Brief Announcement: Self-stabilizing Virtual Synchrony

Shlomi Dolev, Chryssis Georgiou, Ioannis Marcoullis, Elad M. Schiller

Brief Announcement: Self-stabilizing Virtual Synchrony

Shlomi Dolev[1], Chryssis Georgiou[2], Ioannis Marcoullis[2], and Elad M. Schiller[3]

[1] Ben-Gurion University of the Negev, Israel.
[2] University of Cyprus, Cyprus.
[3] Chalmers University of Technology, Sweden.

Introduction. Systems satisfying the Virtual Synchrony (VS) [2] property provide message multicast and group membership services in which all system events, group membership changes, and incoming messages, are delivered in the same order. VS is an important abstraction, proven to be extremely useful when implemented over asynchronous, typically large-scale, message-passing distributed systems, as it simplifies the design of distributed applications, e.g., *State Machine Replication* (SMR). The VS property ensures that two or more processors that participate in two consecutive communicating groups should have delivered the same messages. Self-stabilizing systems [1,3] can tolerate transient faults that drive the system to an unpredicted arbitrary configuration. Such systems automatically regain consistency from any such configuration, and then produce the desired system behavior ensuring it for a practically infinite number of successive steps, e.g., 2^{64} steps. We present the first, to our knowledge, *self-stabilizing virtual synchrony* algorithm.

An Overview of Our Results. We consider an asynchronous message passing system consisting of n uniquely identified processors of which a minority may become inactive (crash). Any message that is sent infinitely often from one active processor to another active processor is eventually received. The communication links have known bounded capacity, and can emulate reliable FIFO communication channel protocols using existing self-stabilizing algorithms.

Bounded labeling scheme for multiple writers. We extend the labeling scheme of [1] to support counter incrementing by multiple label creators (writers) rather than by a single writer. The *labels* are related to an integer counter allowing the system to stabilize. A 64-bit counter, for example, is considered to be practically infinite. There are two main challenges to achieve the result. Multiple writers can concurrently create labels. To overcome this issue, we include the writer identity to break symmetry and decide which label is the most recent one. In this way, the scheme ensures that every active processor p_i eventually "cleans up" the system from obsolete labels of which p_i appears to be the creator, but may be the result of the system's initial arbitrary state.

The second challenge is to overcome problems emerging from labels attributed to inactive processors that cannot clean-up their own labels. Note that there is no knowledge of these processors' inactivity. Consider an initial system state including a cycle of labels $\ell_1 \prec \ell_2 \prec \ell_3 \prec \ell_1$, all of the same creator, say p_x, where $\prec$ is the label order relation. If p_x is active, it will eventually learn about these labels and introduce a label greater than them all. But if p_x is inactive, the system's asynchronous nature may present the three labels to some active

processor p_i in their order of precedence and force their adoption, indefinitely. We settle this issue by keeping a label history of proven sufficient label size. The algorithm is proved to be self-stabilizing and to provide a global maximal label.

Practically infinite counter for multiple writers. We extend the labeling scheme to handle *counters*, where a counter consists of a *label*, as used in the labeling scheme; an integer *sequence number*, ranging from 0 to 2^b, say $b = 64$; and a processor *id*. The counter increment algorithm uses the same structures and procedures as the labeling algorithm, but now with counters instead of labels. The challenge for the counter algorithm is to make sure that when a label has an exhausted counter (i.e., one that has reached its maximum) the label is changed and a new label is chosen. The counter algorithm is proved to guarantee eventually monotonic counter increment by multiple writers given a minority of inactive processors. The counter increment algorithm can be used to obtain a self-stabilizing MWMR shared memory emulation.

Self-stabilizing virtual synchrony. Systems guaranteeing the VS property provide two main services: a membership service and a reliable multicast service. We provide these services in a coordinator-based solution, considering a single majority group in the system, the primary partition [2]. The membership service provides the current group *view* of the recently live and connected group members. A view is composed of the view identifier obtained from the counter increment algorithm, and the group membership is provided by a failure detector (FD) as the one described in [3]. The output of the coordinator's FD defines the set of view members; this helps to maintain a consistent membership among the group members, despite inaccuracies between the various FDs. The coordinator is also responsible for the consistency of the multicast mechanism within the group. To this end, it requests, collects and combines input from the group members, and then multicasts the updated information before initiating a new multicast round. Each participant keeps the last delivered message and the view identifier that delivered this message. This, together with the intersection property of majorities, and after taking care of some subtle issues, provides the VS property. As part of our VS solution, we also implement a virtually synchronous SMR algorithm. Every processor maintains a replica of the state machine and the last processed (composite) message. Starting from an arbitrary configuration, our algorithm eventually implements replicated automaton emulation that preserves VS. Full details can be found in [4].

References

1. N. Alon, H. Attiya, S. Dolev, S. Dubois, M. Potop-Butucaru, and S. Tixeuil. Practically stabilizing SWMR atomic memory in message-passing systems. *J. of Comp. and Sys. Sci.*, 81(4): 692–701, 2015.
2. K. Birman. A history of the virtual synchrony replication model. In *Replication: Theory and Practice*, pages 91–120, 2010.
3. P. Blanchard, S. Dolev, J. Beauquier, and S. Delaët. Practically self-stabilizing Paxos replicated state-machine. In *Proc. of NETYS'14*, pages 99–121, 2014.
4. S. Dolev, Ch. Georgiou, I. Marcoullis, and E.M. Schiller. Self-stabilizing virtual synchrony. In *Proc. of SSS'15*, pages 248–264, 2015. (Also in arXiv:1502.05183.)