Brief Announcement: On the Time Complexity of Distributed Topological Self-Stabilization*

Dominik Gall (TUM), Riko Jacob (TUM), Andrea Richa (ASU), Christian Scheideler (UPB), Stefan Schmid (UPB), Hanjo Täubig (TUM)

1 Introduction

This brief announcement proposes a new model to measure the distributed time complexity of topological self-stabilization. In the field of topological self-stabilization, nodes—e.g., machines in a p2p network—seek to establish a certain network structure in a robust manner (see, e.g., [2] for a distributed algorithm for skip graphs). While several complexity models have been proposed and analyzed over the last years, these models are often inappropriate to adequately model parallel efficiency: either they are overly pessimistic in the sense that they can force the algorithm to work serially, or they are too optimistic in the sense that contention issues are neglected. We hope that our approach will inspire researchers in the community to analyze other problems from this perspective. For a complete technical report about our model, related literature and algorithms, the reader is referred to [1].

2 Case Study: Graph Linearization

Our model is best understood with an example. We hence examine a concrete problem, namely *graph linearization*: Given a connected overlay network with unique node identifiers, we want to construct a chain network which is sorted w.r.t. these IDs. For simplicity, we consider undirected graphs only.

We study two distributed algorithms LIN_{all} and LIN_{max} that work on *node triples*: A so-called *linearization step* involves three nodes u, v, and v' with the property that u is connected to v and v' and either $u < v < v'$ or $v' < v < u$. In both cases, u may command the nodes to move the edge $\{u, v'\}$ to $\{v, v'\}$. If $u < v < v'$, this is called a *right* linearization and otherwise a *left* linearization (see Figure 1). Since only three nodes are involved in such a linearization, this is an efficient operation.

Algorithm LIN_{all} is very simple: Each node constantly tries to linearize its neighbors according to the *linearize left* and *linearize right* rules. In doing so, *all* possible triples on both sides are proposed to a hypothetical scheduler. LIN_{max} is similar to LIN_{all}: but instead of proposing all possible triples on each side, LIN_{max} only proposes the triple which is the furthest (w.r.t. IDs) on the corresponding side.

* Research supported by the DFG project SCHE 1592/1-1, and NSF Award number CCF-0830704.

3 Parallel Time Complexity

We define the *distributed runtime* of a self-stabilizing algorithm as the total number of *rounds* required in the worst-case to establish a desirable topology (e.g., a linearized one) from an *arbitrary* initial network.

A round should capture what can be done in a distributed setting per unit time. In our approach, in each round, the scheduling layer may select any set of *independent operations* to be executed by the nodes, in the sense that each node v can only be part of *one* operation (i.e., one linearization triple): if multiple neighbors of v (or v itself) want to change—i.e., add or remove—edges incident to v, at most one edge change is scheduled. Thus, no node is overloaded.

Fig. 1. Left and right linearization step.

We studied the runtime of $\mathrm{LIN}_{\mathrm{all}}$ and $\mathrm{LIN}_{\mathrm{max}}$ for different scheduling regimes: (1) *Worst-case scheduler* $\mathcal{S}_{\mathrm{wc}}$: This scheduler must select a *maximal* independent set of possible linearization steps in each round, but it may do so to enforce a runtime (or work) that is as large as possible. (2) *Randomized scheduler* $\mathcal{S}_{\mathrm{rand}}$: This scheduler considers the set of operations in a random order and selects, in one round, every operation that is independent of the previously selected operations in that order. (3) *Greedy scheduler* $\mathcal{S}_{\mathrm{greedy}}$: This scheduler orders the nodes according to their degrees, from maximum to minimum. (4) *Best-case scheduler* $\mathcal{S}_{\mathrm{opt}}$: The round triples are selected in order to minimize the runtime (or work) of the algorithm.

The analysis of the different schedulers provides interesting insights into an algorithm's performance. In particular, the randomized scheduler can be regarded as a distributed mechanism where node coordinate their operations by local control. We have derived the following bounds [1].

Theorem 1 *Under a worst-case scheduler $\mathcal{S}_{wc}$, LIN_{max} terminates after $O(n^2)$ work (single linearization steps), where n is the total number of nodes in the system. This is tight in the sense that there are situations where LIN_{max} requires $\Omega(n^2)$ rounds under $\mathcal{S}_{wc}$. LIN_{all} runs at most $O(n^2 \log n)$ rounds under $\mathcal{S}_{wc}$and at most $O(n \log n)$ rounds under $\mathcal{S}_{greedy}$. On the other hand, there are situations where both LIN_{all} and LIN_{max} require at least $\Omega(n)$ rounds, even under an optimal scheduler $\mathcal{S}_{opt}$.*

References

1. D. Gall, R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Täubig. Modeling scalability in distributed self-stabilization: The case of graph linearization. Technical Report TUM-I0835, Technische Universität München, Computer Science Dept., Nov. 2008.
2. R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Täubig. A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. In *Proc. PODC*, 2009.