A Self-stabilizing PIF Algorithm for Educated Unique Process Selection

Oday Jubran(✉) and Oliver Theel

Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
{jubran,theel}@informatik.uni-oldenburg.de

Abstract. Many applications, methods, and models are based on underlying self-stabilizing mutual exclusion algorithms. The efficiency of such applications is correlated to the efficiency of the algorithms, which reflects a quick recovery from failures, and a fast service time. In this work, we focus on a property correlated to this field, namely *Educated Selection*, which indicates that the selection of processes to be granted unique privilege is deterministic and based on evaluating the local states of processes, or the global configuration. We present a self-stabilizing Propagation of Information with Feedback (PIF) algorithm for trees using the shared memory model. The algorithm exploits the PIF technique for achieving fast educated unique process selection.

Keywords: Self-stabilization · Propagation of information with feedback (PIF) · Mutual exclusion · Educated selection

1 Introduction

Self-stabilization [1] ensures that a system's desired behavior is eventually obtained and never voluntarily violated regardless of the system's initial behavior. Self-stabilization was considered in distributed systems using the *shared memory model*, where a process is enabled to execute an action if a condition over the registers, visible to the process, is satisfied. Running an action changes the registers' values, potentially enabling other processes to run actions.

The mutual exclusion problem was considered in self-stabilization, e.g. [2]. Mutual exclusion comprises: (1) a safety property that at most one process is granted a privilege in each state, and (2) a liveness property that each process is privileged infinitely often. The second property is usually referred as *fairness*.

Mutual exclusion does not necessarily require the process selection for granting the privilege to be deterministic. However, for some systems, it is useful if the process selection is based on local or global criteria, e.g. energy measurements or QoS indicators, towards increasing the performance of the systems. We denote such deterministic process selection as an *educated selection*.

This work was partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, http://www.avacs.org/).

A. Bouajjani and H. Fauconnier (Eds.): NETYS 2015, LNCS 9466, pp. 485–489, 2015.
DOI: 10.1007/978-3-319-26850-7_36

The *Propagation of Information with Feedback* (PIF) [3] is a useful token-passing approach for educated selection, in which a process sends a wave of tokens, and receives a feedback. In this work, we apply a PIF approach for educated unique process selection by extending an earlier approach [4]. We present a self-stabilizing PIF algorithm for trees using the shared memory model. The algorithm performs reasonably fast educated unique process selection based on local or global criteria in (a)synchronous environments.

Outline. Section 2 gives the basic notation. Section 3 presents our self-stabilizing PIF algorithm. Section 4 draws a conclusion.

2 Notation

We consider a tree topology using the shared memory model. A **tree** $T = (P, E)$ is a set of *processes* P and *edges* $E \subseteq P \times P$. The *parent* of a process p is denoted by θ_p, and the children of p are denoted by C_p. The root is denoted by *root*, and the set of leaves (resp. inner processes) in P is denoted by *Leaves* (resp. *Inner*). The number of all processes is n. Each **process** has variables, constants, and a unique id in $\{0, ..., n-1\}$: A process p of id i is denoted by p_i, and a variable v of p_i is referred by $p_i.v$. A **state** σ is a valuation of the variables of some process p. A **configuration** is a vector $[\sigma_0, ..., \sigma_{n-1}]$ of the states of all processes.

3 Algorithm

We present an algorithm, which is based on an extended scheme of the well-known mutual exclusion algorithm of Dijkstra [2]. Section 3.1 presents the algorithm for educated selection based on local states, and Sect. 3.2 extends the algorithm for educated selection based on global configurations

3.1 Educated Selection Based on Local States

Algorithm 1 shows the algorithm. Each process owns the following variables: (1) $up \in \mathbb{B}$ and $x \in \mathbb{B}$. We assert that $up = \top$ for the root, and $up = \bot$ for each leaf. (2) $\ell \in \{0, ..., n-1\}$: this variable stores a process's id to direct particular tokens to selected processes. We say that a process p *points* to a process p_i when $p.\ell = i$. (3) We abstract the local criteria of each process p_i by a variable $m_i \in \mathbb{R}$, such that p_i is selected only if the value of $p_i.m$ is the maximum among all other processes. We assume that the value of $p_i.m$ is updated by p_i independent of the algorithm, and is returned by the function $update_m()$.

We define the function $choose : 2^P \Rightarrow \{0, ..., n-1\}$ as follows: given a subset $P' \subseteq P$, the function returns the id of a process that has the maximum value of m among P'. A process runs $critSection()$, if it is privileged.

The stable behavior of Algorithm 1 is an infinite repetition of two PIF cycles, where in each cycle, the root propagates a token to all processes, and receives a feedback from all processes, yielding four types of tokens.

