Self-Stabilizing K-out-of-L Exclusion on Tree Networks

Stéphane Devismes, VERIMAG

Joint work with:
- Ajoy K. Datta (Univ. Of Nevada)
- Florian Horn (LIAFA)
- Lawrence L. Larmore (Univ. Of Nevada)
Roadmap

• Recall on Self-stabilization

• Definition of the problem

• The solution

• Conclusion and perspectives
Roadmap

• Recall on Self-stabilization

• Definition of the problem

• The solution

• Conclusion and perspectives
Self-Stabilization: Closure + Convergence
Self-Stabilization: Closure + Convergence

States of the System
Self-Stabilization: Closure + Convergence

States of the System

Illegitimate States

Legitimate States
Self-Stabilization: Closure + Convergence

States of the System
Self-Stabilization: Closure + Convergence

States of the System

Closure
Self-Stabilization: Closure + Convergence

States of the System

May 18th 2009

SHAMAN Meeting

4
Self-Stabilization: Closure + Convergence

States of the System
Self-Stabilization: Closure + Convergence

States of the System

May 18th 2009

SHAMAN Meeting
Self-Stabilization: Closure + Convergence

States of the System

Convergence

May 18th 2009   SHAMAN Meeting
Self-Stabilization: Closure + Convergence

States of the System

May 18th 2009

SHAMAN Meeting

4
Property: Tolerance to Transient Faults
Roadmap

• Recall on Self-stabilization

• Definition of the problem

• The solution

• Conclusion and Perspectives
$K$-out-of-$L$ Exclusion [Raynal, 91]

- $L$ resource units

- Requests from 1 to $K$ resource units ($K \leq L$)
K-out-of-L Exclusion

• 3 property to ensure:
  
  – Safety
  
  – Fairness
  
  – Efficiency
Safety

• At any time:
  – Each resource unit is used by at most one process
  – Each process uses at most $K$ resource units
  – At most $L$ resource units are available
Fairness

• Each request (of at most $K$ units) is satisfied in finite time

(_i.e._ the process then uses the resource units it holds in a special section of code called *critical section*)
Efficiency
Efficiency

• « As many requests as possible must be satisfied simultaneously »
Efficiency

• « As many requests as possible must be satisfied simultaneously »
Efficiency

• « As many requests as possible must be satisfied simultaneously »

• More formally: \((K,L)\)-Liveness
(K,L)-Liveness
(K,L)-Liveness
Waiting Time

« The maximum number of time, all other processes can enter in the critical section before some process $p$, starting from the moment requests the critical section »
Roadmap

• Recall on Self-stabilization
• Definition of the problem
• The solution
• Conclusion and perspectives
Model
Model

- Message-passing
- Bounded process memories
Model

- Message-passing
- Bounded process memories
- FIFO bidirectional links
Model

• Message-passing
• Bounded process memories
• FIFO bidirectional links
• Topology : oriented tree
Model

• Message-passing
• Bounded process memories
• FIFO bidirectional links
• Topology : oriented tree

• Necessary condition [Gouda-Multari, 91]:
  – Bounded link capacity (C\text{MAX})
Approach

- Token-based (resource units = tokens)

- Modular:
  - Non Self-stabilizing $K$-out-of-$L$ Exclusion
  - Self-stabilizing Controller
DFS circulation
First idea:

\[ K\text{-out-of-}L \text{ Exclusion} = L\text{-circulation} \]
First idea:

\( K\text{-out-of-}L \text{ Exclusion} = L\text{-circulation} \)
First idea:

$K$-out-of-$L$ Exclusion = $L$-circulation
First idea:

$K$-out-of-$L$ Exclusion = $L$-circulation
First idea:

$K$-out-of-$L$ Exclusion = $L$-circulation

$\{0, 2\}$
First idea:

\( K\text{-out-of-}L \) Exclusion = \( L\text{-circulation} \)
First idea:

\( K\text{-out-of-}L\) Exclusion = \(L\)-circulation

\[
\begin{align*}
\{0,2,2\} \\
\text{<CS>}
\end{align*}
\]
First idea:

\[ K\text{-out-of-}L \text{ Exclusion} = L\text{-circulation} \]
Deadlocks
Deadlocks
Solution

• Circulation of a special token: the *pusher*

• Upon receiving the *pusher*: a node releases all its resource tokens if its request is not satisfied
Livelock
Livellock
Livelock

Diagram:

- Node 0
- Node 1
- Node 2

Connections:
- 0 → 1
- 0 → 2
- 1 → 0

Note: The diagram illustrates a live lock scenario with nodes and connections.
Livelock
Livelock
Livelock
Solution

• Circulation of a *Priority Token*:
  
  – A requesting process keeps the *Priority Token* while its request is unsatisfied
  
  – The *Priority Token* cancels the effect of the *Pusher Token*
Self-Stabilization: The Controller
Self-Stabilization: The Controller

- After transient faults:
  - Some tokens may have disappeared
  - Some tokens may have been duplicated
Self-Stabilization: The Controller

• After transient faults:
  – Some tokens may have disappeared
  – Some tokens may been duplicated

