ALGORITHMS: RECOVERABLE MUTEX AND CONSENSUS

Presenter: Wojciech Golab
wgolab@uwaterloo.ca
PODC 2019
Toronto
August 2\textsuperscript{nd}, 2019
OUTLINE

• Background

• Recoverable Mutex

• Recoverable Consensus
BACKGROUND
PROCESS VS. THREAD

Theory:
process = thread

Practice:
process = collection of parallel threads

This talk: theory
MEMORY HIERARCHY

- CPU registers
- L1/L2/L3 cache
- DRAM
- secondary storage
WHAT HAPPENS DURING A FAILURE?

Case 1: system-wide failure (reboot)

- power outage
- kernel panic

1. write X
2. write Y
3. write Z
4. crash
WHAT HAPPENS DURING A FAILURE?

Case 2: individual process failure (no reboot)

- software bug
- uncaught exception
- deadlock breakup

1. write X
2. write Y
3. write Z
4. crash
...

1. 
2. 
3. 
4. 
...
WHAT HAPPENS DURING A FAILURE?

Case 2: individual process failure (no reboot)

Question: Did we lose anything important?

Answer:
Yes, the program counter, stack pointer, and the values of certain program variables.
WHAT HAPPENS DURING A FAILURE?

Example:

```python
if T.TestAndSet() then
    // loser
else
    // winner
end if
```
WHAT HAPPENS DURING A FAILURE?

Example:

```plaintext
CPU register := T.TestAndSet()
if CPU register = 1 then
   // loser
else
   // winner
end if
```
In both failure modes, we should be concerned with the potential loss of the response to a Write or Read-Modify-Write operation on a shared variable.

Exception: multi-reader single-writer registers.
ASSUMPTIONS

1. Asynchronous shared memory.
2. Crash-recovery failures (system-wide or independent).
3. Max number of processes $N$ known ahead of time.
4. Participation by all processes is not required (e.g., possible that only $k < N$ processes take steps in an execution).
5. Read-Modify-Write primitives return responses in volatile CPU registers.
OBSERVATIONS

1. In an execution involving $N$ processes, the maximum number of failures is not bounded by $N$. It is unbounded!

2. In an execution containing infinitely many independent failures, it is possible that some processes fail only a finite number of times.
RECOVERABLE MUTEX
MUTUAL EXCLUSION PROBLEM

Asynchronous, reliable processes.
REMOTE MEMORY REFERENCES (RMR)

**Legend:**
- P: processor
- M: memory module
- C: cache

**Distributed Shared Memory (DSM):**
- Remote read or write
- Interconnect
- Read + cache miss

**Cache-Coherent (CC):**
- Write + invalidation
- Interconnect
- Read + cache miss

**Notes:**
- Remote memory references (RMR) are mechanisms allowing processors to access memory modules on different processors.
- DSM and CC are two strategies for achieving this, each with different characteristics.
Asynchronous, unreliable processes.

Golab and Ramaraju [PODC'16]
TERMINOLOGY

**Passage:**
Sequence of step taken by a process from when it begins Recover to when it completes Exit, or crashes, whichever occurs first.

**Super-passage:**
Maximal non-empty collection of consecutive passages executed by the same process where (only) the last passage in the collection is failure-free.
OBSERVATIONS

1. A process enters the CS at most once in each passage.
2. A process may enter the CS up to $f + 1$ times in a super-passage where it fails $f$ times.
3. A process must reenter the CS after it fails in Exit.
TWO WAYS TO NEST LockS

Way 1:

L1.lock()
L2.lock()
L2.unlock()
L1.unlock()

Way 2:

L1.lock()
L2.lock()
L1.unlock()
L2.unlock()
RME CORRECTNESS PROPERTIES

Mutual Exclusion (ME)
Deadlock Freedom (DF)
Starvation Freedom (SF)
Bounded Recovery (BR)
Critical Section Re-entry (CSR)

Golab and Ramaraju [PODC'16]
EXAMPLE OF REVISED PROPERTY

Starvation Freedom (SF):

For any infinite fair history $H$, if a process $p_i$ leaves the non-critical section in some step of $H$ then eventually $p_i$ itself enters the CS, or else there are infinitely many crash steps in $H$. 
Critical Section Re-entry (CSR):
If a process $p$ crashes inside the CS, then the next process to enter the CS is also $p$.

