Brief Announcement: Clock Distribution with Gradient TRIX

Christoph Lenzen
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Shreyas Srinivas[1]
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

Gradient clock synchronisation (GCS) algorithms minimise the worst-case clock offset between the nodes in a distributed network of diameter D and size n. They achieve optimal offsets of $\Theta(\log D)$ locally, i.e. between adjacent nodes [8], and $\Theta(D)$ globally [1]. A key open problem in this area is to achieve fault tolerance at minimal overhead in terms of the number of edges.

In this work, we achieve this goal under the assumption of an average-case distribution of faults, i.e., nodes fail with independent probability $p \in o(n^{-1/2})$. In more detail, we present a self-stabilising GCS algorithm for a grid-like directed graph with in- and out-degrees of 3. Note that even for tolerating a single fault, this degree is necessary. Moreover, the failure probability p is the largest possible ensuring the necessary condition that for each node at most one in-neighbour fails with probability $1 - o(1)$. Our algorithm achieves asymptotically optimal local skew of $\Theta(\log D)$ with probability $1 - o(1)$; this holds under general worst-case assumptions on link delay and clock speed variations, provided they change slowly relative to the speed of the system.

On the one hand, our results are of practical interest. As we discuss with care, the fault model is suitable for synchronously clocked hardware. Since our algorithm can simultaneously sustain a constant number of arbitrary changes due to faults in each clock cycle, it achieves sufficient robustness to dramatically increase the size of synchronously clocked Systems-on-Chip.

On the other hand, our result is of a theoretical and algorithmic nature. We show that for a worst-case distribution of f 1-local faulty nodes within our fault model's locality constraints, our algorithm achieves a local skew of $O(5^f \log D)$, while for our model with probabilistic distribution of faults the algorithm achieves $O(\log D)$. Our work raises questions for further theoretical investigation on techniques for fault tolerance and trade-offs between fault distribution and edge density of graphs.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Hardware → Fault tolerance

Keywords and phrases local skew, gradient clock synchronisation, average-case fault-tolerance, self-stabilisation, Systems-on-Chip

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.51

Related Version *Full Version*: `https://arxiv.org/abs/2301.05073` [9]

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 716562).

Acknowledgements Shreyas Srinivas is a doctoral student at the Graduate School of Computer Science, Saarbrücken.

1 The Basic Problem

The problems of distributed clock synchronisation and distribution are concerned with getting nodes in a network to agree on a common notion of time, expressed by their output logical clocks. The extent of disagreement is quantified by *clock skews* i.e. the maximum

[1] corresponding author

instantaneous difference in the output clock values of two nodes. An algorithm for this problem must seek to minimise two kinds of skew: *local skew*, i.e. clock skews between adjacent nodes, and *global skew*,i.e. clock skews between any pair of nodes in the network. Equally such an algorithm must be resilient to faults, both permanent and transient. From a pragmatic standpoint, we would like to achieve all the above properties without cluttering up our graph with replicated nodes and edges.

In this work we study the problem of distributing clock signals through a grid like *graph of diameter D* with optimal global and local clock skews of respectively $O(D)$ and $O(\log D)$, which is self stabilising and resilient to a reasonable distribution of faults. Finally, we would like to achieve the aforementioned resilience to faults by adding the minimum possible amount of edge and vertex redundancy into our network. While this last requirement arises from our desire to clock VLSI systems, the question is of independent theoretical interest since edge connectivity is an expensive resource in several domains. Our results challenge the notion that fault tolerance always requires masses of edge and vertex replication, by showing that reasonable levels of fault tolerance can be achieved at little loss of optimal performance without excessive edge replication. We summarise our desiderata below:

Fault-Tolerant Clock Synchronisation Problem (Informal)

Compute at each node of a distributed system a logical clock with the following properties.

- **Minimising Global Skew**: The skew between any pair of nodes i.e. *Global Skew* is minimised as a function of the network diameter D: $\Theta(D)$.
- **Minimising Local Skew**: The skew between adjacent pairs of nodes i.e. *Local Skew* is minimised as a function of the network diameter D: $\Theta(\log D)$.
- **Fault Tolerance**: A set of at most f permanently faulty processes according to a given fault model does not increase clock skews (up to a constant factor).
- **Self-Stabilisation**: After system-wide transient (i.e., temporary) faults, the processes re-converge to optimal skews.
- **Optimal Edge Density**: Achieve the above in a network topology with minimal node degree.

