Making Order in Chaos: Self-stabilizing Byzantine Synchronization

Danny Dolev*
School of Engineering and Computer Science,
The Hebrew University of Jerusalem, Israel

May 6, 2007

Introduction

The difficulty of fault tolerant synchronization: Coordination and synchronization are among the most fundamental elements of a distributed task. Nodes typically infer about the state of the other correct nodes from their own internal states. In the classic distributed paradigms some extent of initial synchrony or consistency is always assumed [3]. Even in the classic asynchronous network model, although nothing is assumed on the time taken for message delivery, it is typically assumed that nodes have a controlled and common initialization phase [4]. Thus it is assumed that the global state is at least partially consistent so that correct nodes have a common notion as to when the system last initialized. This greatly facilitates the progression of the algorithm in "asynchronous rounds" in which a node knows that if it has commenced some specific round r then all other correct nodes have progressed to at least some lesser round. This leads to a "state-machine replication" approach as a general framework to address the consistency in a distributed environment (see [5]). Typically, the asynchronous model does not allow for deterministic fault tolerance as it might not be possible to distinguish between a late message and a faulty sender (or a lost message). In the synchronous network model, nodes may assume bounded time for message delivery (when the system is stable) in addition to assuming that nodes have a common initialization phase. These two assumptions allow nodes to use timing criteria to deduce whether certain actions should have already taken place. This allows for resilience to permanent faults and plays a pivotal role in the ability to tolerate Byzantine nodes. Synchronization enables correct nodes to determine whether a certain message received at a certain time or with a certain value at this certain time does not agree with the node's perception of the global progress of the algorithm. In order for all

*Part of the work was done while the author visited Cornell University. This research was supported in part by ISF, NSF, CCR, and AFSOR. Email: dolev@cs.huji.ac.il

correct nodes to view symmetrically whether a node does not behave according to the protocol, it is required to assume that nodes have similar perceptions of the progress of the algorithm.

A self-stabilizing algorithm does not assume a common initialization phase. This is required due to transient failures that might corrupt the local state of nodes, such as the notion as to how long ago the system or algorithm was initialized. The combination of self-stabilization and Byzantine fault tolerance poses a special challenge. The difficulty stems from the apparent cyclic paradox of the role of synchronization for containing the faulty nodes combined with the fact that a self-stabilizing algorithm cannot assume any sort of synchronization or inference of the global state from the local state. Observe that assuming a fully synchronous model in which nodes progress in perfect lock-step does not ease this problem (cf. [2]).

The problem in general is to return to a consistent global state from a corrupted global state. The problem as stated through pulse synchronization, is to attain a consistent global state with respect to the pulse event only. I.e. that a correct node can infer that other correct nodes will have invoked their pulse within a very small time window of its own pulse invocation. Interestingly enough, this type of synchronization is sufficient for eventually attaining a consistent general global state from any corrupted general global state [1].

Self-stabilizing Byzantine pulse synchronization is a surprisingly subtle and difficult problem. Non-stabilizing Byzantine algorithms assume that all correct nodes have symmetric views on the other correct nodes. E.g. if a node received a message from a correct node then its assumed all correct nodes did so to. Following transient failures though, a node might initialize in a spurious state reflecting some spurious messages from correct nodes. With the pulse synchronization problem, this spurious state may be enough to trigger a pulse at the node. In order to synchronize their pulses nodes need to broadcast that they have invoked a pulse or that they are about to do so. Correct nodes need to observe such messages until a certain threshold for invoking a pulse is reached. When nodes invoke their pulses this threshold will be reached again subsequent to invoking the pulse, causing a correct node to immediately invoke a pulse again and again.

To prevent incessant pulse invocations, a straightforward solution is to have a large enough period subsequent to the pulse invocation in which a node does not consider received messages towards the threshold. This is exactly where the complimentary pitfall lies, since some correct nodes may initialize in a state that causes them to invoke a pulse based on spurious messages from correct nodes. The consequent pulse message might then arrive at other correct nodes that initialize in a period in which they do not consider received messages. Byzantine nodes can, by sending carefully timed messages, cause correct nodes to invoke their pulses in perfect anti-synchrony forever. It is no trivial task to circumvent these difficulties.

It is interesting to observe that Byzantine (non-stabilizing) pulse synchronization can be trivially derived from Byzantine clock synchronization. Self-stabilizing (non-Byzantine) pulse synchronization can be easily achieved by fol-

lowing any node that invokes a pulse. Self-stabilizing Byzantine pulse synchronization on the other hand is apparently an extremely tricky task.

In the talk we will cover the state of the art in combining the ability to overcome Byzantine faults, while maintaining the self-stabilization property. We will allude at how this approach can be employed in large systems.

References

[1] A. Daliot and D. Dolev, "*Self-stabilization of Byzantine Protocols*", Proc. of the 7th Symposium on Self-Stabilizing Systems (SSS'05 Barcelona), pp. 48-67, 2005.

[2] S. Dolev, and J. L. Welch, "*Self-Stabilizing Clock Synchronization in the presence of Byzantine faults*", Journal of the ACM, Vol. 51, Issue 5, pp. 780 - 799, 2004.

[3] D. Dolev, C. Dwork and L. Stockmeyer, "*On the minimal synchronism needed for distributed consensus*", J. ACM, Vol. 34, Issue 1, pp. 77-97, 1987.

[4] N. Lynch, "*Distributed Algorithms*", Morgan Kaufmann, 1996.

[5] F. B. Schneider, "*Implementing fault-tolerant services using the state machine approach: a tutorial*", ACM Computing Surveys (CSUR), Volume 22, Issue 4, pp. 299-319, 1990.