Self-Stabilizing Token Distribution with Constant-Space for Trees

Yuichi Sudo¹, Ajoy K. Datta², Lawrence L. Larmore², Toshimitsu Masuzawa¹

1. Osaka University, Japan
2. The University of Nevada, Las Vegas, USA
Token Distribution Problem

- Originally defined by Peleg and Upfal in 1989 [4]

- Initially: *n* tokens are arbitrarily distributed (*n*: #nodes)
- GOAL: Each node has exactly one token

- Constraints: A node holds at most *l* tokens at any time
  (*l*: token space capacity at a node)

Generalized Token Distribution

- Initially: \( nk \) tokens are arbitrarily distributed (\( n: \#\text{nodes} \))
- GOAL: Each node has exactly \( k \) tokens

Case of \( k = 2 \)

- Constraints: A node holds at most \( l \) tokens at any time
  \((l: \text{token space capacity at a node})\)
Self-stabilizing (SS) Token Distribution

- Initially: **Any number of tokens** are *arbitrarily* distributed. Each process has an arbitrary state.
- GOAL: Each node has **exactly** $k$ **tokens** ($n$: #nodes)

Constraints: A node holds **at most** $l$ **tokens** at any time ($l$: token space capacity at a node)

Assumption
- **Rooted tree** networks
- The root can push/pull tokens to/from the **external store**
- Each node knows $k$
Model

- Asynchronous trees
- Link register model
  - Two registers at each link, one for each direction
  - Nodes can communicate only through the registers
- Token space capacity
  - A node has a token space for at most $l$ tokens
  - A link register has a token space for at most one token
  - Tokens are transferred one by one
Efficiency metrics

• Convergence time
  • Evaluated in asynchronous rounds

• Number of **redundant** token moves
  • #(token moves) – (the optimum number of token moves)
  • i.e., the number of **unnecessary** token moves

• Work space of a node and a register
  • Space for all variables **other than tokens**
Our Contributions: SS token distribution

<table>
<thead>
<tr>
<th></th>
<th>convergence time</th>
<th>#(redundant token moves)</th>
<th>work space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>node</td>
</tr>
<tr>
<td>Base</td>
<td>$O(nl)$</td>
<td>$O(nh\epsilon)$</td>
<td>0</td>
</tr>
<tr>
<td>SyncDist</td>
<td>$O(nl)$</td>
<td>$O(nh)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>PIFDist</td>
<td>$O(nhl)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Lower bounds</td>
<td>$\Omega(nl)$</td>
<td>$\Omega(n)$</td>
<td>-</td>
</tr>
</tbody>
</table>

- Unfair daemon
- **Constant work space**
- Base, SyncDist: **optimal convergence time**
- SyncDist: improved #(redundant token moves)
- PIFDist: **optimal #(redundant token moves)**
  at the expense of increased convergence time

$$\epsilon = \min(k, l - k)$$
Strategy

• Each node $v$ determine whether or not its subtree $T_v$ has excess/shortage of tokens (i.e., more than / less than $k \cdot |T_k|$ tokens)

  • Excess $\rightarrow$ $v$ sends a token to parent $p_v$
  • Shortage $\rightarrow$ $v$ asks $p_v$ to send a token to $v$
  • Balanced $\rightarrow$ Do nothing

• Root resolves the excess or shortage of the whole tree by pushing to or pulling from the external store

If every node **always** detects the excess/shortage correctly,
  • redundant token moves never happen
  • token distribution is eventually achieved
Strategy for **constant work space**

- Each node tries to detect the excess/shortage of its subtree **without counting #tokens**
- **Use 6 symbols** to inform the parent of excess/shortage in $T_v$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>excess</td>
</tr>
<tr>
<td>0⁺</td>
<td>excess or balanced</td>
</tr>
<tr>
<td>0</td>
<td>balanced</td>
</tr>
<tr>
<td>0⁻</td>
<td>shortage or balanced</td>
</tr>
<tr>
<td>-1</td>
<td>shortage</td>
</tr>
<tr>
<td>⊥</td>
<td>unsure</td>
</tr>
</tbody>
</table>

- Node $v$ sends a token to its parent if its symbol is $+1$
- Parent $p_v$ sends a token to $v$ if $v$’s symbol is $-1$
Determine a Symbol for a **Leaf** Node

- Each leaf can easily determine its symbol because its subtree consists of only itself.

