Self-stabilizing Byzantine Asynchronous Unison*,**

Swan Dubois[1], Maria Gradinariu Potop-Butucaru[1], Mikhail Nesterenko[2], and Sébastien Tixeuil[1]

[1] LIP6 - UMR 7606 Université Pierre et Marie Curie - Paris 6 & INRIA, France
[2] Kent State University, USA

Abstract. We explore asynchronous unison in the presence of systemic transient and permanent Byzantine faults in shared memory. We observe that the problem is not solvable under less than strongly fair scheduler or for system topologies with maximum node degree greater than two.

We present a self-stabilizing Byzantine-tolerant solution to asynchronous unison for chain and ring topologies. Our algorithm has minimum possible containment radius and optimal stabilization time.

Asynchronous unison [2, 3] requires processors to maintain synchrony between their counters called *clocks*. Specifically, each processor has to increment its clock indefinitely while the clock *drift* from its neighbors should not exceed 1. Asynchronous unison is a fundamental building block for a number of principal tasks in distributed systems such as distributed snapshots [4] and synchronization [5, 6].

A practical large-scale distributed system must counter a variety of transient and permanent faults. A systemic transient fault may perturb the system and leave it in an arbitrary configuration. *Self-stabilization* [7, 8] is a versatile technique for transient fault forward recovery. *Byzantine fault* [9] is the most generic permanent fault model: a faulty processor may behave arbitrarily. However, designing distributed systems that handle both transient and permanent faults proved to be rather difficult [10–12]. Some of the difficulty is due to the inability of the system to counter Byzantine behavior by relying on the information contained in the global system configuration: a transient fault may place the system in an arbitrary configuration.

In the context of the above discussion, considering joint Byzantine and systemic transient fault tolerance for asynchronous unison appears futile. Indeed, the Byzantine processor may keep setting its clock to an arbitrary value while the clocks of the correct processors are completely out of synchrony. Hence, we are happy to report that the problem is solvable in some cases. In this paper we present a shared-memory Byzantine-tolerant self-stabilizing asynchronous unison algorithm that operates on chain and ring system topologies. The algorithm operates under a strongly fair scheduler. We show that the problem is unsolvable

* A full version of this work is available in [1].

** This work was funded in part by ANR projects SHAMAN, ALADDIN, and SPADES.

C. Lu, T. Masuzawa, and M. Mosbah (Eds.): OPODIS 2010, LNCS 6490, pp. 83–86, 2010.

for any other topology or under less stringent scheduler. Our algorithm achieves minimal fault-containment radius: each correct processor eventually synchronizes with its correct neighbors. We prove our algorithm correct and demonstrate that its stabilization time is asymptotically optimal.

Related work. The impetus of the present research is the result by Dubois et al [13]. They consider joint tolerance to crash faults and systemic transient faults. The key observation that enables this avenue of research is that the adopted definition of asynchronous unison does not preclude the correct processors from decrementing their clocks. This allows the processors to synchronize and maintain unison even while their neighbors may crash or behave arbitrarily.

There are several pure self-stabilizing solutions to the unison problem [2, 3, 14]. None of those tolerate Byzantine faults. Classic Byzantine fault tolerance focuses on masking the fault. There are self-stabilizing Byzantine-tolerant clock synchronization algorithms for completely connected synchronous systems both probabilistic [11, 15] and deterministic [16, 17]. The probabilistic and deterministic solutions tolerate up to one-third and one-fourth of faulty processors respectively.

Another approach to joint transient and Byzantine tolerance is *containment.* For tasks whose correctness can be checked locally, such as vertex coloring, link coloring or dining philosophers, the fault may be isolated within a region of the system. *Strict stabilization* guarantees that there exists a containment radius outside of which the processors are not affected by the fault [12, 18–20]. We say that that an algorithm is $(c, f)-$strictly stabilizing if it is strictly stabilizing with a radius of c and can tolerate up to f Byzantine processors. Yet some problems are not local and do not admit strict stabilization. However, the tolerance requirements may be weakened to *strong-stabilization* [21, 22] which allows the processors outside the containment radius to be affected by Byzantine processors after the convergence of the system. The faulty processors can affect these correct processors only a finite number of times after the convergence of the system. Strong-stabilization enables solution to several problems, such as tree orientation and tree construction.

Contributions. The first step is to characterize the necessary assumptions on the system (topology, scheduling, and number of faults) to solve the asynchronous unison in a strictly stabilizing way. Following [13] and the fact that a Byzantine processor may simulate a crashed one, we can state some impossibility results.

Theorem 1. *There does not exist a (f, d)-strictly stabilizing solution to the asynchronous unison problem in shared memory for any distance $d \geq 0$ if the communication graph of the distributed system contains processors of degree greater than two or if the number of faults is greater than one or if the scheduler is either unfair or weakly fair.*

Then, it remains to explore the case of asynchronous unison on chains and rings under the strongly fair scheduler with at most one Byzantine processor. We provide in the following the intuition of the provided algorithm.

