Contention-Related Crash Failures

Anaïs Durand

LIP6, Sorbonne Université, Paris

April 1st, 2019
Set Agreement and Renaming in the Presence of Contention-Related Crash Failures

Joint work with:

Michel Raynal

Gadi Taubenfeld
Computational Model

- Asynchronous deterministic system
- $n$ processes $p_1, \ldots, p_n$
- Atomic read/write registers
- $0 \leq t < n$ process crashes
- Participation required
Process crashes

2 kinds of process crashes usually considered:

- Initially dead processes
- “Classical” (any-time) crashes: no constraints
- **Contention** = \# processes that accessed a shared register ≈ \# processes that started to compute

- \( \lambda \) = predefined contention threshold

- 2 possible definitions:

\[ \begin{align*}
\text{No crashes} & \quad \Rightarrow \lambda \\
\text{No crashes} & \quad \Rightarrow \text{contention}
\end{align*} \]
**Contention-Related Crash Failures**

- **Contention** = \# processes that accessed a shared register
  \[\approx\] \# processes that started to compute

- \(\lambda\) = predefined contention threshold

- 2 possible definitions:

  - \(\lambda\)-constrained crashes
  - No crashes

\[\lambda\] contention
Contention-Related vs. Any-Time Crash Failures

■ Consensus:
  ► [Fischer et al., 85]: **Impossible** with one any-time crash failure.
  ► [Taubenfeld, 18]: Algorithm that tolerates one \((n - 1)\)-constrained crash failure for \(n > 1\).

■ \(k\)-Set Agreement, \(1 \leq k < n\):
  ► [Borowsky, Gafni, 93]: **Impossible** with \(k\) any-time crash failures.
  ► [Taubenfeld, 18]: Algorithm that tolerates \(\ell + k - 2\) \((n - \ell)\)-constrained crash failures for \(\ell \geq 1\) and \(n \geq 2\ell + k - 2\).
Consider a problem $P$ that can be solved with $t$ any-time crash failures, but impossible with $t + 1$ any-time crash failures.

Given $\lambda$, can $P$ be solved with both

\[
\begin{align*}
t_1 & \text{ $\lambda$-constrained} \\
\text{and} \\
t_2 & \leq t \text{ any-time crash failures, with } t_1 + t_2 > t?
\end{align*}
\]

We consider here: $k$-set agreement (for $k \geq 2$) and renaming
$k$-Set Agreement
Definition

- One-shot object

- Operation $propose(\nu)$: propose value $\nu$ and return a decided value

- Properties:
  - **Validity**: decided value $\in$ proposed values
  - **Agreement**: $\leq k$ decided values
  - **Termination**: every correct process decides
**k-Set Agreement Algorithm: Properties**

- \( \lambda = n - k \)
- \( k \geq 2 \)
- \( k = m + f, \; m \geq 0, \; f \geq 1 \)

<table>
<thead>
<tr>
<th>total # of faults</th>
<th>( t = 2m + f - 1 = k + m - 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )-constrained crashes</td>
<td>( 2m )</td>
</tr>
<tr>
<td>any-time crashes</td>
<td>( f - 1 )</td>
</tr>
</tbody>
</table>

[Borowsky, Gafni, 93]: Impossible with \( k \) any-time crash failures.
$k$-Set Agreement: Parameters

Parameters $f$ and $m$ allow the user to tune the proportion of each type of crash failures.

- For max #any-time ($\leq k-1$), $t = k - 1$
  
- For max #\#-constrained ($\leq 2k-2$), $t = 2k - 2$

\[ t = k - 1 \]

\[ t = 2 \left\lfloor \frac{k}{2} \right\rfloor + \left\lceil \frac{k}{2} \right\rceil - 1 \]
**k-Set Agreement: Shared Registers (1/2)**

- **$DEC$:** atomic register, initially $\perp$

- **$PART[1 \ldots n]$:** snapshot object, initially $[\text{down, \ldots, down}]$
  
  - Atomic (linearizable) operations $\text{write}()$ and $\text{snapshot}()$
  
  - $\approx$ array of single-writer multi-reader atomic registers $PART[1 \ldots n]$ such that:
    
    - $p_i$ invokes $\text{write}(v) = \text{writes } v \text{ into } PART[i]$
    
    - $p_i$ invokes $\text{snapshot}() = \text{obtains the value of the array } PART[1 \ldots n]$ as if it read simultaneously and instantaneously all its entries
**k-Set Agreement: Shared Registers (2/2)**

- **MUTEX[1]**: one-shot deadlock-free \( f \)-mutex

- **MUTEX[2]**: one-shot deadlock-free \( m \)-mutex

  - **Operations**: acquire() and release() (invoked at most once)
  - **Properties**:
    - Mutual exclusion: \( \leq m \) processes simultaneously in critical section
    - Deadlock-freedom: if \( < m \) processes crashes, then \( \geq 1 \) process invoking acquire() terminates its invocation
operation $\text{propose}(\text{in}_i)$ is