Algorithm 1. Algorithm for a Process p in a Topology $T = (P, E)$

Constants: $id \in \{0, ..., n-1\}$

Variables: $x \in \mathbb{B}$, $up \in \mathbb{B}$, $\ell \in \{0, ..., n-1\}$, $m \in \mathbb{R}$

Assertions: $root.up = \top \wedge \forall\, q \in Leaves \bullet q.up = \bot$

Tokens

$token_1 : \theta_p.x \neq x \wedge \neg x$ % Search Token
$token_2 : up \wedge x \wedge \forall\, ch \in C_p \bullet ch.x = x \wedge \neg ch.up$ % Feedback Token
$token_3 : \theta_p.x \neq x \wedge x$ % Execute Token
$token_4 : up \wedge \neg x \wedge \forall\, ch \in C_p \bullet ch.x = x \wedge \neg ch.up$ % Complete Token

Functions

$update_m()$ $:= \{v \in \mathbb{R} \mid v$ is independent of the algorithm$\}$
$choose(P' \subseteq P) := \{i \mid p_i \in P' \wedge \forall\, q \in P' \bullet p_i.m = \max(q.m)\}$
$critSection()$: Access Critical Section

Guarded Commands (c_i : $guard \longrightarrow action$)

Root Sub-Algorithm

1 : $token_2 \longrightarrow \ell := choose(\{p\} \cup C_p);\ m := p_\ell.m;\ x = \neg x;$
2 : $token_4 \wedge \ell = id \longrightarrow$ ***critSection()***; $m := update_m();\ x = \neg x;$ % Privileged
3 : $token_4 \wedge \ell \neq id \longrightarrow m := update_m(); x = \neg x;$

Inner Process Sub-Algorithm

4 : $token_1 \longrightarrow m := update_m();\ up := \top;\ x := \neg x;$
5 : $token_2 \wedge \neg token_3 \longrightarrow \ell := choose(\{p\} \cup C_p);\ m = p_\ell.m;\ up := \bot;$
6 : $token_3 \wedge \theta_p.\ell = id \wedge \ell = id \longrightarrow$ ***critSection()***; $up := \top;\ x := \neg x;$ % Privileged
7 : $token_3 \wedge \theta_p.\ell = id \wedge \exists\, q \in C_p \bullet p.\ell = q.id \longrightarrow up := \top;\ x := \neg x;$
8 : $token_3 \wedge \neg(\theta_p.\ell = id \vee \exists\, q \in C_p \bullet q.\ell = q.id) \longrightarrow \ell := id;\ up := \top;\ x := \neg x;$
9 : $token_4 \wedge \neg token_1 \longrightarrow up := \bot;$

Leaf Sub-Algorithm

10 : $token_1 \longrightarrow m := update_m();\ \ell := id;\ x := \neg x;$
11 : $token_3 \wedge \theta_p.\ell = id \longrightarrow$ ***critSection()***; $x := \neg x;$ % Privileged
12 : $token_3 \wedge \theta_p.\ell \neq id \longrightarrow\ x := \neg x;$

Algorithm 2. Extending Algorithm 1

Additional Variables

$snapShot = [k_0, ..., k_{n-1}]$, where $k_i \in \mathbb{R}$ for $0 \leq i \leq n-1$

Extended Functions

$update_m([k_0, ..., k_{n-1}]) = \{v \in \mathbb{R} \mid v$ is dependent of $[k_0, ..., k_{n-1}]\}$

Extended Guarded Commands $(2, 3, 4, 6, 9, 10, 11)$

$2' : ... \longrightarrow$ ***critSection()***; $snapShot.k_{id} = k;\ m := update_m(snapShot);\ x = \neg x;$
$3' : ... \longrightarrow snapShot = p_\ell.snapShot;\ m := update_m(snapShot);\ x := \neg x;$
$4' : ... \longrightarrow snapShot = \theta_p.snapShot;\ m := update_m(snapShot);\ up := \top; x := \neg x;$
$6' : ... \longrightarrow$ ***critSection()***; $snapShot.k_{id} := k;\ up := \bot;\ x := \neg x;$
$9' : ... \longrightarrow snapShot := p_\ell.snapShot;\ up := \bot;$
$10' : ... \longrightarrow snapShot = \theta_p.snapShot;\ m := update_m(snapShot);\ \ell := id;\ x := \neg x;$
$11' : ... \longrightarrow$ ***critSection()***; $snapShot.k_{id} = k;\ m := update_m(snapShot);\ x := \neg x;$

First PIF Cycle
$token_1 \downarrow$: the root sends $token_1$. When a process p receives $token_1$, p updates m, and forwards the token to its children (c_4), until $token_1$ reaches the leaves. Each leaf l_i updates $l_i.m$, and sends $token_2$ to its parent (c_{10}).
$token_2 \uparrow$: when a process p receives $token_2$, p points to a process q, where $q \in \{p\} \cup C_p$ and q has the maximum value of m among p and its children (c_5). Then, p copies $q.m$, and switches the value of $p.up(c_5)$. With this action, each process p eventually points to a path that leads to the process with the original maximum value of m, after copying it. Eventually, $token_2$ reaches the root, and the root starts the second PIF cycle (c_1).

