Renaming Algorithms

Corentin Travers

LaBRI, Bordeaux

CoA Workshop, April 2019
a := 2;
// enter critical section
{ // critical section
    b:= 2 +a;
    c:= 2*b - 4;
}
// exit critical section

[Dijsktra 1965]
Resource can be accessed by at most one process at a time
Resource can be accessed by at most one process at a time
Renaming

\( \text{n processes} \)

\begin{align*}
\text{initial names} & \quad 23 & \quad 595 & \quad 1 & \quad \ldots & \quad \ldots & \quad 2 \\
\text{new names} & \quad 1 & \quad 2 & \quad 3 & \quad \square & \quad \square & \quad \square & \quad \square & \quad \square & \quad 9
\end{align*}
Renaming

*n* processes

Initial names: 23, 595, 1, ..., 2

New names: 1, 2, 3, ..., 9
One-shot Renaming

Parameters

- total $n$ number of processes
- $[1..N]$ range of initial names
- $[1..M]$ range of new names $M << N$
Parameters

- total $n$ number of processes
- $[1..N]$ range of initial names
- $[1..M]$ range of new names $M \ll N$

`getname()`

- Returns a unique name in $[1..M]$
- At most one invocation per process
• Range of allocated names is a function of \( \# \) invocations of getname()

if \( k \) procs invoke getname, each new name \( \in [1..f(k)] \)
Renaming in Shared Memory
register Operations:

- write(v)
- read()
register Operations:

- \texttt{write}(v)
- \texttt{read}()

Size:

- unbounded
register Operations:

- write(v)
- read()

Size:

- unbounded
- (large enough to store any initial name)
Atomic Register

- Each operation is instantaneous
- **Read** returns last value **written**

```
  p read() ⊥
  q write(4)
  p write(5)
  q read() 5
```
Asynchronous Process

- Each Proc. is arbitrarily slow
Asynchronous Process

- Each Proc. is arbitrarily slow

\begin{verbatim}
 p read()
 q write()
 p write()
 q read()
\end{verbatim}
Asynchronous Process

- Each Proc. is arbitrarily slow

\[
\begin{array}{c}
p \text{ read}() \\
q \text{ write}() \\
p \text{ write}() \\
q \text{ read}() \\
p \text{ read}() \\
p \text{ write}() \\
q \text{ write}() \\
q \text{ read}() \\
p \text{ write}()
\end{array}
\]
Asynchronous Process

- Each Proc. is arbitrarily slow
- Each proc. may not participate

\[
\begin{align*}
\text{p} & \quad \text{read()} \quad \text{p} & \quad \text{read()} \quad \text{p} & \quad \text{read()} \\
\text{q} & \quad \text{write()} \quad \text{p} & \quad \text{write()} \quad \text{q} & \quad \text{write()} \\
\text{p} & \quad \text{write()} \quad \text{q} & \quad \text{write()} \quad \text{q} & \quad \text{read()} \\
\text{q} & \quad \text{read()} \quad \text{q} & \quad \text{read()} \quad \text{p} & \quad \text{write()}
\end{align*}
\]
Wait-free Algorithms

$n$ processes, initial names $1..N$

getname()
    perform at most $B(n, N)$ read/write ops
return new name
wait-free Algorithms

\( n \) processes, initial names 1..\( N \)

```plaintext
getname()
    perform at most \( B(n, N) \) read/write ops
    return new name
```

- Mutual exclusion prohibited
- Provide crash-fault tolerance
Moir and Anderson Renaming Algorithm
\[ \leq 1 \quad \text{stop} \quad \leq k - 1 \quad \text{right} \]

\[ \leq k - 1 \quad \text{left} \]

\[ k = \# \text{ procs invoking the object} \]
• Each invocation returns \textit{stop, left} or \textit{right}

If only one process invokes the object:
• it obtains \textit{stop}

If $k \geq 2$ processes invoke the object:
• At \textit{least one proc.} obtains \textit{stop} or \textit{right}
• At \textit{least one proc.} obtains \textit{stop} or \textit{left}
register $C, Door$

Init $Door \leftarrow open$
register \( C, Door \)

**Init** \( Door \leftarrow \text{open} \)

split():

\[
C.\text{write}(my_id) \\
d \leftarrow Door.\text{read}()
\]
register $C$, $Door$

Init $Door \leftarrow open$

split():
    $C.write(my\_id)$
    $d \leftarrow Door.read()$
    if $d = closed$ then return right
register \( C, Door \)