1. $\text{PART} . \text{write}(\text{up})$; % signal participation
operation propose\((i)\) is

(1) \(\text{PART}.\text{write}(\text{up})\);
% signal participation

(2) repeat

(3) \(\text{part}_{i} := \text{PART}.\text{snapshot}()\);
% wait for \(n - t\)

(4) \(\text{count}_{i} := |\{x \text{ such that } \text{part}_{i}[x] = \text{up}\}|\);
% participants

(5) until \(\text{count}_{i} \geq n - t\) end repeat;

if \(\text{count}_{i} \leq \lambda\) then
% split processes into groups

(7) \(\text{group}_{i} := 2\);
% ⇝ \(\text{MUTEX}[2]\) (m-mutex)

else
% end if

(9) \(\text{group}_{i} := 1\);
% ⇝ \(\text{MUTEX}[1]\) (f-mutex)

end if

launch in

// the threads \(T_{1}\) and \(T_{2}\);
**k-Set Agreement Algorithm (1/2)**

**operation** `propose(i)` **is**

1. `PART.write(up);` \hspace{1cm} % signal participation
2. **repeat**
3. \hspace{1cm} `part_i := PART.snapshot();` \hspace{1cm} % wait for $n - t$
4. \hspace{1cm} `count_i := |\{x \text{ such that } part_i[x] = up\}|;` \hspace{1cm} % participants
5. **until** `count_i \geq n - t` **end repeat**;
6. **if** `count_i \leq \lambda` **then**
7. \hspace{1cm} `group_i := 2;` \hspace{1cm} % split processes into groups
8. **else**
9. \hspace{1cm} `group_i := 1;` \hspace{1cm} % $\rightsquigarrow MUTEX[1]$ (f-mutex)
10. **end if**
**k-Set Agreement Algorithm (1/2)**

**operation** `propose(in_i)`

1. `PART .write(up);`  
   % signal participation

2. **repeat**

3. `part_i := PART .snapshot();`  
   % wait for n – t

4. `count_i := \{x \text{ such that } part_i[x] = up\};`  
   % participants

5. **until** `count_i \geq n - t` **end repeat**

6. **if** `count_i \leq \lambda` **then**

7. `group_i := 2;`  
   % \( \rightarrow \) MUTEX[2] (m-mutex)

8. **else**

9. `group_i := 1;`  
   % \( \rightarrow \) MUTEX[1] (f-mutex)

10. **end if**

11. **launch in** // the threads \( T_1 \) and \( T_2 \);
thread $T_1$ is

12. loop forever
13. \hspace{2em} if $DEC \neq \bot$ then
14. \hspace{4em} return($DEC$);
15. \hspace{2em} end if;
16. end loop;

% wait for a decided value
thread $T_1$ is
(12) loop forever
(13) if $DEC \neq \bot$ then
(14) return($DEC$);
(15) end if;
(16) end loop;

% wait for a decided value

% decide a value if enters its CS

thread $T_2$ is
(17) if $group_i = 1 \lor m > 0$ then
(18) $MUTEX[group_i].acquire()$;
(19) if $DEC = \bot$ then
(20) $DEC := in_i$;
(21) end if
(22) $MUTEX[group_i].release()$;
(23) return($DEC$);
(24) end if;
thread $T_1$ is
(12) loop forever
(13) if $DEC \neq \bot$ then
(14) return($DEC$);
(15) end if;
(16) end loop;

thread $T_2$ is
(17) if $\text{group}_i = 1 \lor m > 0$ then
(18) $\text{MUTEX}[\text{group}_i].\text{acquire}()$;
(19) if $DEC = \bot$ then
(20) $DEC := \text{in}_i$;
(21) end if
(22) $\text{MUTEX}[\text{group}_i].\text{release}()$;
(23) return($DEC$);
(24) end if;

\[ \text{Decided value} = DEC \]
thread $T_1$ is
(12)  loop forever
(13)       if $\mathit{DEC} \neq \bot$ then
(14)         return($\mathit{DEC}$);
(15)       end if;
(16)  end loop;

thread $T_2$ is
(17)  if $\mathit{group}_i = 1 \lor m > 0$ then
(18)       $\mathit{MUTEX}[\mathit{group}_i].\mathit{acquire}()$;
(19)       if $\mathit{DEC} = \bot$ then
(20)           $\mathit{DEC} := \mathit{in}_i$;
(21)       end if
(22)       $\mathit{MUTEX}[\mathit{group}_i].\mathit{release}()$;
(23)       return($\mathit{DEC}$);
(24)  end if;

