Self-Stabilizing Master--Slave Token Circulation and Efficient Size-Computation in a Unidirectional Ring of Arbitrary Size

Wayne Goddard & Pradip K. Srimani
Clemson University, South Carolina
{goddard,srimani}@cs.clemson.edu

CLEMSON UNIVERSITY

Self-Stabilization

- Self-stabilization is a cost effective optimistic way of providing tolerance from *transient* faults when temporary unavailability of system is acceptable for a short period of time. *It is no longer necessary to assume a bound on the number of failures.*
- Use local knowledge to achieve global objective.
- Allow the system to start from an arbitrary initial global state without global reset or external coordination (it is difficult to power cycle the network to force it into a well defined initial state)
- Provide scalability; nodes can leave or enter the system with relative ease.
- Uniform code at each participating node (sometimes, a special node is assumed)

Problem Statement

- Recently, researchers have proposed a new model for self-stabilization. They considered the problem of **determining the number *n of nodes in a ring* using constant space.** Well, actually this is impossible, since even writing down the value *n requires O(log n)* space. So, to obtain a "space-efficient" algorithm, one node is designated that would determine the answer, but all other nodes are required to use constant space. We call such an algorithm a *master–slave algorithm*. These semi-uniform algorithms are closer to model sensor or ad hoc networks where the sensor nodes are extremely resource challenged and a base station or a cluster head needs to determine some information.

- We propose a new algorithm that computes the size of the ring at the master node in *O(n log n) time compared to O(n^3) steps taken by the existing*algorithm in [3] using the same computing paradigm.

Model of Execution

- We focus here on a *central daemon, also known* as a serial daemon. A central daemon picks a single arbitrary privileged node to move (execute the rule's action) at each step. In contrast, the *distributed daemon* taps a nonempty subset of the privileged nodes to move at each step. Our algorithms work under both daemons.
- The network has a unique master node, and all other nodes are anonymous; thus the algorithms are semi-uniform.
- We assume a *unidirectional ring network*, with the master node labeled 0 and the other nodes in order labeled 1 through $n - 1$. Any node i *has a predecessor* $i - 1$ and the master node 0 has its predecessor $n - 1$.

Master–Slave Token Circulation in Rings

- In the AB algorithm, each node has a token variable t_i that can be in one of three states, say $\mathbb{A}$, $\mathbb{B}$, or $\mathbb{C}$. The rule for any slave node *i is trivial: If* $t_i = t_{i-1}$, *then* $t_i = t_{i-1}$. *When node i changes its t-value, we say that* it has received the token, and any desired actions are performed then.

- AB Algorithm:

  Master Node 0**:**

  if $t_0 = t_{n-1}$ *then* change t_0 *according to formula Update*

  Slave Node *i***:**

  if $t_i = t_{i-1}$ *then* $t_i = t_{i-1}$.

- *Update Rule:*

  Randomized Update**:**

  Let $t \in \{\mathbb{A}, \mathbb{B}, \mathbb{C}\}$ – *t uniformly at random.*

Properties of AB Algorithm

Lemma 1 (a) The AB algorithm cannot terminate. (b) The number of privileged nodes cannot increase.

Lemma 2 If the AB algorithm does not converge to a legal final configuration, then the sequence of distinct values of t_0 is periodic.

Theorem 1 Under Randomized Update, the expected time to convergence of one circulating token is $O(n \log n)$ steps, regardless of the behavior of the daemon.

Remark: The AB algorithm handles a distributed daemon too. Indeed, if the distributed daemon does not choose all the nodes simultaneously, then in any unidirectional ring algorithm, this is equivalent to a sequence of individual moves (move the front most node first). So the only issue with considering a distributed daemon is what happens if the daemon selects all nodes simultaneously. This can, of course, only happen if every node is privileged. As argued before, that state cannot continue indefinitely.