The *drift* between two processors p and q is the absolute value of the difference between their clock values. Two processors p and q are *in unison* if the drift between them is no more than 1. An *island* is a segment of correct processors such that for each processor p, if its neighbor q is also in this island, then p and q are in unison. A processor with no in-unison neighbors is assumed to be a single-processor island. The main idea of the algorithm is as follows. Our algorithm forms islands of processors with synchronized clocks. The algorithm is designed such that the clocks of the processors with adjacent islands drift closer to each other and the islands eventually merge. If a faulty processor restricts the drift of one such island, for example by never changing its clock, the other islands still drift and synchronize with the affected island. Then, we prove the following theorem:

Theorem 2. *Our algorithm is a* $(1, 0)$*-strictly stabilizing asynchronous unison under the strongly fair scheduler on ring and chains topology.*

Finally, we explore the time complexity of our algorithm and of the problem. We prove the following result:

Theorem 3. *The stabilization complexity of our algorithm is optimal. It stabilizes in* $\Theta(L)$ *asynchronous rounds where* L *is the largest drift between correct processors in the initial configuration of the system.*

Conclusion. In this paper we explored joint tolerance to Byzantine and systemic transient faults for the asynchronous unison problem in shared memory. Some open problems follow.

Solutions under distributed scheduler, that allows multiple concurrent steps, remain to be explored. Another way to complete these results is to consider bounded clocks. The existence of a solution for shared memory execution model opens another avenue of research. It is interesting to consider the existence of a solution in lower atomicity models such as shared register or message-passing. We conjecture that a solution in such model is more difficult to obtain as the lower atomicity tends to empower faulty processors. Indeed, in the shared-register model a Byzantine processor may report differing clock values to its right and left neighbor. Such behavior makes a single fault ring topology essentially equivalent to two fault chain topology. The latter is proven unsolvable. Hence, we posit that in the lower atomicity models, the only topology that allows a solution to asynchronous unison to the chain.

References

1. Dubois, S., Potop-Butucaru, M.G., Nesterenko, M., Tixeuil, S.: Self-stabilizing byzantine asynchronous unison. CoRR abs/0912.0134 (2009)
2. Gouda, M.G., Herman, T.: Stabilizing unison. Inf. Process. Lett. 35(4), 171–175 (1990)
3. Couvreur, J.M., Francez, N., Gouda, M.G.: Asynchronous unison (extended abstract). In: ICDCS, pp. 486–493 (1992)
4. Chandy, K.M., Lamport, L.: Distributed snapshots: Determining global states of distributed systems. ACM Trans. Comput. Syst. 3(1), 63–75 (1985)

5. Awerbuch, B.: Complexity of network synchronization. J. ACM 32(4), 804–823 (1985)
6. Awerbuch, B., Kutten, S., Mansour, Y., Patt-Shamir, B., Varghese, G.: A time-optimal self-stabilizing synchronizer using a phase clock. IEEE Trans. Dependable Sec. Comput. 4(3), 180–190 (2007)
7. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. ACM Commun. 17(11), 643–644 (1974)
8. Dolev, S.: Self-stabilization. MIT Press, Cambridge (March 2000)
9. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982)
10. Daliot, A., Dolev, D.: Self-stabilization of byzantine protocols. In: Herman, T., Tixeuil, S. (eds.) SSS 2005. LNCS, vol. 3764, pp. 48–67. Springer, Heidelberg (2005)
11. Dolev, S., Welch, J.L.: Self-stabilizing clock synchronization in the presence of byzantine faults. J. ACM 51(5), 780–799 (2004)
12. Nesterenko, M., Arora, A.: Tolerance to unbounded byzantine faults. In: 21st Symposium on Reliable Distributed Systems (SRDS 2002), p. 22. IEEE Computer Society, Los Alamitos (2002)
13. Dubois, S., Potop-Butucaru, M., Tixeuil, S.: Brief announcement: Dynamic FTSS in Asynchronous Systems: the Case of Unison. In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 291–293. Springer, Heidelberg (2009)
14. Boulinier, C., Petit, F., Villain, V.: When graph theory helps self-stabilization. In: Chaudhuri, S., Kutten, S. (eds.) PODC, pp. 150–159. ACM, New York (2004)
15. Ben-Or, M., Dolev, D., Hoch, E.N.: Fast self-stabilizing byzantine tolerant digital clock synchronization. In: Bazzi, R.A., Patt-Shamir, B. (eds.) PODC, pp. 385–394. ACM, New York (2008)
16. Dolev, D., Hoch, E.N.: On self-stabilizing synchronous actions despite byzantine attacks. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 193–207. Springer, Heidelberg (2007)
17. Hoch, E.N., Dolev, D., Daliot, A.: Self-stabilizing byzantine digital clock synchronization. In: [23], pp. 350–362
18. Masuzawa, T., Tixeuil, S.: Stabilizing link-coloration of arbitrary networks with unbounded byzantine faults. International Journal of Principles and Applications of Information Science and Technology (PAIST) 1(1), 1–13 (2007)
19. Sakurai, Y., Ooshita, F., Masuzawa, T.: A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In: Higashino, T. (ed.) OPODIS 2004. LNCS, vol. 3544, pp. 283–298. Springer, Heidelberg (2005)
20. Dubois, S., Masuzawa, T., Tixeuil, S.: The impact of topology on byzantine containment in stabilization. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 495–509. Springer, Heidelberg (2010)
21. Masuzawa, T., Tixeuil, S.: Bounding the impact of unbounded attacks in stabilization. In: [23], pp. 440–453
22. Dubois, S., Masuzawa, T., Tixeuil, S.: On byzantine containment properties of the min+1 protocol. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 96–110. Springer, Heidelberg (2010)
23. Datta, A.K., Gradinariu, M. (eds.): SSS 2006. LNCS, vol. 4280. Springer, Heidelberg (2006)