2 Our Model

We describe a slightly simplified model and leave the motivation behind it to Section 4.

- **Our Network**: We describe our *grid-like* network here. Starting with a simple connected base graph $H = (V, E)$ of minimum degree 2 and diameter D, we derive the graph $G = (V_G, E_G)$ for synchronisation as follows: for each $\ell \in \mathbb{N}$ we create a copy V_ℓ of V. Denoting by (v, ℓ) the copy of $v \in V$ in V_ℓ, we define $E_\ell := \{((v, \ell), (w, \ell+1)) \mid \{v, w\} \in E \vee v = w\}$. We now obtain G by setting $V_G := \bigcup_{\ell\in\mathbb{N}} V_\ell$ and $E_G := \bigcup_{\ell\in\mathbb{N}} E_\ell$. That is, for each *layer* $\ell \in \mathbb{N}$ we have a copy of $v \in V$, which has outgoing edges to the copies of itself and all its neighbours on layer $\ell + 1$, where ℓ is bounded from above by some value in $\Theta(\sqrt{n})$. Since V_G is a DAG, we refer to out-neighbours as *successors* and in-neighbours as *predecessors*. An example base graph and the construction of two layers from it are shown in Figure 1.

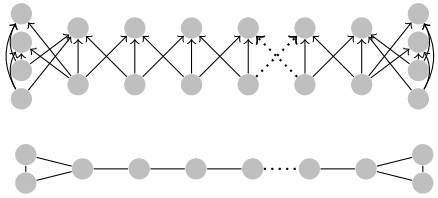

Figure 1 The figure on the bottom shows an example base graph and the figure on the top shows a two layer example of the grid graph constructed from the base graph.

- **Fault Model**: An unknown subset $F \subset V_G$ is *faulty*, meaning that these nodes do not adhere to the clock distribution protocol. We assume that each node fails independently with probability $p \in o(1/\sqrt{n})$.[2] In particular, this entails that with probability $1 - o(1)$, no node has two faulty predecessors, i.e., faults are 1-local.
- **Communication:** Each node can broadcast pulse messages on its out-edges. If node $v_\ell \in V_\ell$ broadcasts at time $t_{v,\ell}$, its successors receive its message at (potentially different) times from $[t_{v,\ell} + d - u, t_{v,\ell} + d]$. The maximum end-to-end *delay* d includes computation-induced ones. Typically, the delay *uncertainty* u is much smaller than d. We assume delays change much slower than the output clock frequency. Faulty nodes can send pulses at arbitrarily.
- **Local Clocks and Computations:** Each node (v, ℓ) has an imperfect local time reference by query access to a *hardware clock* $H_{v,\ell} \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ satisfying

  $$\forall t < t' \in \mathbb{R}_{\geq 0},\ t' - t \leq H_{v,\ell}(t') - H_{v,\ell}(t) \leq \vartheta(t' - t)$$

  for a drift parameter $\vartheta > 1$. No phase relation is assumed between the hardware clocks. Hardware clock speeds change slowly relative to the frequency of the output clocks. Computations are deterministic and can be triggered by arrival of messages or timers off the hardware clocks.
- **Output and Skew**: The algorithm outputs logical clocks in the form of pulses such the *pulses* generated by correct nodes are synchronised. For simplicity, we assume that correct nodes on layer 0 generate well-synchronised pulses at times $t^k_{v,0}$ for $k \in \mathbb{N}_{>0}$ at a frequency we control. Other correct nodes generate pulses $t^k_{v,\ell}$, $k \in \mathbb{N}_{>0}$, based on the pulse messages received from their predecessors. We seek to minimise the worst-case *local skew* that the algorithm guarantees. The local skew is defined as the largest offset between the k-th pulses of adjacent nodes on the same layer or pulses k and $k+1$ of adjacent nodes on layers ℓ and $\ell + 1$, whichever is larger. Formally, for $\ell \in \mathbb{N}$, we define

