orolia ENTERPRISE

The Advent of Tightly Synchronized Clocks in Distributed Systems

Murat Demirbas

May 1, 2018

1. Time in distributed systems

The last decade has brought a paradigm shift in computing systems from single processor devices — whose performance plateaued — to distributed computing systems. In distributed systems, nodes execute concurrently with limited information about what the other nodes are executing at the moment. Thus, a fundamental problem in distributed systems is to coordinate the execution of these independent nodes effectively to achieve performance while preserving correctness.

Unfortunately, distributed coordination is a notoriously tricky problem. Since the nodes do not have access to a shared state and a common clock, they need to communicate and synchronize state frequently in order to coordinate on the order of events. However, excessive communication hinders the performance/scalability of the distributed system. Due to this inherent tradeoff, distributed systems designers need to walk a tight rope.

A key component of distributed coordination is the enforcement of consistent views at all nodes for the ordering of significant events. To this end, events are "timestamped" with logical counters or — increasingly lately — with tightly-synchronized physical time.

To explain the concepts of counters, timestamps, causality, and happened-before, we first look at a brief history of time in distributed systems in this section. We describe the rise of better synchronized clocks in Section 2, and applications of these tightly synchronized clocks in distributed systems in Section 3.

1.1 Logical clocks

Logical clocks (LC) [11] were proposed in 1978 by Leslie Lamport as a way of timestamping and ordering events in an *asynchronous* distributed system. LC are entirely divorced from wallclock/physical clocks and real time: the nodes do not have access to clocks, there is no bound on message delay, and on the speed/rate of processing of nodes.

The causality relationship LC captured is called happened-before (**hb**), and is defined based on *passing of information*, rather than passing of time. Event A happened-before event B, if A and B are on the same node and A comes earlier than B, or A is a send event and B is the corresponding receive event, or happened-before is defined transitively based on the previous two.

LC has some drawbacks. Using LC, it is not possible to query events in relation to real time. Moreover, if two events do not have any happened-before causal information flow between them, LC cannot order them and declares them to be concurrent even when the two events are apart by several minutes in real time. Finally, for capturing **hb**, LC assume all communication occur using LC timestamped messages and there are no backchannels in the system, however, this is hard to enforce for today's large scale heterogeneous system of systems.

1.2 Loosely synchronized clocks

Network Time Protocol (NTP) [14] is a clock synchronization protocol between computer systems across the Internet. In 1988, 10 years after the logical clocks paper, NTPv1 was developed. NTP works by estimating the time that messages take to travel between hosts, so that a host can account for this travel time when adjusting its clock to match a more authoritative source with more precise clock. However, a big source of error for NTP is the asymmetry in the links, where there is no delay one way, but a lot of latency and queuing delay on the other way. Due to the multihop and often asymmetric bandwidth communication in Internet, it is hard to tightly bound NTP accuracy. NTP can usually maintain time to within tens of milliseconds over the public Internet, but occasional periods of hundreds of milliseconds errors are possible due to communication failures or cold starts.

Due to the uncertainty in NTP synchronization, it is impossible to compare/order events that have close timestamps. Thus, NTP timestamps may violate the happened before relation: even though event E happened-before (i.e., causally precedes) event F, NTP clock on the node that F occurs may assign F a smaller NTP timestamp than that of E's. It is also possible that NTP clocks violate monotonicity even for timestamping on the same node due to leap seconds and discrete jumps in clocks to correct large time differentials.

Other problems that NTP clocks face include the following. Systems clocks may move at different speeds due to hardware problems. Virtualization can wreak havoc on kernel timekeeping. Upstream

NTP servers can lie. NTP is also prone to security attacks. Having your timeservers is good for increased security against NTP attacks [13]. Finance sector does not use NIST public source NTP servers since men-in-the-middle attack is possible.

Despite these problems, several distributed systems/applications employed NTP in ad hoc undisciplined ways and this led to serious problems. One example of this is any distributed database with a "last write wins" policy for resolving conflicting writes [9].

2 The rise of better synchronized clocks

The potential benefits of using synchronized clocks in distributed systems have been recognized as early as 1991: Barbara Liskov's "Practical uses of synchronized clocks in distributed systems" paper [12] made a strong case for the use of synchronized clocks in improving efficiency/performance of distributed systems particularly in the context of concurrency control. On the other hand, in the following decades, there was not much progress in the adoption of synchronized clocks in distributed systems —except for the straightforward use of local timers for implementing simple lease primitives. A major reason for this is likely the loose-synchrony and problems with NTP clocks.

In recent years we see a strong trend toward tight clock synchronization in datacenters and find that several distributed systems embrace synchronized clocks. To support tight clock synchronization guarantees for Google Spanner [3], Google datacenters have been deploying atomic clocks and GPS clocks. Recently, Amazon Web Services (AWS) announced Amazon Time Sync service [19], "a time synchronization service delivered over Network Time Protocol (NTP) which uses a fleet of redundant satellite-connected and atomic clocks in each region to deliver a highly accurate reference clock."

