply by having one master timekeeping server send timing messages to the other nodes in the network.

In this series of articles, we discuss current technologies for maintaining time synchronization across the network. The most common approach today is NTP (Network Time Protocol), used extensively over the Internet. However, some synchronization problems go beyond the scope of NTP and require additional technologies.

In Part 1, we discuss the basics of how NTP adjusts time between systems. Part 2 discusses additional NTP features that make these adjustments more precise. Part 3 discusses a totally different approach, called Lamport logical clocks, which has the potential of virtually eliminating data collisions in active/active systems without reverting to synchronous replication.

A Brief History of Time – The Leap Second

In the words of the brilliant physicist Professor Stephen Hawking, let us first look at the evolution of timekeeping and its impact on computer time synchronization, or *computer network chronometry.*

Our civil time is based on three astronomical rotations – the earth about its axis (which takes one day), the moon about the earth (which takes one month), and the earth about the sun (which takes one year). For the last 4,000 years and until recently, timekeeping has been the realm of astronomers.

The day was measured by the amount of time between two *transits of the sun*, or when the sun reached its highest point in the sky. This time was broken up into 24 hours and an hour into 3600 seconds.

Early astronomers determined that a year was 365 days. However, by Julius Caesar's time, the calendar year had slipped by eight weeks relative to the solar year. As a consequence, the Julian calendar, adopted in 46 BC, added an extra day every fourth year (the leap year).

Still, by 1545, the discrepancy of the Julian year had crept to ten days. As a result, Pope Gregory XIII issued a dictum that a solar year was 365.2422 days, leading to the skipping of the leap year every 400 years. This became the Gregorian calendar in use today.

The saga continues. In 1940, it was discovered that the earth's rotation is slowing down, probably due to tidal friction and atmospheric drag. In fact, geologists now believe that 300 million years ago, there were 400 days in a year. As a consequence, the duration of a solar second is slowly changing. With the invention of the atomic clock in 1948, it became possible to measure time more accurately and independently from the wiggling and wobbling of the earth. At this point, physicists took over the job of timekeeping from the astronomers. They redefined a second as being 9,192,631,770 transitions of the cesium atom. This gives a time resolution of about a tenth of a picosecond. Periodically, laboratories around the world tell the Bureau International de l'Heure (BIH) in Paris how many times their cesium atomic clocks have ticked. The BIH averages these and produces the International Atomic Time (TAI). TAI is the number of cesium transitions since January 1, 1958.

However, this is not quite accurate. A TAI day is about 3 msec. short, and the day is getting longer all the time. To correct this problem, BIH introduces a leap second whenever the discrepancy between TAI and the mean solar day grows to 800 msec. This gives rise to a time system that keeps in phase with the sun and is called Universal Coordinated Time, or UTC. It replaces the old standard, Greenwich Mean Time (GMT), which is solar time.

The importance of all of this on computer network chronometry is that leap seconds have to be added to whatever primary timing source we are using. In some cases, this is done by the primary time source. In other cases, we have to do it.

Primary Reference Clocks

A *primary reference clock* is a clock that keeps TAI or UTC time. There are several types of these clocks available for use in synchronizing computer networks.

Atomic Clocks

A system could use a local cesium atomic clock. However, this is very expensive and not typically done. If an atomic clock is used, the user must add leap seconds.

Radio

Various governments broadcast UTC over radio channels. In the U.S., these channels include WWV and WWVB in Boulder, Colorado, and WWVH in Hawaii.

Computers using these channels as a source must estimate the communication latency between the radio station and their computer site, and add that latency to the broadcast time in order to arrive at the real time.

WWV broadcasts at high frequencies from 2.5 to 20 megahertz. It propagates by bouncing a signal off of the upper ionosphere, and it is therefore receivable over most of the western hemisphere. However, since the height of the upper ionosphere changes during the day, the latency is quite variable, and leads to significant jitter in the calculated time. Accuracies of one to ten milliseconds are achievable.

WWVB broadcasts on a much lower frequency – 60 kilohertz. Its signal bounces off of the lower atmosphere. Therefore, it is much more stable (accuracies within 50 microseconds can be achieved) but is receivable over a much smaller area (the continental United States).

Radio time receivers are available from Spectracom, Tranconex, Kinemetrics, and others and cost in the order of a few thousand dollars.

Loran

The U.S. Coast Guard and agencies in other nations provide Loran service for navigational purposes. Loran provides time accuracies of about a microsecond within a few hundred kilometers of a ground station due to its use of ground waves. Beyond that, Loran depends upon the lower ionosphere and provides accuracies within 50 microseconds.

Satellite

More recently, the GOES (Geosynchronous Orbit Environment Satellite) and the GPS (Global Positioning Satellite) systems provide timing information accurate to within a few hundred microseconds. GPS timing promises to lead to less expensive receivers. After all, GPS technology has now reached the cell phone.