\begin{itemize}
\item[\textbf{a}] Decided value $= \mathit{DEC}$
\item[\textbf{b}] $\mathit{DEC}$ assigned to \textit{proposed} values $\mathit{in}_i$ in CS
\end{itemize}
**k-Set Agreement Algorithm: Validity & Agreement**

thread $T_1$ is
(12) loop forever
(13) if $DEC \neq \bot$ then
(14) return($DEC$);
(15) end if;
(16) end loop;

thread $T_2$ is
(17) if $\text{group}_i = 1 \lor m > 0$ then
(18) $\text{MUTEX}[^{\text{group}_i}].\text{acquire}()$;
(19) if $DEC = \bot$ then
(20) $DEC := in_i$;
(21) end if
(22) $\text{MUTEX}[^{\text{group}_i}].\text{release}()$;
(23) return($DEC$);
(24) end if;

\[ \text{Decided value} = DEC \]

\[ \text{DEC assigned to proposed values in}_i \text{ in CS} \]

\[ \text{MUTEX}[1] \leadsto \leq f \neq \text{values} \]
\[ \text{MUTEX}[2] \leadsto \leq m \neq \text{values} \]

\[ \Rightarrow \leq f + m = k \text{ decided values} \]
$k$-Set Agreement Algorithm: Termination (1/5)

(1) $\text{PART}.\text{write}(\text{up});$

(2) repeat

(3) $\text{part}_i := \text{PART}.\text{snapshot}();$

(4) $\text{count}_i := |\{x \text{ such that } \text{part}_i[x] = \text{up}\}|;$

(5) until $\text{count}_i \geq n - t$ end repeat;

\[ a \leq t \text{ crashes } + \text{ participation required} \]
\[ \leadsto \text{ eventually } \text{count}_i \geq n - t \text{ at every correct process } p_i \]
(1) \( \text{PART}.\text{write}(\text{up}); \)
(2) \text{repeat}
(3) \( \text{part}_i := \text{PART}.\text{snapshot}(); \)
(4) \( \text{count}_i := |\{x \text{ such that } \text{part}_i[x] = \text{up}\}|; \)
(5) \text{until } \text{count}_i \geq n - t \text{ end repeat} ;
(6) \text{if } \text{count}_i \leq \lambda \text{ then}
(7) \quad \text{group}_i := 2;
(8) \text{else}
(9) \quad \text{group}_i := 1;
(10) \text{end if}

\text{a} \leq t \text{ crashes } + \text{ participation required } \\
\leadsto \text{ eventually } \text{count}_i \geq n - t \text{ at every correct process } p_i

\text{b} \leq n - k \text{ processes with } \text{count}_i \leq n - k = \lambda \text{ when leaving loop (2)-(5) } \\
\leadsto \leq n - k \text{ processes in group 2}
thread \( T_1 \) is

(12) loop forever

(13) if \( DEC \neq \bot \) then

(14) return(\( DEC \));

(15) end if;

(16) end loop;

thread \( T_2 \) is

(17) if \( group_i = 1 \lor m > 0 \) then

(18) \( MUTEX[group_i].acquire() \);

(19) if \( DEC = \bot \) then

(20) \( DEC := in_i; \)

(21) end if

(22) \( MUTEX[group_i].release() \);

(23) return(\( DEC \));

(24) end if;

\( a \) \( \leq t \) crashes + participation required

\( \Rightarrow \) eventually \( count_i \geq n - t \) at every correct process \( p_i \)

\( b \) \( \leq n - k \) processes with \( count_i \leq n - k = \lambda \) when leaving loop (2)-(5)

\( \Rightarrow \leq n - k \) processes in group 2

\( c \) one process decides \( \Rightarrow \) every correct process decides
**k-Set Agreement Algorithm: Termination (2/5)**

**Diagram 1**
- **If** $m = 0$: $k = m + f = f$

![Diagram](image)

- $n - t$ correct processes
- $f - 1$ any-time crashes
- Group 1
- Group 2
\( k \)-Set Agreement Algorithm: Termination (2/5)

\[ \text{d If } m = 0: \ k = m + f = f \]

- If \( m = 0 \):
  - \( k = m + f = f \)
  - \( f \)-1 any-time crashes
  - \( n - t \) correct processes
  - \( n - t = n - (f - 1) = n - k + 1 \)

\[ n - t \]

\[ \text{Group 1} \]

\[ \text{Group 2} \]

\[ n - k \]
If $m = 0$: $k = m + f = f$

$n - t$ correct processes

$n - t = n - (f - 1) = n - k + 1$

$f$-1 any-time crashes

$n-k$

Group 1

Group 2
\[ \text{If } m = 0: \quad k = m + f = f \]

\[ n - t = n - (f - 1) = n - k + 1 \]

\[ \Rightarrow \geq 1 \text{ correct process } \& \leq f - 1 \text{ (any-time) crashes in group 1} \]