Better Size Computation on Rings

Informal Description of Our Approach:

Our approach is based on the standard algorithm for counting a set of cells in a 1-tape Turing Machine: compute the count one bit at a time. The master node determines the ring-size one bit per round (in a right-to-left fashion) such that O(log n) circulations of the token suffice, thereby significantly reducing the execution time.

Distributed Algorithm

The concept of the distributed algorithm is as follows.

- Every slave node has an Alive bit. The master node starts by sending round a Reset token that sets every node's Alive bit to true. After that, the master sends out a Counter token with a single bit.
- The first round, this Counter token determines the parity of the number of nodes, since every node toggles the Counter bit. Furthermore, the master node sends the token out with Counter bit clear. Every node toggles the bit; but those nodes that set the bit also clear their Alive bit.
- The second time the Counter token circulates, it does the same thing, except that nodes that are Dead simply pass on the Counter token unchanged. In this way, the master node receives the parity of $\lceil n/2 \rceil$. And so on.

In log n rounds, the master node can calculate the total number of nodes. In order to get the master to know that the process is complete, one adds another "Pristine" bit to the token; this state is cleared by the first node to toggle the Counter bit. If the master node gets the Pristine state back, then it knows all nodes are dead and the algorithm is complete.

Data Structures at node i

- A 3-state token variable t_i that can have any of the three values 𝔸, 𝔹, or ℂ (this can be implemented by using only 2 bits).
- A 4-state status variable $status_i$ *that can have any* of four values ℙ*ristine*, ℝ*eset*, ℤ*ero, or* 𝕆*ne (this* can be implemented by using only 2 bits).
- Each slave node i ($i > 0$) has a one-bit Boolean flag $live_i$. We say that a slave node i is alive if $live_i$ is true; otherwise, it is dead.
- The unique master node ($i = 0$) has two integer variables *count* and *pos* to store the size of the ring.

Rule for the Master Node ($i = 0$):

if $t_0 = t_{n-1}$ then

Set t_0 = choose by Update from $\{\mathbb{A}, \mathbb{B}, \mathbb{C}\} - \{t_{n-1}\}$;

if $status_{n-1} = \mathbb{R}$ then

{ $status_0 = \mathbb{P}$; $count = 0$; $pos = 0$; }

else if $status_{n-1} = \mathbb{P}$ then

{ $status_0 = \mathbb{R}$; $count\,{+}{+}$; }

else if $status_{n-1} = \mathbb{Z}$ then

{ $status_0 = \mathbb{P}$; $pos\,{+}{+}$; }

else if $status_{n-1} = \mathbb{O}$ then

{ $status_0 = \mathbb{P}$; $count {+}= 2^{pos}$; $pos\,{+}{+}$; }

Rule for a Slave Node ($i > 0$):

if $t_i \neq t_{i-1}$ then

Set $t_i = t_{i-1}$;

if $status_{i-1} = \mathbb{R}$ then

{ $status_i = \mathbb{R}$; $live_i = \text{true}$; }

else if $status_{i-1} = \mathbb{P}$ then

{ if $live_i$ then {$status_i = \mathbb{O}$; $live_i = \text{false}$;} else $status_i = \mathbb{P}$; }

else if $status_{i-1} = \mathbb{Z}$ then

{ if $live_i$ then {$status_i = \mathbb{O}$; $live_i = \text{false}$; } else $status_i = \mathbb{Z}$; }

else if $status_{i-1} = \mathbb{O}$ then

{ if $live_i$ then $status_i = \mathbb{Z}$; else $status_i = \mathbb{O}$; }

Master–Slave Algorithm `RING` to compute the size of an Unidirectional Ring

Conclusions

- Starting from an arbitrary legal configuration, the integer variable count at the master node will contain the size n of the ring in at most $2(\lceil \log_2 n \rceil + 1)$ rounds (O(n log n) steps).
- Our algorithm computes the size of the ring at the master node in *O(n log n) time compared to O(n^3) steps taken by a recent* algorithm in [3] using the same computing paradigm.
- It seems likely that one should be able to obtain master– slave algorithms for other problems in networks.

Thank You Very Much

Questions?