Communication Efficient Self-Stabilizing Leader Election

Final presentation

Sebastian Holler
Introduction

- **Distributed systems**: Individual computing devices communicating with each other
- **Message passing model**: Nodes interact by exchanging messages over adjacent edges
- **Synchronous systems**: Time is divided in discrete, synchronous rounds
Introduction

• **Spanning tree:**
  – A tree connecting all nodes of the graph
  – Enables broadcast and convergecast
  – Requires agreement on root node

• **Leader election:**
  – Nodes elect leader among them
  – In the end exactly one leader

➢ Trade-off between time and message complexity

https://www.sixhat.net/tag/minimum-spanning-tree/
Introduction

• *Communication overhead* as efficiency measure:
  – Number of messages
  – Size of messages

• *Self-stabilizing* algorithms:
  – Initial node configuration chosen adversarially
  – Reach a correct state in finite number of rounds
    ➢ Introduces fault tolerance

• Algorithms solving self-stabilizing leader election communication efficient quite rare
High-level concept

• Initial knowledge of nodes:
  – IDs of adjacent edges
  – Upper limit $N$ on number of nodes

• General idea:
  – Merge node-disjoint subtrees over *crossing* edges repeatedly
  – Root of resulting spanning tree becomes leader
Algorithm is divided into phases, in which each root follows these steps:
Traverse module

- Token traverses tree in DFS order
- Only node holding token can send messages
  - Communication efficiency
- Simulation of Broadcast and convergecast by piggybacking
  - Communication efficiency
Fault detection

• Restart of node if fault is detected during traversal:
  1. Transform node into new singleton tree
  2. Reset node’s internal variables
  3. Start new propose phase

• Fault handling:
  – Node timeout
  – Malformed trees
  – Root receives signal to start traversal but does not hold cold token
Algorithm is divided into phases, in which each root follows these steps:

1. Random coin toss
   - Propose
     - Search epochs
     - Root transfer epoch
     - Proposing epoch
   - Accept
     - Accepting epochs
Cross module

- Based on randomized *FindAny* algorithm by King et al.
- Finds crossing edge with constant non-zero probability
- Constant number of iterations of broadcast and convergecast required
- Constant message size by running $O(\log n)$ traversals per original iteration
Transfer module

- Re-root subtree to node adjacent to crossing edge
- Swap parent-child relations along direct path
- Transport cold token to new root
Algorithm is divided into phases, in which each root follows these steps:
Accept module

- Accept incoming merging proposals
- $O(\log n)$ traversal iterations
- *Retraction ready* nodes can accept one proposal per iteration
- Proposing node dissolves token and becomes child
Theorem 1

• Each node is restarted at most once
• There exists some time \( t_r = O(N) \), after which:
  – All nodes are part of any correct subtree
  – No node is restarted anymore
  – No token dies

➢ All faults from initial configuration are fixed
Theorem 2

- $t_s \geq t_r$ earliest time s.t. only one token is alive
- One final subtree covers the whole graph
- Self-stabilization reached
- $t_s = O(N \log^2 N)$ in expectation and whp.
Theorem 3

- Until stabilization, \( O(N + t_s) = O(N \log^2 N) \) messages are sent in expectation and whp.
- After stabilization, at most one message is sent per round
- Pass token messages contribute the most to message complexity
Conclusion

• Défago’s algorithm for self-stabilizing leader election is highly message efficient

➢ Still state-of-the-art and superior to the competition

• Main reasons for message efficiency:
  – Clever usage of tree traversal
  – Self-stabilization achieved by powerful fault detection mechanisms