Here we discuss some key contributors to realizing stronger clock synchronization recently.

2.1 Better clocks

Recently, with the advent of technology, atomic clocks got much cheaper and more common. Atomic clocks often use Rubidium, which has a drift of only 1 microsecond a day. OCXO ovenized oscillators provide the next best way to have precise clocks in a box: Temperature change has the most effect in crystal oscillation rate, and ovenizing the oscillators provides a way to control/compensate temperature change limiting the clock drift to 25 microsecond a day.

2.2 Better communication links

The key to better clock synchronization is to make the communication latencies deterministic so that they can be factored out. Using GPS [6], although the communication is done with a satellite ~12,500 miles miles away, since it is singlehop, it is possible to precisely determine the flight time of the signal and remove that for performing clock synchronization. Because of this reason, GPS synchronization is much more precise than NTP synchronization, and is able to achieve up to ~10 nanosecond precision in synchronization.

To reduce the latency and unpredictability of communication links, datacenter communication networks have been getting frequent upgrades. As a result, the nondeterminism introduced at the routers/switches inside the datacenters are getting smaller. To make the clock synchronization even more precise, it is possible to remove the nondeterministic communication delay further by putting the timestamp at the sending of the message below MAC layer rather than above the MAC layer. The Precision Time Protocol (PTP) [4] enables hardware timestamping and newer implementations of NTP introduced support for low-level timestamping.

PTP also incorporates measures to eliminate link delay asymmetry. The time provider sends timestamps to the client during a 3-way handshake, so the client can calculate the in-flight-time delays between the time-server and itself more precisely. In contrast, in NTP, the time provider is oblivious to the client and is stateless with respect to the client.

2.3 Better APIs

Google Spanner paper [3] introduced the Google TrueTime (TT) service and API. TrueTime is a global synchronized clock with bounded non-zero error, and the TT API captures and exposes this synchronization uncertainty interval of the clock to the users of the service. TT.now() returns a TTinterval that is guaranteed to contain the absolute time during which TT.now() was invoked. If two intervals do not overlap, then we know calls were definitely ordered in real time. If the intervals overlap, we do not know the actual order.

Another example of a new synchronized clock API is hybrid logical clocks (HLC) [10] that integrate the general techniques of logical time with synchronized time of physical clocks. HLC utilizes the synchronized physical clocks of individual processes and tracks causality information only in the uncertainty windows of the synchronized physical clocks. Like TrueTime, HLC does not pretend to create a single unified timeline [17] but allows the system designer to build on the best available knowledge of time as perceived and communicated across a cluster. Moreover, HLC provides survivability and resilience to time-synchronization errors and can make progress, and capture causality information even when time synchronization has degraded or is not available.

3 Applications of synchronized clocks in distributed systems

With the availability of more tightly synchronized clocks, we saw an increase in the use of clocks in distributed systems applications. Here we talk about them under two categories: distributed coordination applications and distributed monitoring applications.

3.1 Distributed coordination

Distributed coordination plays an important role in datacenter and cloud computing, particularly in leader election, group membership, cluster management, service discovery, resource/access management, and consistent replication of the master nodes in services. More specifically, distributed coordination finds applications in building large-scale cloud native databases and web services, which need to provide consistency guarantees to the face of server and network failures.

Tightly synchronized clocks enable nodes to operate approximately on a common time axis making it possible to tradeoff explicit communication with passing of time for achieving coordination.

In finance and e-commerce, synchronized clocks are used for determining transaction order. Similarly, in distributed databases, tightly synchronized clocks allow a database to enforce external consistency. In Spanner, in order to achieve external consistency, a write-transaction waits out the clock uncertainty period before releasing locks on the relevant records and committing. Better clock synchronization means shorter uncertainty periods to wait out.

Tightly synchronized clocks also benefit snapshot reads (i.e., reads in the past). By just giving a time in the past, the snapshot read can get a consistent cut read of all the variables requested at that given time. This is not an easy feat to accomplish in a distributed system without using tightly synchronized clocks, as it would require capturing and recording causality relationships across many different versions of the variables involved so that a consistent cut can be identified for all the variables requested in the snapshot read.

CockroachDB [2] also uses synchronized clocks, but in the absence of access to atomic clocks in private datacenters as Google does, CockroachDB uses loosely synchronized clocks and HLC to achieve consistent snapshots. Using tightly synchronized clocks for consistent snapshot reads across many nodes in a distributed database prevents the need for the nodes to communicate/coordinate with each other and leads to improved throughput and latency [5].

Finally, on the analytics side, there has been renewed interest in timeseries databases. Recent popular timeseries databases include QuasarDB [15], InfluxDB [8], and TimescaleDB [18].

