A Dynamic Distributed Algorithm for Read Write Locks

Soumeya Leila Hernane\textsuperscript{1,2,3} \quad \text{Jens Gustedt}\textsuperscript{3} \quad \text{Mohamed Benyettou}\textsuperscript{1}

\textsuperscript{1} University of Science and Technology of Oran USTO, Algeria
\textsuperscript{2} Lorraine University, France
\textsuperscript{3} AlGorille
\text{INRIA Grand-Est & LORIA}
\text{Nancy France}
Outline

1 Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2 Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3 Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4 Current and future work
Outline

1. Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2. Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3. Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4. Current and future work
Outline

1. Motivation
   - Mutual exclusion algorithms
     - The bases of Naimi-Trehel Algorithm [5]
     - Concurrency

2. Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3. Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4. Current and future work
Mutual exclusion algorithms

- Permission based (Lamport [2], Maekawa [3], Ricart Agrawala [7])
- The possession of a token which is passed between nodes ([4], Raymond [6]).
- Naimi-Trehel [4] based on path reversal is the benchmark for mutual exclusion.
- Symbol of critical section → Cs
Outline

1 Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2 Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3 Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4 Current and future work
Motivation

The bases of Naimi-Trehel Algorithm [5]

Mutual exclusion

Naimi-Trehel Algorithm

- Logical dynamic tree structure.
- Root of the tree → The last that requested the token.
- Distributed queue of waiting requests (FIFO strategy).

Contributions

- Allow departure of processes.
- Include shared token for successive readers.
Mutual exclusion

Naimi-Trehel Algorithm

- Logical dynamic tree structure.
- Root of the tree → The last that requested the token.
- Distributed queue of waiting requests (FIFO strategy).

Contributions

- Allow departure of processes.
- Include shared token for successive readers.

Soumeya Leila Hernane (INRIA)
Mutual exclusion

Naimi-Trehel Algorithm

- Logical dynamic tree structure.
- Root of the tree → The last that requested the token.
- Distributed queue of waiting requests (FIFO strategy).

Contributions

- Allow departure of processes.
- Include shared token for successive readers.
Mutual exclusion

Naimi-Trehel Algorithm

- Logical dynamic tree structure.
- Root of the tree → The last that requested the token.
- Distributed queue of waiting requests (FIFO strategy).

Contributions

- Allow departure of processes.
- Include shared token for successive readers.
Outline

1 Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2 Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3 Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4 Current and future work
Concurrent requests in Naimi-Trehel’s Algorithm

Initial state
Motivation
Concurrency

Concurrent requests in Naimi-Trehel’s Algorithm

$p_3$ requests the token
Concurrent requests in Naimi-Trehel’s Algorithm

\[ p_2 \text{ and } p_5 \text{ request the token} \]
Concurrent requests in Naimi-Trehel’s Algorithm

$p_2$ and $p_5$ in transit
Concurrent requests in Naimi-Trehel’s Algorithm

$p_5$ in transit
Concurrent requests in Naimi-Trehel’s Algorithm

Completion of requests
Outline

1. Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2. Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3. Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4. Current and future work
1 Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2 Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3 Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4 Current and future work
Dynamic Mutual Exclusion Algorithm

New requirements

New variables

- \{p_1, \ldots, p_{i-1}, p_i, \ldots, p_n\} a finite set of processes
- Waiting queue of received requests
- Ack, boolean \(\rightarrow\) End of request
- Children, array of processes that have \(p_i\) as parent
- Doubly linked Next, Previous list rather than Next chain

Avoid overlapping of operations

- Either \(p_i\) sends, either it receives request
- From parent to parent, \(p_i\) sends request until the root is reached
- \(p_i\) browses the same path and sends Ack
- Average message complexity of \(O(\log(N))\)
Dynamic Mutual Exclusion Algorithm

New requirements

New variables

- \( \{p_1..p_{i-1}, p_i..p_n\} \) a finite set of processes
- Waiting queue of received requests
- \( \text{Ack} \), boolean → End of request
- \( \text{Children} \), array of processes that have \( p_i \) as \textit{parent}
- Doubly linked \textit{Next, Previous} list rather than \textit{Next} chain

Avoid overlapping of operations

- Either \( p_i \) sends, either it receives request
- From \textit{parent} to \textit{parent}, \( p_i \) sends request until the root is reached
- \( p_i \) browses the same path and sends \text{Ack}
- Average message complexity of \( O(\log(N)) \).
States for handling requests

- Let $T = (t_0, \ldots, t_n)$ an ordered sequence of times with $t_i < t_{i+1}$ referring to requests reception by $p_i$
- Let $T' = \{t'_0, \ldots, t'_i, \ldots\}$ be the set of times of the acknowledgments of those registered requests

\[
t_0 < t'_0 < \cdots t_i < t'_i < \cdots < t_n < t'_n.
\]

- Let $State(p_i, t)$ be the state of $p_i$ at time $t$

\[
STATE(p_i, t) = \text{Idle} \Rightarrow \begin{cases} 
SEND = \text{false} \\
REQ = \text{Null} \\
WAIT\_ACK = \text{false}
\end{cases}
\]
States for handling requests

- Let $T = (t_0, \ldots, t_n)$ an ordered sequence of times with $t_i < t_{i+1}$ referring to requests reception by $p_i$
- Let $T' = \{ \ldots, t'_i, \ldots \}$ be the set of times of the acknowledgments of those registered requests

\[
t_0 < t'_0 < \cdots t_i < t'_i \cdots < t_n < t'_n.
\]
- Let $\text{State}(p_i, t)$ be the state of $p_i$ at time $t$

\[
\begin{align*}
\text{STATE}(p_i, t) = \text{Busy} & \Rightarrow \\
\text{SEND} &= \text{false} \\
\text{REQ} &\neq \text{Null} \\
\text{WAIT}_\text{ACK} &= \text{true}
\end{align*}
\]
Dynamic Mutual Exclusion Algorithm

New requirements

States for handling requests

\[ T = (t_0, \ldots, t_n) \] is an ordered sequence of times with \( t_i < t_{i+1} \) referring to requests reception by \( p_i \).