Time Synchronization with NTP

The Network Time Protocol (see www.ntp.org) is one of the oldest protocols on the Internet. Developed by David Mills at the University of Delaware as an open source offering, it is maintained by him and a staff of volunteers.[1]

NTP is a protocol for synchronizing computer system clocks over packet-switched variable latency networks. It is designed particularly to resist the effects of variable channel latency, or jitter. It provides the mechanisms for a distributed subnet of time servers to operate in a self-organizing, hierarchical master/slave configuration and to synchronize the local computer clocks within the subnet to national time standards. In practice, it achieves timing accuracy of better than ten milliseconds over public networks and 200 microseconds over LANs.

Version 0 of NTP went into service in 1985. Version 4 is now in use, and version 5 is being specified. Version 3 is very well documented via RFC 1305,[2] but the version 4 documentation is still in progress.

Determining and Correcting Timing Errors

Timestamps

NTP reports time via an NTP timestamp. This timestamp is 64 bits long. The high-order 32 bits represent the number of seconds since January 1, 1900, and the low-order bits represent the fraction of a second (a resolution of about 200 picoseconds).

The 32-bit length for the seconds field presents a rollover problem. NTP clocks will roll back to 1900 in 2038, reminiscent of the Unix 2036 problem. To correct this, future versions of NTP plan to use a 128-bit timestamp, 64 bits for seconds and 64 bits for the fraction of a second. The 64-bit seconds field will take us out until after the universe goes dark – 585 billion years. According to David Mills, the 64-bit fractional field is enough to resolve the time that it takes a photon to pass an electron. 128 bits should do us for a while.

Offset and Skew

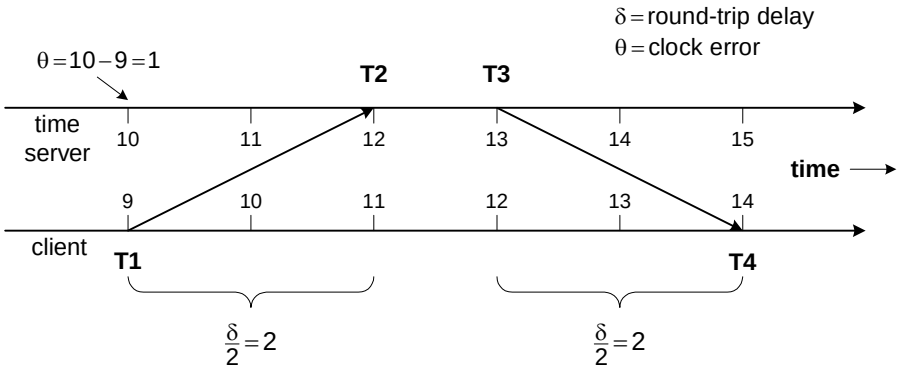
NTP provides the facilities to compare a client clock to a time-server clock and to determine the *offset* of the client clock relative to the time-server clock. The frequency difference, or *skew*, between the clocks can also be determined. Knowing the offset, the client clock can be adjusted

[1] Mills, D., A Brief History of NTP Time: Confessions of an Internet Timekeeper, http://www.cis.udel.edu/~mills/database/papers/history.pdf

[2] http://www.eecis.udel.edu/~mills/database/rfc/rfc1305/

to agree with the time-server clock. If the client clock is using a clock the frequency of which can be controlled (such as a voltage-controlled oscillator), its clock frequency can be adjusted by knowing the skew.

The offset is simply determined by the client sending a timestamped message to the time server and by the time server returning that message with its own timestamps, as shown in Figure 1. I apologize for the use of Greek symbols (which I usually try to avoid), but I do use them here to be consistent with the NTP documentation.



Measuring Clock Error
Figure 1

Let

δ (delta) be the round-trip delay time of the communication channel.
θ (theta) be the time offset of the client clock from the server clock.

More specifically, θ is

θ = time-server clock time – client clock time

In Figure 1, the client system timeline is shown on the bottom with its clock ticks, and the time server system is shown at the top with its clock ticks. In this example, the client is slow relative to the server by one clock tick. When the server thinks that the time is 10, the client thinks that it is only 9. Thus, θ = +1. The question is how can the client determine that it is slow by one tick?

The client can determine the offset of its clock by sending the server a message with a timestamp T1. When the server receives it, it immediately adds its current time as timestamp T2. When it is ready to reply to the message, the server adds a second timestamp T3. The client then receives the response message at time T4, according to its local clock.

From the above figure, the channel latency, δ, which is the round-trip time for sending a message, is

$$\delta = (T4 - T1) - (T3 - T2) \qquad (1)$$

In the above example, the channel latency is [(14 – 9) – (13 – 12)] = 4 clock ticks.