Property required for nesting locks correctly!
TEST-AND-SET LOCK

Shared variable: $T$, initially 0

Algorithm for process $p_i$:
Enter

loop while TestAndSet($T$) = 1
back off
end loop

Critical Section
Exit

$T := 0$

Properties:
Mutual Exclusion
Deadlock Freedom
Wait-free Exit
<table>
<thead>
<tr>
<th>Upper &amp; lower bounds</th>
<th>single-word Read/Write/CAS...</th>
<th>+ single-word FAS/FAA/CAS</th>
</tr>
</thead>
</table>
| **Mutual Exclusion** | **O**(log N)  
Yang, Anderson [DC'95]  
Ω**(log N)  
Attiya, Hendler, Woelfel [STOC'08] | **O**(1)  
Mellor-Crummey, Scott [TOCS'91] |
| **Recoverable Mutual Exclusion** | **O**(log N)  
Golab, Raramaju [PODC'16]  
Jayanti, Joshi [DISC'17]  
Ω**(log N)  
Attiya, Hendler, Woelfel [STOC'08] | **O**(log N / log log N)  
Golab, Hendler [PODC'17]  
Jayanti, Jayanti, Joshi [PODC'19]  
**O**(1)  
Golab, Hendler [PODC'18] |
AN IMPORTANT DIFFERENCE

\[ \mathcal{O}(\log N / \log \log N) \]
Golab, Hendler [PODC'17]
Jayanti, Jayanti, Joshi [PODC’19]

\[ \mathcal{O}(1) \]
Golab, Hendler [PODC'18]

independent failures

system-wide failures
THOUGHTS ON STARVATION FREEDOM

Golab and Ramaraju [PODC’16] allow processes to starve in executions with infinitely many failures:

**Starvation Freedom:**
For any infinite fair history $H$, if a process $p_i$ leaves the non-critical section in some step of $H$ then eventually $p_i$ itself enters the CS, or else there are infinitely many crash steps in $H$. 

27
THOUGHTS ON STARVATION FREEDOM

Golab and Hendler [PODC’18] introduced an additional correctness property that mitigates this problem:

**Failures-Robust Fairness (FRF):**
For any fair history $H$ containing infinitely many super-passages, if a process $p_i$ leaves the NCS in some step of $H$ then $p_i$ eventually itself enters the CS.
THOUGHTS ON STARVATION FREEDOM

Jayanti, Jayanti, and Joshi [PODC’19] also proposed an alternative SF property:

**Starvation Freedom:**

If every process crashes only a finite number of times in each of its super-passages in a run, then every process enters the CS in each of its super-passages in that run.
RELATIONSHIP BETWEEN PROPERTIES

Strongest

Golab and Ramaraju
Starvation Freedom
+
Golab and Hendler
Failures Robust Fairness

Weakest

Jayanti, Jayanti and Joshi
Starvation Freedom

Golab and Ramaraju
Starvation Freedom

?
RECOVERABLE CONSENSUS
PROBLEM DEFINITION

Agreement: distinct processes never output different decisions.

Validity: each decision returned is the proposal value of some process.

Recoverable wait-freedom: each time a process executes it algorithm from the beginning, it either returns a decision after a finite number of its own steps, or crashes.
## CONSENSUS HIERARCHY

<table>
<thead>
<tr>
<th>Type</th>
<th>Consensus Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare-And-Swap</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Test-And-Set</td>
<td></td>
</tr>
<tr>
<td>Fetch-And-Store</td>
<td></td>
</tr>
<tr>
<td>Fetch-And-Add</td>
<td>2</td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>Queue</td>
<td></td>
</tr>
<tr>
<td>Read/Write Register</td>
<td>1</td>
</tr>
</tbody>
</table>

Herlihy, 1991
## Recoverable Consensus Hierarchy: System-Wide Failures

<table>
<thead>
<tr>
<th>Type</th>
<th>R-Consensus Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare-And-Swap</td>
<td>∞</td>
</tr>
<tr>
<td>Test-And-Set</td>
<td></td>
</tr>
<tr>
<td>Fetch-And-Store</td>
<td></td>
</tr>
<tr>
<td>Fetch-And-Add</td>
<td>2</td>
</tr>
<tr>
<td>Queue</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>Read/Write Register</td>
<td>1</td>
</tr>
</tbody>
</table>
TRANSFORMATION

Shared variables:
- $P[1..2]$: array of proposal values, init ⊥
- $C$: conventional wait-free 2-process consensus object
- $D$: decision, init ⊥

Private variables:
- other: process ID
- $d$: decided value
OBSERVATIONS

If a process begins executing $C$ and then crashes, it cannot execute $C$ again!

If a process knows that it lost, then it also knows exactly who won.
**Procedure Decide** \((v: \text{proposal value})\) for proc. \(p_i, i \in 1..2\)

1. \(\text{if } i = 1 \text{ then } \text{other} := 2 \text{ else } \text{other} := 1\)
2. \(\text{if } P[i] = \bot \land P[\text{other}] = \bot \text{ then}\)
   \(\quad P[i] := v\)
   \(\quad d := C.\text{Decide}(v)\)
   \(\quad D := d\)
   \(\quad \text{return } d\)
3. \(\text{else if } D \neq \bot \text{ then}\)
   \(\quad \text{return } D\)
4. \(\text{else if } P[i] \neq \bot \land P[\text{other}] = \bot \text{ then}\)
   \(\quad \text{return } P[i]\)
5. \(\text{else if } P[i] = \bot \land P[\text{other}] \neq \bot \text{ then}\)
   \(\quad \text{return } P[\text{other}]\)
6. \(\text{else } // P[i] \neq \bot \land P[\text{other}] \neq \bot\)
   \(\quad \text{return } P[1]\)
## Recoverable Consensus Hierarchy:

\[ \leq F \text{ Independent Failures} \]