• Solution: a Controller Token counts and regulates the number of tokens
Example: Count the Resource Tokens
Example: Count the *Resource Tokens*
Example: Count the *Resource Tokens*
Example: Count the Resource Tokens

\[ \{0, 1\} \]
Example: Count the *Resource Tokens*
Example: Count the Resource Tokens

\begin{center}
\begin{tikzpicture}
  \node (a) at (0,0) {$a$};
  \node (b) at (-1,1) {$b$};
  \node (c) at (-1,-1) {$c$};
  \node (d) at (1,1) {$d$};
  \node (e) at (1,-1) {$e$};
  \node (r) at (0,2) {$r$};
  \node (start) at (-2,0) {START};

  \path[->,thick]
    (a) edge node [above] {$0$} (b)
    (b) edge node [above] {$1$} (c)
    (c) edge node [above] {$0$} (a)
    (a) edge node [above] {$2$} (r)
    (r) edge node [above] {$1$} (d)
    (d) edge node [above] {$0$} (e)
    (e) edge node [above] {$0$} (d)
    (d) edge node [above] {$3$} (g)
    (g) edge node [above] {$0$} (r)
    (r) edge node [above] {$0$} (start);

  \filldraw[red] (a) circle (0.1); \node at (a) {$1$};
  \filldraw[black] (d) circle (0.1); \node at (d) {$1$};
  \node at (r) {$\{0,1\}$};
\end{tikzpicture}
\end{center}
Example: Count the Resource Tokens
Example: Count the *Resource Tokens*
Example: Count the *Resource Tokens*

- At the end of the traversal:
  - The number of token is known
    - Too much tokens: RESET
    - Lack of tokens: Creation at the root
How to stabilize the Controller?

- Implemented as a Self-Stabilizing DFTC using the Varghese Counter Flushing
Self-Stabilizing DFTC

\[ r \]
Self-Stabilizing DFTC
Self-Stabilizing DFTC
Self-Stabilizing DFTC

TokenHolder:
• Receive *MesVal* from a child and *MyVal* = *MesVal*
• Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*
Self-Stabilizing DFTC

TokenHolder:
- Receive \( MesVal \) from a child and \( MyVal = MesVal \)
- Receive \( MesVal \) from Parent and \( MyVal \neq MesVal \)
Self-Stabilizing DFTC

TokenHolder:
- Receive $MesVal$ from a child and $MyVal = MesVal$
- Receive $MesVal$ from Parent and $MyVal ≠ MesVal$

May 18th 2009  SHAMAN Meeting  28
Self-Stabilizing DFTC

TokenHolder:
- Receive $MesVal$ from a child and $MyVal = MesVal$
- Receive $MesVal$ from Parent and $MyVal \neq MesVal$
Self-Stabilizing DFTC

TokenHolder:
• Receive \( \text{MesVal} \) from a child and \( \text{MyVal} = \text{MesVal} \)
• Receive \( \text{MesVal} \) from Parent and \( \text{MyVal} \neq \text{MesVal} \)
Self-Stabilizing DFTC

TokenHolder:

- Receive $MesVal$ from a child and $MyVal = MesVal$
- Receive $MesVal$ from Parent and $MyVal \neq MesVal$
Self-Stabilizing DFTC

TokenHolder:
- Receive $MesVal$ from a child and $MyVal = MesVal$
- Receive $MesVal$ from Parent and $MyVal \neq MesVal$
Self-Stabilizing DFTC

TokenHolder:
- Receive $MesVal$ from a child and $MyVal = MesVal$
- Receive $MesVal$ from Parent and $MyVal \neq MesVal$
Self-Stabilizing DFTC

TokenHolder:
- Receive MesVal from a child and MyVal = MesVal
- Receive MesVal from Parent and MyVal ≠ MesVal

Stabilize using (n-1)(2CMAX+1)+1 values
Roadmap

• Recall on Self-stabilization

• Definition of the problem

• The solution

• Conclusion and perspectives
Conclusion
Conclusion

• Waiting Time: $L^*(2n-3)^2$
Conclusion

• Waiting Time: \( L^*(2n-3)^2 \)

• Oriented Tree -> Arbitrary Rooted Network
  (Huang-Chen BFS Tree)
Conclusion

• Waiting Time: $L^*(2n-3)^2$

• Oriented Tree -> Arbitrary Rooted Network
  (Huang-Chen BFS Tree)

• Bounded/Unbounded Link Capacity
  (Katz & Perry)
Perspectives
Perspectives

• Compute the convergence time
Perspectives

• Compute the convergence time

• Enhance the waiting time?
Perspectives

- Compute the convergence time
- Enhance the waiting time?
- Reactive solution?
Perspectives

- Compute the convergence time
- Enhance the waiting time?
- Reactive solution?
- Fault-Containment?
Thank you!
(\(K,L\))-Liveness

- Let \(V\) be the set of processes
- Let \(I\) be a subset of processes that execute their critical section forever (in particular they hold some resource units forever)
- Let \(\alpha\) be the number of resource units held by \(I\)
- Let \(R\) be the subset of \(V - I\) such that any process in \(R\) is a requestor
- Let \(r_{\text{max}}\) by the maximal request of a process in \(R\)
- If \(R \neq \emptyset\) and \(r_{\text{max}} \leq L - \alpha\) then at least one member of \(R\) eventually satisfies its request