Second PIF Cycle
$token_3 \downarrow$: the root sends $token_3$. If a process p receives $token_3$, one of three possible cases exists:

- **Case (1)** represented by commands c_6, c_{11}: if θ_p points to p and p points to itself, then p has a privilege. p runs *critSection*() and forwards the token.
- **Case (2)** represented by c_7: if θ_p points to p and p points to one of its children q, this implies that the selected process exists in the subtree rooted by q. p passes $token_3$, while keeping $p.\ell = q.id$.
- **Case (3)** represented by c_8, c_{12}: if θ_p is not pointing to p, or p is neither pointing to itself nor to one of its children, then there is no selected process in the maximal subtree rooted by p. p sets ℓ to $p.id$, to prohibit any child from running *critSection*() after forwarding $token_3$.

Note that, if $token_3$ is directed to a subtree T', in which there is no selected process, then c_8 is enabled in each process in T' in the current PIF cycle.
$token_4 \uparrow$: Next, $token_4$ is forwarded to the root (c_9). The root receives $token_4$ which involves all its children. If the selected process is the root, then c_2 is enabled, the root runs *critSection*(), and sends $token_1$ to its children. Otherwise (c_3), the root simply starts a new PIF cycle.

Regarding the time complexity: (1) The algorithm guarantees unique process selection in d rounds, where d is the tree depth. (2) The algorithm guarantees that after at most $3d$ rounds, each PIF cycle lasts $2d$ rounds, and within any two subsequent PIF cycles, exactly one process is privileged.

3.2 Educated Selection Based on Global Configurations

We extend Algorithm 1 for educated selection based on configurations. We show the extension in Algorithm 2. In Algorithm 1, the update value of m, returned by $update_m()$, is based on the local state. In Algorithm 2, the value of m is updated according to the global configuration. This indicates that each process should know the configuration. We abstract the configuration by the vector *snapShot*, owned by each process, and defined as follows: $snapShot = [k_0, ..., k_{n-1}]$, where $k_i \in \mathbb{R}$, for $0 \leq i \leq n-1$, is the relevant evaluation of the local state of p_i. Now, each process p updates $p.m$ according to the value of $p.snapShot$.

The extended commands from Algorithm 1 are c_2-c_4, c_6, c_9-c_{11}. With the extension, the stable behavior is as follows: in the first PIF cycle, when a process receives $token_1$, it copies the parent's snapshot, and updates m according to the snapshot ($c_{4'}, c_{10'}$). With this action, a copy of the snapshot reaches each process. The remainder of the first PIF cycle continues normally. In the second PIF cycle, the selected process runs *critSection*(), and modifies its snapshot based on the new value of k ($c_{6'}, c_{11'}$). Next, the parent of p copies the new snapshot, and forwards it to the root ($c_{9'}$). $c_{2'}$ and $c_{3'}$ concern extended root commands.

In the above behavior, it is assumed that the snapshot sent by the root matches the values of k of all processes. If there is an incorrect value of some k, the snapshot is said to be *inconsistent*. For inconsistent snapshots: we say that a snapshot *snap* is *highlighted* iff it contains at least one *null* value of some k. We also say that *snap* is empty if it contains only *null* values.

The snapshot inconsistency is corrected in the first PIF cycle: (1) When the root propagates $token_1$ with an inconsistent *snapShot*, there exists a process p_j such that $p_j.k$ is not equal to $snapShot.k_j$. Eventually, p_j receives $token_1$. (2) When p_j copies the snapshot, p_j checks if there is an inconsistency, or if $\theta_p.snapShot$ is empty. In both cases, p_j sets its snapshot empty. (3) Next, all processes in the maximal subtree rooted by p_j set their snapshots empty, analogous to step 2, since $token_1$ reaches every process. (4) Now, starting from the leaves, for each process p that receives $token_2$, if p recognizes a highlighted snapshot in one of its children or itself, then p creates a new snapshot by merging the snapshots of its children, and adding its value of k. Now, the snapshot of p contains correct values of all processes in the subtree rooted by p, and *null* values for the processes that are not in the subtree. (5) After the root receives a feedback token ($token_2$), it merges the new snapshots, yielding a correct one.

4 Conclusion

We presented a self-stabilizing PIF algorithm for educated unique process selection for trees using the shared memory model. The algorithm ensures that a process is selected to execute an action only if it is distinguished from other processes according to some criterion, and the criterion is based on the local state of the selected process. We denote the criterion as whether or not a particular number value of a process is maximal among all processes. We extended the algorithm for selecting processes based on a global configuration by propagating a snapshot of the local states of all processes.

References

1. Dolev, S.: Self-Stabilization. The MIT Press, Cambridge (2000)
2. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM **17**(11), 643–644 (1974)
3. Segall, A.: Distributed network protocols. IEEE Trans. Inf. Theory **29**(1), 23–34 (1983)
4. Jubran, O., Theel, O.: Exploiting synchronicity for immediate feedback in self-stabilizing PIF algorithms. In: PRDC, pp. 106–115. IEEE (2014)