  $$\mathcal{L}_\ell := \sup_{k \in \mathbb{N}} \max_{\substack{\{v,w\} \in E \\ (v,\ell),(w,\ell) \notin F}} \{|t^k_{v,\ell} - t^k_{w,\ell}|\}, \quad \mathcal{L}_{\ell,\ell+1} := \sup_{k \in \mathbb{N}} \max_{\substack{((v,\ell),(w,\ell+1)) \in E_\ell \\ (v,\ell),(w,\ell+1) \notin F}} \{|t^{k+1}_{v,\ell} - t^k_{w,\ell+1}|\},$$

  and $\mathcal{L} := \sup_{\ell \in \mathbb{N}} \max\{\mathcal{L}_\ell, \mathcal{L}_{\ell,\ell+1}\}$.
  Between consecutive layers, we synchronise consecutive pulses. After initialisation, which is complete once the first pulse propagated through the grid, this is equivalent to a layer-dependent index shift of pulse numbers.

[2] We stress that this requirement is not stronger than that of [3, 10] and [2] for $f = 1$ in any practical sense. If faults correlate in a way clustering them together, it is likely that neighbours fail. Assuming independence (or, more generally, negative correlation) captures "faults do not cluster" in the most straightforward way that allows us to exploit this property beyond immediate neighbours.

3 The Key Technical Ingredients and Results

Our results are obtained from a medley of two lines of results, see Table 1 for a summary.

- **Clock Synchronisation and Optimal Skews**: This line dates back to the work of Fan and Lynch [4], who introduced the problem of gradient clock synchronisation (hereon *the GCS problem*), which expanded the clock synchronisation problem to general graphs. A fruitful line of work [1, 7, 8] established both lower and upper bounds of $\Theta(\log D)$ on skews that could be achieved. In fact the aptly named *GCS* (family of) *algorithms* additionally guaranteed the property of resilience to transient faults i.e. self-stabilisation. However the *GCS* algorithm is stubbornly intolerant of even a single faulty node that can lie to different neighbours which are otherwise distantly connected. In [2], the authors achieved resilience to 1-local faults by a massive replication of the vertices and edges in the original network (based on a general scheme with factor-$O(f^2)$ edge overhead), but requiring 20-fold edge replication and 4-fold vertex replication renders the scheme impractical.
- **Fault Tolerance Clock Distribution in Sparse Grids:** The other line of work [3, 10] focused on protocols for distributing a signal generated from a fault tolerant base network across a grid with fault tolerance and optimal edge connectivity. In these schemes, the nodes have no local hardware clocks of their own. They forward pulses as they received them according to the forwarding protocol. TRIX [10] has a simple pulse forwarding rule: Each node receives 3 copies of each pulse from 3 grid-adjacent in-neighbours and forwards the median copy. It achieves 1-local fault tolerance at the cost of 2 extra edges per node, but with $O(D)$ local skew.

Our Idea. It is useful to think of the pulses output by clock synchronisation and distribution schemes as discrete time points in a logical clock value they generate for each time instant. Thus we can speak of our logical clock functions being set forward or backward or have its rate of change altered. In the actual algorithm this is handled by altering the time at which successive pulses of the output clock are emitted.

We seek the best of both worlds described above. The *GCS algorithm* follows a "*move slowly to the midpoint of all your neighbours' clocks up to a discrete value* κ" rule, i.e. the *gradient rule.* Here κ is a constant picked by the algorithm designer that subsumes measurement errors from all the potential sources of uncertainty, that arise when nodes estimate their neighbours' logical clock values. The GCS algorithm offers optimal local skew, but poor fault tolerance.

In the *TRIX* distribution scheme nodes adjust their logical clocks immediately per a "*jump immediately to the median clock of three*" rule to pick one of three pulses as reference, i.e. the *median* rule. These nodes have no local reference and they merely forward pulses as they receive the second copy of each pulse. This scheme offers excellent 1-local fault tolerance but has sub-optimal $O(D)$ local skew.

Gradient TRIX. Our scheme *Gradient TRIX* attempts to combine these two rules as follows:

- It adapts a generalisation of the TRIX grid, described in Section 2. In particular, unlike TRIX, the nodes now have local clock references.
- The simple *median* pulse forwarding rule is replaced by a wait and forward rule configurable according to a parameter Λ that dictates the time period we seek to achieve for the output pulses. This fixes Λ as well as $\kappa = \Omega\left(u + \left(1 - \frac{1}{\vartheta}\right)(\Lambda - d)\right)$.

