Synchronous vs. Asynchronous Unison

Christian Boulinier, Franck Petit, and Vincent Villain
LaRIA, CNRS FRE 2733, Université de Picardie Jules Verne, France

Abstract. This paper considers the self-stabilizing unison problem. The contribution of this paper is threefold. First, we establish that when any self-stabilizing asynchronous unison protocol runs in synchronous systems, it converges to synchronous unison if the size of the clock $K$ is greater than $C_G$, $C_G$ being the length of the maximal cycle of the shortest maximal cycle basis if the graph contains cycles, 2 otherwise (tree networks). The second result demonstrates that the asynchronous unison in [3] provides a universal self-stabilizing synchronous unison for trees which is optimal in memory space. It works with any $K \geq 3$, without any extra state, and stabilizes within $2D$ rounds, where $D$ is the diameter of the network. This protocol gives a positive answer to the question whether there exists or not a universal self-stabilizing synchronous unison for tree networks with a state requirement independant of local or global information of the tree. If $K = 3$, then the stabilization time of this protocol is equal to $D$ only, i.e., it reaches the optimal performance of [3]. The third result of this paper is a self-stabilizing unison for general synchronous systems. It requires $K \geq 2$ only, at least $K + D$ states per process, and its stabilization time is $2D$ only. This is the best solution for general synchronous systems, both for the state requirement and the stabilization time.

1 Introduction

We consider the problem of phase synchronization [9] in self-stabilizing uniform distributed systems. Phase synchronization consists in designing a synchronization mechanism devoted to a distributed protocol made of a sequence of phases $0, 1, \ldots$ such that no process starts to execute its phase $i + 1$ before all processes have completed their phase $i$. It is also required that no process will be permanently blocked from executing its phase $i + 1$ if all processes have completed their phase $i$. This mechanism induces a global abstract device called phase clock to maintain the current phase number, incremented each time a phase completes. In a distributed environment, each process maintains its own copy of the phase clock. Therefore, the problem consists in the design of a protocol insuring that all the phase clocks are in phase. The phrase “in phase” has a natural meaning in synchronous systems. In such systems, a global signal is assumed to simultaneously increment all clock variables. So, the clocks are in phase if the values of all clock variables are identical. The (synchronous) unison [7] problem consists in the design of a protocol to keep all clocks in phase, i.e., to insure identical time on all clocks and increment in unison.
In asynchronous systems, there is no global signal. So, one can at most ensure that no process starts to execute its phase $i + 1$ before all processes have completed their phase $i$. But this kind of synchronization needs $O(D)$ rounds between two phases. So, in general, the synchronization requirement is relaxed as follows: the clock are in phase if the values of two neighboring processes do not differ by no more than 1, and the clock value of each process is incremented by 1 infinitely often. The asynchronous unison \cite{4} deals with this criteria.

**Related Works.** Numerous works in the area of self-stabilization deals with the phase synchronization problem. In this paper, we focus on deterministic solutions for uniform systems only. Moreover, we limit our discussion to tree and general networks. In the rest of this section, $K$ is the size of the clock, $S$ is the number of states the processes are required to have, $D$ is the diameter of the network, $n$ the number of processes, and $\Delta$ is the maximum degree of a process.

**Self-stabilizing Synchronous Unison.** The first self-stabilizing synchronous unison is given in \cite{7}. It works on a general graph but it requires unbounded clocks. The first protocol with a bounded memory space is proposed in \cite{1}. It needs $K \geq 2\Delta D$, and stabilizes in $3\Delta D$ steps. As it is noticed in \cite{8}, the $\Delta$ factor is due to the model: It is assumed that a process cannot read more than the state of one neighbor at a time. From now on, all the protocols we discuss will be assumed to work on a model where every process can read the state of all its neighbors at a time. In this model, the solution in \cite{1} needs $K \geq 2D$ ($S = K$) and stabilizes in at most $3D$ steps only. To our knowledge, this is the only deterministic synchronous unison for general uniform networks (according to our restrictions).

A solution for tree networks is proposed in \cite{8}. It requires $K = 3^m$ ($m > 0$), $S = K$, and stabilizes in $(D \times (K - 1))/2$ steps. Note that the stabilization time is equal to $D$ only for $m = 1$ ($K = 3$), but is greater than $2D$ when $m \geq 2$. Thus, in the case $3^m \geq 2D$, the solution in \cite{1} is better. In terms of stabilization time, the best solution on trees is proposed in \cite{11}. It stabilizes in at most $D$ steps only. Moreover, this protocol is “universal”, meaning that $K$ can take any value greater than or equal to 2 and the state requirement does not need any global information like either $n$ or $D$. It depends on $\Delta$ only: $S = (\Delta + 1)K$.

**Self-stabilizing Asynchronous Unison.** The self-stabilizing asynchronous unison was introduced in \cite{4}. Two deterministic protocols are proposed. The former works assuming unbounded clock, the latter needs $K \geq n^2$ (according to our model) ($S = K$). In \cite{3}, the authors show the lower bound for $K$. $K$ must be greater than $C_G$, where $C_G$ is the length of the maximal cycle of the shortest maximal cycle basis if the graph contains cycles, 2 otherwise (tree networks). They also show that $S$, the amount of space, must be greater than $K + T_G - 2$, where $T_G$ is the length of the longest chordless cycle (0 in tree networks). In the same paper, they present an algorithm reaching these bounds. This protocol is optimal in terms of state requirement. One can notice that $C_G$ and $T_G$ are bounded by $n$. So, even if $C_G$ and $T_G$ are unknown, we can choose $K \geq n + 1$ and $S = K + n - 2 \geq 2n - 1$. The protocol is still better than \cite{4}.