Summary of Lamport’s “Time, Clocks, and the Ordering of Events in a Distributed System”

The main focus of this paper was to illustrate the complexity of ordering asynchronous events in a distributed system, and to propose a solution to how the events can be ordered by processes with the use of clocks. The paper started by defining how events should be ordered in a system through the concept of partial orderings and giving a relationship for events that come before others. The paper then introduces logical clocks and defines an algorithm for processing requests. Lamport provides an algorithm which details how processes can request and release resources that are queued up in a distributed system. With this algorithm, the paper solves a critical problem – the ordering of events in a distributed system. This problem is of utmost importance when we consider the two main challenges of working with a distributed system. As we discussed in class, the two details that make a distributed system are that: events are asynchronous and there is an unknown time delay between events. Knowing how to order events for our processes addresses both of these problems in some way – the algorithm lets processes know the order to process events with the use of a timestamp, and queuing up events allows processes to execute them as they receive events (regardless of the amount of time delay).

The following is a summary of the algorithm mentioned which solves the problem of ordering events. First, a process requests a resource to every other process using a timestamp, and puts that message on its request queue. When other processes see the request message, they put it on their request queue and send a response to the initial process. When the initial process is done with the resource, it removes the request message from its queue and sends a time-stamped release resource message to every other process. The other processes remove the request message for the initial process from their queue after receiving the release message. The next process is then given access to the resource based on the next request in the queue using timestamps.

The reading then goes on to speak of alternate approaches. If timestamps are not a reliable indicator of ordering, then it produces an anomalous behavior when executing the algorithm. In order to continue using this algorithm, physical clocks are needed to keep better track of time for asynchronous events. The clock synchronizing algorithm given in the paper solves this problem. It is adequately discussed in the paper, and an extensive proof of the theorem used is provided in the index of the paper.

This work raises research questions that can be analyzed in the future. For instance, how can the algorithm given in the paper be modified in the case that you want to cancel an event sent to every process in the system? Should the queues wait indefinitely for new messages, or should there be some reasonable timeout? Does this algorithm sufficiently create quorum amongst processes in a real-life system? This paper is a useful starting point for people interested in research on the effect of time-ordered events in a distributed system. One could spend more time analyzing algorithms for requesting and releasing resources in a distributed system. While this paper was deep in theory, someone interested in doing research today could do a practical analysis of time-ordered events in an actual system.