- The pulse forwarding rule is a variant of the *GCS algorithm* that safely and consistently combines the *gradient rule* with a modified *median rule*. More specifically, in addition to discrete adaptations of traditional GCS, typically called *Slow and Fast conditions* [9, Definition 9 and 10], we have a third set of *Jump Conditions* [9, Definition 11].

Intuitively, each row of the grid is playing a pass the GCS parcel game. For the duration of forwarding one pulse, each row is pretending to simulate a variation of the GCS algorithm on the base graph and then pass the baton to the next row. It is in this intuition that one can glimpse the idea behind the skew result of Theorem 1.

▶ **Theorem 1.** *If there are no faults, then* $\mathcal{L}_\ell \leq 4\kappa(2+\log D)$ *for all* $\ell \in \mathbb{N}$.

This bound also accounts for suitable parameter choices that ensure that adjacent rows are closely synchronised, while a much more challenging version of the gradient property ensures synchronisation within the rows of the grid-like graph.

Up to technical details, the algorithm's self-stabilisation property is an immediate consequence of the directed propagation of pulses through the grid; once the first layer starts generating pulses at the right frequency with small local skew, the other layers follow.

▶ **Theorem 2.** *The pulse propagation algorithm can be implemented in a self-stabilising way. It stabilises within* $O(\sqrt{n})$ *pulses.*

Further, f permanent 1-local faults in the grid become temporary faults from the perspective of the GCS algorithm simulated on the base graph. However, for each row containing such a faulty node, the local skew might be increased by a constant factor in the worst case. Thus, we get what appears to be a substantial skew build up in the worst situation that f 1-local faults permit.

▶ **Theorem 3.** *If there are at most* f *faulty nodes and none in layer* 0*, then* $\mathcal{L}_\ell \in O(5^f \kappa \log D)$.

However, when faults are uniformly randomly distributed with each node being faulty with probability i.i.d. $o(1/\sqrt{n})$, the faults are sufficiently sparse that self-stabilisation of the simulated GCS algorithm will reduce the local skew fast enough to prevent the above exponential increase.

▶ **Theorem 4.** *With probability* $1-o(1)$*,* $\mathcal{L}_\ell \in O(\kappa \log D)$ *for all* $\ell \in \mathbb{N}$.

A limitation of our results inherent to the directed propagation of pulses that ensures self-stabilisation of the overall scheme is that sudden changes in the timing of many links disrupt synchronisation.

▶ **Theorem 5.** *If faulty nodes do not change the timing of their output pulses, then* $\mathcal{L} \in O(\kappa \log D)$ *with probability* $1-o(1)$.

On the other hand, in the considered application scenario of clock distribution on chips, the scheme is strong enough to handle the expected limited changes that occur in a clock cycle, i.e., a sub-nanosecond timescale.

▶ **Corollary 6.** *With probability* $1-o(1)$*,* $\mathcal{L} \in O(\kappa \log D)$ *even when in each pulse (i) a constant number of faulty nodes change their output behaviour and timing, (ii) link delays vary by up to* $n^{-1/2}u \log D$*, and (iii) hardware clock speeds vary by up to* $n^{-1/2}(\vartheta-1)\log D$.

[3] Given a graph topology G, the augmented graph contains a $3f+1$-clique of replica vertices for each node v in G and $\Theta(f^2)$ copies of each edge $\{v,w\} \in G$ corresponding to all the possible pairs of the replicas of v and w

Table 1 Comparison with related work. Except GCS, "resilience" refers to Byzantine fault-tolerance, i.e., worst-case behaviour of faulty nodes. However, in our work the fault model is restricted: Only a few faulty nodes change their behaviour within a short amount of time. In turn, we are the first to simultaneously achieve optimal skew bounds, self-stabilisation, and minimal degrees.