**Init** \( Door \leftarrow \text{open} \)

split():

\[ C.\text{write}(my\_id) \]

\[ d \leftarrow Door.\text{read}() \]

**if** \( d = \text{closed} \) **then** return **right**

**else**

\[ Door.\text{write}(\text{closed}) \]

\[ c \leftarrow C.\text{read}() \]
register $C$, $Door$

Init $Door ← open$

split():
    $C\cdot$write($my\_id$)
    $d ← Door\cdot$read()
    if $d = closed$ then return right
    else
        $Door\cdot$write($closed$)
        $c ← C\cdot$read()
        if $c = my\_id$ then return stop
        else return left
Solo Executions

register $C, Door$

Init $Door \leftarrow open$

split():

$C.write(my\_id)$

$d \leftarrow Door.read()$

if $d = closed$ then return right

else

$Door.write(closed)$

$c \leftarrow C.read()$

if $c = my\_id$ then return stop

else return left

Solo execution:
object returns stop
register $C$, $Door$

**Init** $Door \leftarrow$ open

split():

$C$.write($my_id$)

d $\leftarrow$ $Door$.read()

if $d = closed$ then return right
else

$Door$.write($closed$)

c $\leftarrow$ $C$.read()

if $c = my_id$ then return stop
else return left

Some proc has to close the door:
at least one proc. *does not* obtain **right**
By contradiction: assume \( p \) and \( q \) obtain stop
By contradiction: assume $p$ and $q$ obtain stop
By contradiction: assume $p$ and $q$ obtain \textbf{stop}
By contradiction: assume $p$ and $q$ obtain \textbf{stop}
By contradiction: assume $p$ and $q$ obtain stop
By contradiction: assume $p$ and $q$ obtain \textbf{stop}
register $C, Door$

Init $Door \leftarrow open$

split():
   \begin{align*}
   &C.write(my_id) \\
   &d \leftarrow Door.read() \\
   &\text{if } d = \text{closed} \text{ then return right} \\
   &\text{else}\n   &\hspace{1em} Door.write(\text{closed})
   \end{align*}

(*) $c \leftarrow C.read()$
   \begin{align*}
   &\text{if } c = my_id \text{ then return stop} \\
   &\text{else return left}
   \end{align*}

If every proc gets left: last proc. that writes $C$ sees its id at (*) and return stop
A Network of Splitters

1 ➔ 2 ➔ 4 ➔ 7 ➔ 11

3 ➔ 5 ➔ 8 ➔ 12

6 ➔ 9 ➔ 13

10 ➔ 14

15
Moir-Anderson Algorithm

A network of splitters
  • Each splitter numbered from 1 to \( k(k + 1)/2 \)
  • If \( p \) gets \textbf{stop} from splitter \( i \):
    \( \rightarrow \) returns \( i \) as its new name

If \( k \) procs. invoke \texttt{getname()}:
  • new names \( \in [1..\theta(k^2)] \)
  • \( O(k) \) read/write operations per process
Every participating process gets a new name
Every participating process gets a new name
Every participating process gets a new name
Every participating process gets a new name
Every participating process gets a new name
Every participating process gets a new name
Every participating process gets a new name
Every participating process gets a new name
For $k$ participating processes

- namespace: $[1..\Theta(k^2)]$
- work: $O(k)$ per process/ $O(k^2)$ total
Randomized splitter

\[ \leq 1 \]

stop

Pr 1/2

right

Pr 1/2

left

\[ k = \# \text{ procs invoking the object} \]
Randomized Splitter Object

- Each invocation returns stop, left or right

If only one process invokes the object:
- It obtains stop

If $k \geq 2$ processes invoke the object:
- At most one process obtains stop
- Each process that does not stop
  - obtains right with proba. $1/2$
  - obtains left with proba. $1/2$. 
shared: splitter S /* non randomized */

randsplit():
  \[
  \text{dir} \leftarrow S.\text{split}()
  \]
  \[
  \text{if } \text{dir} = \text{stop} \text{ then return stop}
  \]
  \[
  x \leftarrow \text{flip\_coin()} /* \text{dir} \in \{\text{left, right}\} */
  \]
  \[
  \text{if } x = \text{head} \text{ then return right else return left}
  \]
Tree of Randomized Splitters