Let \( T' = \{ \ldots, t'_i, \ldots \} \) be the set of times of the acknowledgments of those registered requests.

\[ t_0 < t'_0 < \cdots t_i < t'_i \cdots < t_n < t'_n. \]

Let \( \text{State}(p_i, t) \) be the state of \( p_i \) at time \( t \)

\[
\text{STATE}(p_i, t) = \text{Sending} \Rightarrow \begin{cases} 
\text{SEND} = \text{true} \\
\text{REQ} = \text{Null} \\
\text{WAIT\_ACK} = \text{false}
\end{cases}
\]
Outline

1 Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2 Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3 Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4 Current and future work
**Conditions and actions**

- **Idle** and Queue *empty*
- Children → Leaving, Sending or Busy
- waits the its *Next* and its *Previous*
- The doubly linked list *Next, Previous* is shortcut
- $p_i$ sends either its *parent* or a new root elected among its children
New settings for the exit strategy

New variables

- New boolean variables Exit and Leaving
- $p_i$ sets Exit and Leaving to true
- $p_i$ sets Leaving to false

$$\text{STATE}(p_i, t) = \text{Exiting} \Rightarrow \begin{cases} \text{SEND} = \text{false} \\ \text{REQ} = \text{Null} \\ \text{WAIT_ACK} = \text{false} \\ \text{EXIT} = \text{true} \\ \text{LEAVING} = \text{true} \end{cases}$$
Outline

1. Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2. Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3. Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4. Current and future work
Outline

1 Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2 Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3 Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4 Current and future work
New requirements for read requests

**Entering the CS**

- **Reading handler**, the head of, successive read requesters
- Request **Type**, W or R. **Next** and **Previous** exchange their types
- **Reader counter** ⇒ incremented by the **Reading handler**
- **Next writer** sets its **Previous** to the **Reading handler**
New requirements for read requests

- **Reading handler**, the head of, successive read requesters
- Request **Type**, W or R. **Next** and **Previous** exchange their types
- **Reader counter** $\Rightarrow$ incremented by the **Reading handler**
- **Next writer** sets its **Previous** to the **Reading handler**
New requirements for read requests

- Entering the CS
  - **Reading handler**, the head of, successive read requesters
  - Request **Type**, W or R. **Next** and **Previous** exchange their types
  - **Reader counter** ⇒ incremented by the **Reading handler**
  - **Next writer** sets its **Previous** to the **Reading handler**
New requirements for read requests

- **Reading handler**, the head of successive read requesters
- Request **Type**, W or R. **Next** and **Previous** exchange their types
- **Reader counter** ⇒ incremented by the **Reading handler**
- **Next writer** sets its **Previous** to the **Reading handler**
Outline

1. Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2. Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3. Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4. Current and future work
Other requirements for the dynamicity

- One reader at once can exit, the **Reading handler** is informed.
- The **Reading handler** passes the **Reader counter** to its **Next**.
- The **Next** becomes the new **Reading handler**.
- The new **Reading handler** is forwarded to the reader chain.

\[ p_1 \text{ old Reading handler} \]

\[ p_3 \text{ new Reading handler} \]
Other requirements for the dynamicity

- One reader at once can exit, the **Reading handler** is informed
- The **Reading handler** passes the **Reader counter** to its **Next**
- The **Next** becomes the new **Reading handler**
- The new **Reading handler** is forwarded to the reader chain
Outline

1. Motivation
   - Mutual exclusion algorithms
   - The bases of Naimi-Trehel Algorithm [5]
   - Concurrency

2. Dynamic Mutual Exclusion Algorithm
   - New requirements
   - Exit strategy

3. Read Write Dynamic Mutual Exclusion Algorithm
   - Read Write requests
   - Departure of readers

4. Current and future work
Current and future work

- The proof of **Safety** and **liveness** properties
- Include our extended Algorithm to the DHO API [1]
- Provide an efficient management for locking and mapping data (and subranges of it)
- Undertake experiments of that model
- Simulated mode (GRAS API of SimGrid [8])
- Large scale platform (Grid’5000 [https://www.grid5000.fr])
Soumeya Hernane, Leila, Jens Gustedt, and Mohamed Benyettou.  
Modeling and Experimental Validation of the Data Handover API.  

Leslie Lamport.  
Ti clocks, and the ordering of events in a distributed system.  

Mamoru Maekawa.  
An algorithm for mutual exclusion in decentralized systems.  

Mohamed Naimi and Michel Trehel.  
How to detect a failure and regenerate the token in the log(\(N\)) distributed algorithm for mutual exclusion.  
Current and future work

Mohamed Naimi, Michel Trehel, and André Arnold.
A log($N$) distributed mutual exclusion algorithm based on path reversal.

Kerry Raymond.
A tree-based algorithm for distributed mutual exclusion.

Glenn Ricart and Ashok K. Agrawala.
An optimal algorithm for mutual exclusion in computer networks.

Pedro Velho and Arnaud Legrand.
Accuracy study and improvement of network simulation in the SimGrid framework.
In *Simutools ’09*, pages 1–10, Brussels, Belgium, 2009. ICST.