Models of Time for Safety Critical Systems

Partial vs. Total Order – Polychronous vs. Synchronous

Sandeep K. Shukla
Electrical and Computer Engineering Department
Virginia Tech
Blacksburg, VA 24061
Email: shukla@vt.edu

In safety critical systems, accuracy of time is important if the activities of the system are time driven, and the notion of synchrony is based on the accuracy of time. For example, under a GPS based timing, the nodes of a distributed system may agree on the current time to sub-microseconds accuracy and the system can span a wide geographical region – whereas, a system with IEEE 1588 based time synchronization, the same accuracy of synchronization can be obtained but is spatially limited to a few subnets.

The synchronization of clocks for such systems and protocols come at the cost of an overhead of running expensive synchronization algorithms. However, in recent times, systems dependent on precise clock synchronization are vulnerable to cyber attacks. GPS spoofing, malicious disruption of IEEE 1588 based timing by message spoofing, and other kinds of intermediary based attacks etc., can easily make the system nodes perceive time wildly differently. Therefore, even if we are ready to bear the cost of time synchronization, we cannot afford to render our safety-critical systems so tightly dependent on timing accuracy without adequate cyber security overheads.

Time-triggered systems that depend on accurate agreement of time are not necessarily appropriate when cyber defenses are weak, or impractical. For example, sensor networks in a war zone, or cyber physical systems such as a nuclear power plant, can be target of timing disruption based attacks. The recent incidents with the Stuxnet worm showed how easily it is possible to breach the security of such systems. It is also easily conceivable that attacks on time synchronization mechanism in such systems can run havoc on such systems.

In the context of embedded real-time systems this problem is not a new one, and models of time have been a subject of intense research in the past two decades. A distributed system endowed with strong clock synchronization mechanisms can view time as an approximately totally ordered (possibly infinite) set of instants. As long as the required time resolution is within the accuracy provided by the synchronization mechanism, this could work. In absence of clock drifting, poorly designed clock synchronization protocols, or cyber attacks on clock synchronization, distributed algorithms and protocols can be either time driven, or can be made simpler by assuming a linear or total order of time. Proving correctness is also simpler (not necessarily trivial).

Unfortunately, as argued earlier, assumptions of uncompromised clock synchronization, no fault in the system nodes or links etc, are often not valid. Thus a totally ordered time is also not necessarily a safe assumption for safety-critical system designs.

On the other hand, today's systems with multi-tasking, multi-threading, and multi-core precepts have naturally multiple notions of time. For example, two threads T_1 and T_2 may run at different speeds, and may run in parallel intermittently synchronizing on various events. The time inside each thread could be viewed as totally ordered, but since the events happening in one thread could arbitrarily concur against events in the other, except at the thread synchronization points, the time in the other thread is not necessarily the same total order as the first one.

This gives us the notion of *polychronous time*. Thus time is no longer unique or the same at all concurrently running entities, but they have their own totally ordered instants as their local times, and the instants in these local time sets are partially ordered with respect the instants in another thread's local time.

The origin of this partial order can be easily understood with reference to Lamport's famous paper titled "*Time*, *Clocks*, and the *Ordering* of Events in a Distributed System". Since the threads interact via message passing or synchronizing on shared resources, the two local times need to have some notion of ordering at certain instants, but for most part the two threads run asynchronous to each other until such points where such exchanges or synchronizations take place. Upon finishing such required interactions, they again take asynchronous paths until the next synchronization instants arrive.

In a simple minded distributed system, one could schedule a step by step synchronous execution of the threads, so that they are always in lock step. Such a system preserves a total order of time across all the local times. In other words, this is the simplest way to create a single notion of a totally ordered global time. But this amounts to barrier synchronization at every step – which degrades performance unnecessarily. Also, this requires a global coordinator – which is akin to global clock synchronization, and suffers from the same vulnerabilities and overheads.

Thus polychronous notion of time with its partial order over the union of all local times, frees us from the requirements of strong clock synchronization, and work at maximal performance in between synchronization points. The speeds of the concurrent entities do not need to be controlled except when it needs to block and wait for another thread to come to their mutual synchronization event.

The disadvantage of this model of time is that it makes it harder to program threads that are self-timed. Proving correctness of such systems is also harder. This is because a partial order can be linearized in in possibly exponential number of distinct ways. Proving that all possible sequentializations of the system lead to the correct result might lead to an exponential number of possibilities to check. Also, if instead of sequential consistency, another notion of correctness adopted, such as weak consistency, that is even harder. In the efficient but simpler fully time synchronized distributed system every event of the system is uniquely identified with a unique point in a totally ordered time line making it easier to prove correctness.

Therefore, in order to free the programming model from synchronous timing model, and making systems robust to clock synchronization failure based attacks, we have been advocating polychronous model of computation for concurrent systems. However, we do not advocate that programmers design programs against this model manually, or prove correctness themselves. We propose a dataflow model of computation which describes only the intended computation as a set of concurrent dataflow, and a program synthesis technique synthesizes multi-threaded code from such specifications. The generated code does not depend on clock synchronization, and each thread has local notion of time which are totally ordered themselves, and partially ordered globally. The correctness proof obligation is no longer with the implementer but with the modeler who would prove the properties on the concurrent dataflow model. The program synthesis engine of course has to be proven correct in order to claim correctness of the generated code from the model.

In this position paper, we will elaborate on why we consider this polychronous timing model to be advantageous for security and safety critical systems describe our specification formalism, derive the algorithms for code synthesis by analyzing the dependencies, concurrences, and pre-orders. We will illustrate with examples how this model of time makes it simpler to specify, and how code is synthesized.

Bio: Sandeep K. Shukla is a professor of Electrical and Computer Engineering at Virginia Tech. He has published more than 200 conference papers, book chapters, and journal articles. He also co-edited or co-authored nine books. He is a recipient of the PECASE award, NSF CAREER award, Humboldt foundation's Bessel award, a best paper award etc. He is an IEEE computer society distinguished visitor and an ACM visiting speaker. He is currently working with the US Air Force to develop techniques and tools for multi-threaded code synthesis from polychronous specifications. Model of time in computational model is one of his major topics of interest.