**Case of** $k = 2$

- $+1$ tokens: $> k$ tokens
- $0$ tokens: Exactly $k$ tokens
- $-1$ tokens: $< k$ tokens
Determine a Symbol for a **Non-Leaf** Node

- **> k tokens** in $v$
  - All children outputs $+1$, $0^+$, or $0$

- **Exactly k tokens** in $v$
  - All children outputs $+1$, $0^+$, or $0$

- **Exactly k tokens** in $v$
  - All children outputs $0$

- **Exactly k tokens** in $v$
  - All children outputs $0$, $0^-$, or $-1$

- **< k tokens** in $v$
  - All children outputs $0$, $0^-$, or $-1$

- **Otherwise**
Good Properties

- A node is called **consistent** when its symbol does not contradict to the actual excess/shortage in its sub-tree.
- A configuration $\gamma$ is **consistent** if every node is consistent in $\gamma$.

<table>
<thead>
<tr>
<th>+1</th>
<th>excess</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+</td>
<td>excess or balanced</td>
</tr>
<tr>
<td>0</td>
<td>balanced</td>
</tr>
<tr>
<td>0−</td>
<td>shortage or balanced</td>
</tr>
<tr>
<td>-1</td>
<td>shortage</td>
</tr>
<tr>
<td>⊥</td>
<td>unsure</td>
</tr>
</tbody>
</table>

$h$: the height of the tree

- Starting from any configuration, every execution reaches a consistent configuration after $O(h)$ rounds.
- Thereafter, redundant token moves never happen 😊
- Furthermore, at least one token moves in every $O(1)$ rounds $\rightarrow$ Token distribution is achieved within $O(nhl)$ rounds.
- Careful analysis proves that $O(nl)$ rounds are enough.
$\Omega(nl)$ rounds are necessary

Must send $\Omega(n(l-k))$ tokens

Must send $\Omega(nk)$ tokens

Both trees has $\Omega(n)$ nodes

Every node has $l$ tokens

Every node has no token

$\Omega(\max(n(l-k), nk)) = \Omega(nl)$ rounds are required to achieve token distribution
Three algorithms

• Base
  • Simply implement the above strategies
  • **Optimal conv. time** $O(nl)$, **redund. token moves** $O(nh\epsilon)$
    $h$: tree height, $\epsilon = \min(k, l - k)$

• SyncDist
  • Use a **self-stabilizing (loose-)synchronizer** (Datta 15) so that each node can send $O(h)$ tokens in $O(h)$ rounds
  • **Optimal conv. time** $O(nl)$, **redund. token moves** $O(nh)$

• PIFDist
  • Use a **self-stabilizing PIF** (Bui 07) so that each node can send $O(1)$ tokens in $O(h)$ rounds
  • **Conv. time** $O(nlh)$, optimal **redund. token moves** $O(n)$
Conclusions: SS token distribution

<table>
<thead>
<tr>
<th></th>
<th>convergence time</th>
<th>#(redundant token moves)</th>
<th>work space</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>node</td>
<td>link</td>
</tr>
<tr>
<td>Base</td>
<td>$O(nl)$</td>
<td>$O(nh \varepsilon)$</td>
<td>0</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>SyncDist</td>
<td>$O(nl)$</td>
<td>$O(nh)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>PIFDist</td>
<td>$O(nhl)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Lower bounds</td>
<td>$\Omega(nl)$</td>
<td>$\Omega(n)$</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Unfair daemon
- **Constant work space**
- Base, SyncDist: optimal convergence time
- SyncDist: improved $#$ (redundant token moves)
- PIFDist: optimal $#$ (redundant token moves) at the expense of increased convergence time

$\varepsilon = \min(k, l - k)$
Model

- Asynchronous rooted tree
- Link register model
  - Two registers at each link, one for each direction
  - Nodes can communicate only through the registers
- Token space capacity

Case of $k = 2$

Transferred one by one
Token moves

• Each node $\nu$ sends tokens following the two rules
  • Finds a child with symbol $-1 \rightarrow$ Sends a token to the child
  • $\nu$’s symbol is $+1 \rightarrow$ Sends a token to $\nu$’s parent