3.2 Monitoring of distributed systems

In distributed systems, scalability and availability are critical challenges. For developing highly scalable and highly available distributed systems it is important to improve auditability, which enables identifying performance bottlenecks, dependencies among events, and latent concurrency bugs. In distributed systems, many problems are hard to solve with local information, but becomes easy if we can peek at the global state and provide centralized oversight/override.

However, peeking at the global state snapshot of a distributed system is not easy. You need to construct this snapshot in a consistent/coherent way from the logs generated locally at the nodes participating in the distributed system. By using HLC, in our recent work, we developed the Retroscope monitoring framework [1] to help align these logs and perform searches on those sorted logs. In other words, Retroscope captures a progression of globally consistent distributed states of a system and enables the user to examine these states and search for global predicates. While request tracers inspect the system by following the trace of a request, Retroscope performs cut monitoring and examines the system at consistent global cuts, observing the state across many machines and requests.

Another recent monitoring framework that leverages synchronized clocks at the nodes is the Strymon. Similar to Retroscope, Strymon [7] employs cut monitoring and extends that to support online critical path analysis of modern dataflow systems like Spark, Flink and TensorFlow.

There are other significant applications of synchronized clocks in distributed event stream processing and querying systems, including SCADA systems [16]. Recently synchronized clocks have been used for monitoring of electric grid by taking globally consistent snapshots using HLC.

4 Concluding remarks

Tightly synchronized clocks provide a tremendous help for enforcement of consistent views at all nodes for the ordering of significant events. This functionality is getting increasingly more leveraged for building distributed consensus/coordination and distributed monitoring solutions — two significant problems in distributed big data processing systems. We provided some examples of these applications in this report, and we expect even more applications to emerge in the near future as tight clock synchronization and related techniques get prevalent.

Going forward, we need to build principled algorithmic basis and frameworks to enable effective use of tightly synchronized clocks in distributed systems. An important principle for design of distributed protocols is to separate safety and progress in reasoning. Thus, even with improved time synchronization infrastructure, it is best practice to design distributed protocols defensively so that they do not rely on clock synchronization "guarantees" to always hold —because under extreme

conditions or adversarial attacks, those assumptions will inevitably be violated. This way even under the rare conditions when timing assumptions are violated, the distributed system can still satisfy its safety/correctness contract with its users. On the other hand, it is perfectly OK and desirable to depend on timing assumptions for providing better progress and performance properties. With better clock synchronization such distributed systems will be able to deliver better progress and performance results to the users.

References

[1] A. Charapko, A. Ailijiang, M. Demirbas, and S. Kulkarni. Retrospective lightweight distributed snapshots using loosely synchronized clocks. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2061–2066. IEEE, 2017.

[2] Cockroachdb, the sql database for global cloud services. https://www. cockroachlabs.com. Accessed on April 25, 2018.

[3] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[4] J. Eidson and K. Lee. Ieee 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In *Sensors for Industry Conference, 2002. 2nd ISA/IEEE*, pages 98–105. Ieee, 2002.

[5] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18),* pages 81–94, 2018.

[6] I. A. Getting. Perspective/navigation-the global positioning system. *IEEE spectrum*, 30(12):36–38, 1993.

[7] M. Hoffmann, A. Lattuada, J. Liagouris, V. Kalavri, D. Dimitrova, S. Wicki, Z. Chothia, and T. Roscoe. Snailtrail: Generalizing critical paths for online analysis of distributed dataflows. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 95–110, Renton, WA, 2018.

[8] Influxdb,the complete time series platform. https://www.influxdata.com. Accessed on April 25, 2018.

[9] K. Kingsbury. The trouble with timestamps. https://aphyr.com/ posts/299-the-trouble-with-timestamps. Accessed on April 25, 2018.

[10] S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone. Logical physical clocks. In *Principles of Distributed Systems*, pages 17–32. Springer, 2014.

[11] L. Lamport. Time, clocks, and the ordering of events in a distributed system *Communications of the ACM*, 21(7):558–565, July 1978.

[12] B. Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6(4):211–219, 1993.

[13] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the network time protocol. In NDSS, 2016.

[14] D. Mills. A brief history of ntp time: Memoirs of an internet timekeeper. *ACM SIGCOMM Computer Communication Review,* 33(2):9–21, 2003.

[15] Quasardb, time series. at scale. https://www.quasardb.net. Accessed on April 25, 2018.

[16] Supervisory control and data acquisition (scada). http://scada.com. Accessed on April 25, 2018.

[17] J. Sheehy. There is no now. ACM Queue, 13(3):20, 2015.

[18] Timescaledb, sql made scalable. https://www.timescale.com. Accessed on April 25, 2018.

[19] Amazon time sync service. https://aws.amazon.com/about-aws /whats-new/2017/11/introducing-the-amazon-time-sync-service/. Accessed on April 25, 2018.