Assuming that the channel is symmetric - that is, the communication latency is the same in both directions (more about this later), T2 will be equal to T1 plus half of the communication latency (the time that it took the message to get from the client to the server) plus the time offset θ:

$$T2 = T1 + \frac{\delta}{2} + \theta \qquad (2)$$

Likewise, T4 will be equal to T3 plus half of the channel latency (for the time that it takes the message to get from the server to the client) minus the time offset θ:

$$T4 = T3 + \frac{\delta}{2} - \theta \qquad (3)$$

Using Equation (1) for δ and solving either equation (2) or (3) for θ yields

$$\theta = \frac{(T2 - T1) - (T4 - T3)}{2} \qquad (4)$$

Thus, the client's clock offset is easily determined from the timestamps in the message that it sent the server. In the above example, θ = [(12 – 9) – (14 – 13)] / 2 = 1 clock tick, as expected.

Clock Adjustment

Once the client's clock offset is known, the client's clock can be adjusted to be equal to the server clock. There are a couple of problems that must be faced, however. First of all, time cannot go backwards. Therefore, if θ is negative (the client's clock is fast) and if the client's clock must be set back, the clock must be adjusted over a (short) period of time by slowing it down, typically by extending the tick interval or by skipping an occasional tick. Likewise, if the clock must be advanced by a large amount (typically more than 128 milliseconds), this may be done over an extended time interval. These procedures are called *clock slewing*.

However, if the clock offset is very large (typically, minutes), a casual clock adjustment will not be made. The clock is simply reset (if it has to be reset backwards, one has to accept the consequences). This might occur, for instance, if the time server is offline for an extended period of time.

In addition, there most likely will be "jitter" associated with channel latency. That is, the time to get a message across the channel is not absolutely constant and will vary somewhat with each transmission. This is especially true when communication is over the Internet, where congestion can build and where rerouting may occur. Therefore, the calculation of clock offset, θ, may be in error; and subsequent measurements may not agree. (Software will also cause jitter; but with today's systems, this jitter is typically measured in microseconds, whereas channel jitter can be in the order of milliseconds).

Jitter is compensated for by taking many measurements and by averaging the clock offset over these samples. For instance, clock offset may be averaged over the last eight measurements. The last measurements are stored in a buffer. When a new measurement is made, the oldest measurement is dropped from the buffer, the new measurement is added, and the average of these eight measurements becomes the latest estimate of the clock offset.

Finally, the derivation of θ above assumed that the channel is symmetric – that is, the time that it takes to send a message from the client to the time server is the same as the time that it takes for the time server to send a message to the client. This will probably not be the case over the Internet since the request message's route might be quite different from the response message's route. There is also the problem that the time used by the time server itself may be in error. The results of this and other similar problems are improved by taking time measurements over multiple time servers and averaging them. This *clock combining* strategy is described in more detail in Part 2.

Frequency Skew

So far, we have calculated the time offset of the client's clock. If there is a consistent error over a sequence of measurements, it can be deduced that the client's clock is running a little fast or slow with respect to the server's clock. This is called *frequency skew*.

With frequent clock adjustments, frequency skew may not pose a problem. However, if the time server should be lost for an extended period of time, the time difference due to frequency skew may be so great once the time server is again available that a forced reset of the clock is required. This is not a desirable action to take.

The frequency of some clocks can be adjusted. An example is a clock that is driven by a voltage-controlled oscillator (VCO). A VCO is an oscillator the frequency of which can be controlled by a control voltage. If the frequency skew can be determined, a feedback circuit can be used to adjust the VCO's voltage to bring the oscillator's frequency more into line with the time server. In this way, the client's clock can provide a fairly accurate clock over an extended period of time even if the time server is down. (Of course, if clock time is determined by averaging the time offsets from multiple time servers as suggested above and as described in more detail in Part 2, the loss of one time server is not such a problem.)

Frequency skew is easily calculated based on the measured offset and on the measurement interval. If the measurement interval is T_m, the frequency skew is

$$\text{Frequency skew} = \theta / T_m \qquad (5)$$

For instance, if the measurement interval, T_m, is 5 minutes (300 seconds), and if the clock offset is measured to be 100 microseconds, the frequency skew is $10^{-4} / 300 = 3.33 \times 10^{-5}$, or .00333%. If the offset is positive (i.e., the client clock is running slow), the VCO must be sped up by .0033%. If the offset is negative, the client clock must be slowed down. This adjustment would be calculated on each time measurement and perhaps averaged over several measurements so that the client clock rate can be synchronized with the time server's clock rate as closely as possible.

What's Next?

In this Part 1 of *Time Synchronization for Distributed Systems*, we have talked about what is real civil time today and what are the basics of keeping a client system in time synchronization with a time server using NTP. We have noted that synchronizing to a single time server may not provide highly precise times because of asymmetric channel latency, jitter, time-server errors, and other problems. We discuss in Part 2 NTP's algorithms for combining the readings from multiple time servers and for synchronizing these times with real civil time.

In Part 3, we leave NTP and move to an alternate algorithm proposed in a seminal paper by Leslie Lamport for synchronizing clocks in a distributed system so that all are the same but are not necessarily tied to a time standard. This is a simpler algorithm than used by NTP. In addition, it leads to a powerful way to virtually eliminate data collisions in active/active systems without going to synchronous replication.