<table>
<thead>
<tr>
<th>Type</th>
<th>R-Consensus Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare-And-Swap</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Test-And-Set</td>
<td></td>
</tr>
<tr>
<td>Fetch-And-Store</td>
<td></td>
</tr>
<tr>
<td>Fetch-And-Add</td>
<td>2</td>
</tr>
<tr>
<td>Queue</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>Read/Write Register</td>
<td>1</td>
</tr>
</tbody>
</table>
TRANSFORMATION

Shared variables:
- $R[1..n]$:
  - array of read/write register, \textbf{init} 0
- $C[0..f]$:
  - array of conventional wait-free $n$-process consensus objects
- $D[0..f]$:
  - array of read/write register, \textbf{init} \bot

Private variables:
- $k, k'$: integers, uninitialized
- $d$: decision value, uninitialized

\[ f = \text{upper bound on total number of failures} \]
Procedure Decide ($v$: proposal value) for proc. $p_i$, $i \in 1..n$

\begin{align*}
&\textbf{for } k \text{ in } 0..f \text{ do} \\
&\quad \textbf{if } R[i] = k \text{ then} \\
&\qquad R[i] := k + 1 \\
&\qquad // \text{ check for a decision in a lower-numbered iteration} \\
&\quad \textbf{for } k' \in 0..(k-1) \text{ do} \\
&\qquad \quad \textbf{if } D[k'] \neq \perp \text{ then} \\
&\qquad \qquad v := D[k'] \\
&\qquad d := C[k].\text{Decide}(v) \\
&\qquad D[k] := d \\
&\qquad // \text{ check for a collision with a higher-numbered iteration} \\
&\quad \textbf{if } k < f \text{ then} \\
&\qquad \textbf{for } z \in 1..n, z \neq i \text{ do} \\
&\qquad \quad \textbf{if } R[z] > R[i] \text{ then} \\
&\qquad \qquad d := \perp \\
&\quad // \text{ return decision if known} \\
&\quad \textbf{if } d \neq \perp \text{ then} \\
&\quad \text{ return } d
\end{align*}
## RECOVERABLE CONSENSUS HIERARCHY: INDEPENDENT FAILURES

<table>
<thead>
<tr>
<th>Type</th>
<th>R-Consensus Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare-And-Swap</td>
<td>∞</td>
</tr>
<tr>
<td>Test-And-Set</td>
<td>1</td>
</tr>
<tr>
<td>Read/Write Register</td>
<td>1</td>
</tr>
</tbody>
</table>
IMPOSSIBILITY RESULTS

Result 1: space bound
If there are up to $f$ failures then $f + 1$ instances of TAS are necessary.

Result 2: consensus number
If there are arbitrarily many failures, then recoverable consensus is not solvable even with infinitely many TAS objects!
RESULT 1: SPACE BOUND

Valency argument, similar to Herlihy’s but modified for crash-recovery failures.

**v-potent state s:** there exists a sequence of steps starting from s such that some process returns v.

**v-valent state s:** v-potent but not v’-potent for any $v' \neq v$.

**univalent state s:** v-valent for some v.

**bivalent state:** both v-potent and v’-potent for some distinct values v and v’.
RESULT 1: SPACE BOUND

Why Herlihy’s technique breaks:

- crash
- ...
RESULT 1: SPACE BOUND

Why Herlihy’s technique breaks:

\[ v \]
\[ p_1 \text{ non-crash} \]
\[ p_1 \text{ crash} \]
\[ p_2 \text{ non-crash} \]
\[ v \]
\[ p_2 \text{ crash} \]

\[ v \text{-valent} \]

\[ \text{bivalent} \]
RESULT 1: SPACE BOUND

Consider a subset of execution histories satisfying the following invariants:

1. Only one designated process $p_i$ is permitted to fail.
2. If $p_i$ fails $f$ times then it has touched $f$ distinct TAS objects.
RESULT 1: SPACE BOUND

When is $p_i$ allowed to fail?

Only if its previous step was its first access to some TAS object in the execution.
RESULT 2: CONSENSUS NUMBER

Consider a subset of execution histories satisfying the following invariants:

1. Only one designated process $p_i$ is permitted to fail.
2. If $p_i$ fails $f$ times then the other process $p_j$ has taken at least $f$ steps failure-free.

Note: A similar proof technique was developed in parallel by Attiya, Ben-Baruch, and Hendler for NRL [PODC’18].
RESULT 2: CONSENSUS NUMBER

When is $p_i$ allowed to fail?

Only if its previous step was its first access to some TAS object in the execution, and moreover that object was also accessed by the other process $p_j$. 
TAKE-AWAYS
HOW WE GOT HERE

Talked about persistent memory, thought about crash-recover failures.
RESEARCH DIRECTIONS

1. Prove a tight RMR complexity bound for RME with independent failures and single-word primitives.
2. Devise alternative $O(1)$-RMR solutions for RME with simultaneous failures.
3. Establish a more precise relationship between the conventional consensus hierarchy and the recoverable consensus hierarchy.