method	global skew	local skew	resilience	self-stab.	graph topology
LW [11]	$O(1)$	$O(1)$	$< n/3$	no	complete ($D = 1$)
KL [6]	$O(1)$	$O(1)$	$< n/3$	yes	complete ($D = 1$)
HEX [3]	$O(dD)$	$d+O(u^2D/d)$	1-local	yes	grid-like, suboptimal degree
TRIX [10]	$O(uD^2)$	$O(uD)$	1-local	yes	grid-like, optimal degree
GCS [8]	$O(uD)$	$O(u \log D)$	crashes only	yes	arbitrary
Fault-tolerant GCS [2]	$O(uD)$	$O(u \log D)$	f-local	yes	$\Theta(f^2)$-augmented arbitrary graph[3]
Gradient TRIX **(this work)**	$O(uD)$	$O(u \log D)$	independent $p \in o(n^{-1/2})$	yes	grid-like, optimal degree
Gradient TRIX **(this work)**	$O(uD)$	$O(5^f u \log D)$	1-local, f faults	yes	grid-like, optimal degree

4 Motivating our Model: An Exercise in Theory Building

In this final section, we take a closer look at some of our modelling choices that might appear strange at first glance. A key motivation of this work is to produce theoretically correct algorithms which can be applied to the synchronous clocking of VLSI systems. This guides our modelling choices on two fronts:

- **Our Topology**: At a very high level, we would like to synchronise so-called clock islands on modern VLSI systems that currently rely on expensive asynchronous communication. This naturally suggests a grid-like topology. However, a simple grid does not suffice. Even with our extremely sparse network connectivity, we require each node in every row to have three neighbours in adjacent rows, meaning that the nodes at the right and left boundary "miss" a neighbour. Mathematically, the most elegant solution would be a cylinder, but embedding it on a rectangular grid induces a wasteful factor 2 overhead. Instead, we fall back to replicating each node on the right and left boundary once and connecting the two copies. Our scheme is phrased in a more general way, allowing for arbitrary base graphs of minimum degree 2. Our approach achieves this at asymptotically negligible overhead.
- **Our Fault Model**: Here, we strike a fine balance between practical viability and the theoretical optimum. A large class of permanent faults can be chalked up to manufacturing process variations and ageing. While there are correlations, the dominant contributing factors are approximately i.i.d. Further variations due to long voltage droops and temperature variations happen over times ranging a few microseconds to milliseconds [5, chp. 7]; orders of magnitude longer than the typical clock cycle. This justifies assuming that most delays do not change dramatically between consecutive pulses. Note that local oscillators are, ultimately, timed by such delays, so this applies to changes in hardware clock speeds as well.

References

1 Saâd Biaz and Jennifer Lundelius Welch. Closed Form Bounds for Clock Synchronization under Simple Uncertainty Assumptions. *Information Processing Letters*, 80:151–157, 2001. doi:10.1016/S0020-0190(01)00151-X.

2 Johannes Bund, Christoph Lenzen, and Will Rosenbaum. Fault Tolerant Gradient Clock Synchronization. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 357–365, 2019. doi:10.1145/3293611.3331637.

3 Danny Dolev, Matthias Függer, Christoph Lenzen, Martin Perner, and Ulrich Schmid. HEX: Scaling honeycombs is easier than scaling clock trees. *Journal of Computer and System Sciences*, 82(5):929–956, 2016. doi:10.1016/j.jcss.2016.03.001.

4 Rui Fan and Nancy Lynch. Gradient Clock Synchronization. In *Symposium on Principles of Distributed Computing (PODC)*, pages 320–327, 2004. doi:10.1145/1011767.1011815.

5 David Harris and N Weste. Cmos vlsi design. *ed: Pearson Education, Inc*, 2010.

6 Pankaj Khanchandani and Christoph Lenzen. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. *Theory of Computing Systems*, 2018.

7 Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. Clock Synchronization with Bounded Global and Local Skew. In *Symposium on Foundations of Computer Science (FOCS)*, pages 509–518, 2008. doi:10.1109/FOCS.2008.10.

8 Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. Tight Bounds for Clock Synchronization. *J. ACM*, 57(2), 2010. doi:10.1145/1667053.1667057.

9 Christoph Lenzen and Shreyas Srinivas. Gradient trix, 2023. arXiv:2301.05073, doi:10.48550/arXiv.2301.05073.

10 Christoph Lenzen and Ben Wiederhake. TRIX: Low-Skew Pulse Propagation for Fault-Tolerant Hardware, 2020. arXiv:2010.01415.

11 Jennifer Lundelius Welch and Nancy A. Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. *Information and Computation*, 77(1):1–36, 1988. doi:10.1016/0890-5401(88)90043-0.