Attiya et. al 2006

c \log k \cdot \Pr[q \text{ same rand. choices as } p] = \frac{1}{2}

c \log k \cdot \Pr[\text{some proc does not decide}] \leq \left(\frac{k}{2}\right) \frac{1}{k^c} \leq \frac{1}{k^c - 2}
Attiya et al. 2006

Tree of Randomized Splitters

\[ c \log k \cdot \Pr[\text{same rand. choices as } p] = 1 \]

\[ \Pr[\text{some proc does not decide}] \leq \left( \frac{k}{2} \right)^{c - 2} \]
Attiya et al. 2006

Tree of Randomized Splitters

Pr[q same rand. choices as p] = \frac{1}{2} c \log k
\[ \Pr[q \text{ same rand. choices as } p] = \frac{1}{2}c \log k \]
\[ \Pr[\text{some proc does not decide}] \leq \binom{k}{2} \frac{1}{k^c} \leq \frac{1}{k^{c-2}} \]
<table>
<thead>
<tr>
<th></th>
<th>namespace</th>
<th>work per proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid of splitter</td>
<td>$O(k^2)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>tree of</td>
<td>$O(k^c)$</td>
<td>$O(c \log k)$</td>
</tr>
<tr>
<td>randomized splitters</td>
<td>$O(k^c)$</td>
<td>$O(c \log k)$</td>
</tr>
</tbody>
</table>

w.h.p.
So far

<table>
<thead>
<tr>
<th>namespace</th>
<th>work per proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid of splitter $O(k^2)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>tree of randomized splitters $O(k^c)$</td>
<td>$O(c \log k)$ w.h.p.</td>
</tr>
</tbody>
</table>

**Linear namespace?**
$= 1$

win

$= k - 1$

loose

$k = \# \text{ procs invoking the object}$
Each invocation returns **win** or **loose**

- *Exactly one proc.* obtains **win**
Each invocation returns \textbf{win} or \textbf{loose}

- \textit{Exactly one proc.} obtains \textbf{win}
- no deterministic implementation with registers
If $k$ processes participate:

(namespace work per proc. total work)

$O(k)$

$O(k^2)$
If $k$ processes participate:

<table>
<thead>
<tr>
<th>namespace</th>
<th>work per proc.</th>
<th>total work</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$O(k)$</td>
<td>$O(k^2)$</td>
</tr>
</tbody>
</table>
Sorting Networks
Sorting Networks
Sorting Networks
Sorting Networks
From Sorting to Renaming

[Alistarh et. al AACH+11]
From Sorting to Renaming

[From Alistairh et. al AACH+11]
From Sorting to Renaming

[Alistarh et. al AACH+11]
From Sorting to Renaming

[Alistarh et. al AACH+11]
From Sorting to Renaming

[Alistarh et. al AACH+11]
At most 2 processes access each Test&Set
At most 2 processes access each Test&Set

2 proc. Test&Set randomized implementation [Tromp Vitanyi] $O(1)$ expected work
• \( k \) procs participate:
  exit by the *first* \( k \) output wires
• namespace = [1..k]
\begin{itemize}
  \item $k$ procs participate:
    \begin{itemize}
      \item exit by the \textit{first} $k$ output wires
    \end{itemize}
  \item namespace $= [1..k]$
\end{itemize}
• $k$ procs participate: exit by the \textit{first} $k$ output wires

• namespace = $[1..k]$
Assigning Input Wires
Complexity

- Namespace = [1..k]
- (Expected) individual work = \( O(\text{depth of the sorting network}) \)
  - \( O(\log n) \) [AKS] (implicit)
  - \( O(\log^2 n) \) Bitonic sort [Batcher] (explicit)

Complexity can be made adaptive to \# participating procs.
Namespace

- Adaptive renaming in $[1..2k - 2]$ impossible
- Non-adaptive renaming in $[1..2n - 2]$ impossible iff $n$ is a prime power [Castañeda Rajsbaum, Attiya Paz]
### Namespace

- Adaptive renaming in \([1..2k - 2]\) impossible
- Non-adaptive renaming in \([1..2n - 2]\) impossible iff \(n\) is a prime power [Castañeda Rajsbaum, Attiya Paz]

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(k^2))</td>
<td>(O(k))</td>
<td>Moir-Anderson splitter network</td>
</tr>
<tr>
<td>(2k - 1)</td>
<td>(O(k^2))</td>
<td>Afek Merrit</td>
</tr>
<tr>
<td>(6k - 1)</td>
<td>(O(k \log k))</td>
<td>Attiya Fouren</td>
</tr>
<tr>
<td>(8k - \log k - 1)</td>
<td>(O(k))</td>
<td>Chlebus Kowalski (implicit)</td>
</tr>
</tbody>
</table>
Deterministic Wait-free R/W Algorithms

Namespace

• Adaptive renaming in $[1..2k - 2]$ impossible
• Non-adaptive renaming in $[1..2n - 2]$ impossible iff $n$ is a prime power [Castañeda Rajsbaum, Attiya Paz]

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Complexity</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(k^2)$</td>
<td>$O(k)$</td>
<td>Moir-Anderson splitter network</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(k^2)$</td>
<td>Afek Merrit</td>
</tr>
<tr>
<td>$6k - 1$</td>
<td>$O(k \log k)$</td>
<td>Attiya Fouren</td>
</tr>
<tr>
<td>$8k - \log k - 1$</td>
<td>$O(k)$</td>
<td>Chlebus Kowalski (implicit)</td>
</tr>
</tbody>
</table>

Lower bound $\Omega(k)$ for adaptive algorithm
Randomized Renaming

- Expected individual/total work
- And/or name uniqueness w.h.p
Randomized Renaming

- Expected individual/total work
- And/or name uniqueness w.h.p

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Complexity</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$O(\log k)$</td>
<td>Alistarh et. al (implicit [AKS])</td>
</tr>
<tr>
<td>$k$</td>
<td>$O(\log^2 k)$</td>
<td>Alistarh et. al (Bitonic sort)</td>
</tr>
<tr>
<td>$(1 + \epsilon)k$</td>
<td>$O(\log \log k)$</td>
<td>Alistarh et. al (relax name uniqueness)</td>
</tr>
</tbody>
</table>
Randomized Renaming

- Expected individual/total work
- And/or name uniqueness w.h.p

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Complexity</th>
<th>Authors/Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$O(\log k)$</td>
<td>Alistarh et. al (implicit [AKS])</td>
</tr>
<tr>
<td>$k$</td>
<td>$O(\log^2 k)$</td>
<td>Alistarh et. al (Bitonic sort)</td>
</tr>
<tr>
<td>$(1 + \epsilon)k$</td>
<td>$O(\log \log k)$</td>
<td>Alistarh et. al (relax name uniqueness)</td>
</tr>
</tbody>
</table>

Weak lower bound $\Omega(\log \log k)$ work for at least one processes
Message Passing

- Complete Network
- **Failures**: Crash
- Asynchronous

![Network Diagram]
- Complete Network
- **Failures**: Crash
- Asynchronous
<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Partially Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f &lt; \sqrt{n}$</td>
<td>$f &lt; n$</td>
<td>$f &lt; n/2$</td>
</tr>
<tr>
<td>(cst)</td>
<td>(\log f)</td>
<td>(exp(n))</td>
</tr>
<tr>
<td>[AAGT]</td>
<td></td>
<td>[ABDPR]</td>
</tr>
</tbody>
</table>
Renaming in Message Passing

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Partially Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f &lt; \sqrt{n}$</td>
<td>$f &lt; n$</td>
<td>$f &lt; n/2$</td>
</tr>
<tr>
<td>cst</td>
<td>log $f$</td>
<td>$\exp(n)$</td>
</tr>
<tr>
<td>[AAGT]</td>
<td></td>
<td>[ABDPR]</td>
</tr>
</tbody>
</table>
### Renaming in Message Passing

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Partially Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>$&lt; \sqrt{n}$</td>
<td>$&lt; n$</td>
<td>$&lt; n/2$</td>
</tr>
<tr>
<td>$cst$</td>
<td>$\log f$</td>
<td>$???$</td>
<td>$exp(n)$</td>
</tr>
<tr>
<td>[AAGT]</td>
<td></td>
<td></td>
<td>[ABDPR]</td>
</tr>
</tbody>
</table>

- Byzantine failures?
- Messages adversaries?
- Dynamic networks?
Equivalence

Asynchronous models
Shared memory can be simulated in message passing if \#crashs < \(n/2\)

registers can be simulated in message passing iff \#crash < \(n/2\)

[ABD]
• **Descartes** Abstraction Layers For Distributed Computing
  • Non-generalist simulations
  • Inherent cost of simulations

• **FREDDA** FoRmal mEthods for the Design of Distributed Algorithms
  • Robust algorithms: from synchronous to partially synchronous algorithms
    from benign to more severe failures
Thanks