(Properties of DF \( f \)-mutex \( MUTEX[1] \)) \( \Rightarrow \) at least one process decides
If $m > 0$:
- $\left| \text{group 1} \right| \geq f$

- $2m \lambda$-constrained crashes

- $f-1$ any-time crashes

- $n - t$ correct processes

- Group 1

- Group 2
If $m > 0$:

- $|\text{group 1}| \geq f$

2m $\lambda$-constrained crashes

$f-1$ any-time crashes

$n - t$ correct processes

Group 1

Group 2
If $m > 0$:

- $|\text{group 1}| \geq f$

- $2m \lambda$-constrained crashes

- $f-1$ any-time crashes

- $n - t$ correct processes

- Group 1

- Group 2
If $m > 0$:

- $|\text{group 1}| \geq f$

\[ 2m \lambda \text{-constrained crashes} \]
\[ f-1 \text{ any-time crashes} \]
\[ n-t \text{ correct processes} \]

\[ \leadsto \geq 1 \text{ correct process } \& \leq f-1 \text{ (any-time) crashes in group 1} \]

(Properties of DF $f$-mutex MUTEX[1]) \Rightarrow at least one process decides
\textbf{k-Set Agreement Algorithm: Termination (4/5)}

\textbf{d} \textbf{If} \ m > 0:\n\begin{itemize}
\item $|\text{group 1}| < f$, \textbf{correct} $\in$ group 1
\end{itemize}

\begin{tikzpicture}
\node (A) at (0,0) {$2m$ $\lambda$-constrained crashes};
\node (B) at (0,-3) {$n - t$ correct processes};
\node (C) at (3,0) {$f$-1 any-time crashes};
\node (D) at (6,0) {$f$};
\node (E) at (6,-3) {Group 1};
\node (F) at (9,-3) {Group 2};
\end{tikzpicture}

$\Rightarrow \geq 1$ correct process $\& \leq f - 1$ (any-time) crashes in group 1
(Properties of DF $f$-mutex \textit{MUTEX}[1]) $\Rightarrow$ at least one process decides
If \( m > 0 \):

- \(|\text{group 1}| < f\), correct \( \notin \text{group 1} \)

\[
( n - k ) - ( n - t ) = t - k = (2m + f - 1) - (m + f) = m - 1
\]

\( \Rightarrow \) \( \geq 1 \) correct process \& \( \leq m - 1 \) crashes in group 2

(Properties of DF \( m \)-mutex \( MUTEX \)[2]) \( \Rightarrow \) at least one process decides
If $m > 0$:

- $|\text{group 1}| < f$, correct $\notin$ group 1

\[
(n - k) - (n - t) = t - k = (2m + f - 1) - (m + f) = m - 1
\]

$\implies$ at least one correct process & $\leq m - 1$ crashes in group 2

(Properties of DF $m$-mutex $\text{MUTEX}[2]$) $\Rightarrow$ at least one process decides
$k$-Set Agreement Algorithm: Properties

- $\lambda = n - k$
- $k \geq 2$
- $k = m + f$, $m \geq 0$, $f \geq 1$

<table>
<thead>
<tr>
<th>total # of faults</th>
<th>$t = 2m + f - 1 = k + m - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$-constrained crashes</td>
<td>$2m$</td>
</tr>
<tr>
<td>any-time crashes</td>
<td>$f - 1$</td>
</tr>
</tbody>
</table>
$k$-Set Agreement Algorithm: Generalization

- $\lambda = n - \ell$
- $k \leq \ell \leq n$
- $k \geq 2$
- $k = m + f$, $m \geq 0$, $f \geq 1$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>total # of faults</strong></td>
<td>$t = 2m + \ell - k + f - 1$</td>
</tr>
<tr>
<td><strong>$\lambda$-constrained crashes</strong></td>
<td>$2m + \ell - k$</td>
</tr>
<tr>
<td><strong>any-time crashes</strong></td>
<td>$f - 1$</td>
</tr>
</tbody>
</table>
Conclusion

- Notion of contention-related crash failures

- Allows to circumvent impossibility results

- **Better understanding** of fault tolerance:
  In the $k$-set agreement algorithm, can trade 1 “strong” any-time failure for 2 “weak” $(n - k)$-constrained failures

- **Future work:**
  - Tight bounds?
  - General algorithm for $k$-set agreement, $\forall k \geq 1$.
  - What about crashes after the contention threshold $\lambda$?
  - What about other definitions of weak crash failures?
Thank you for your attention!

Do you have any question?
Renaming
### Renaming

**Definition**

- **Initial name**: $id_i$
- **New name space**: $\{1 \ldots M\}$
- **Operation** $rename(id_i)$: return a new name
- **Properties**:
  - **Validity**: new name $\in \{1 \ldots M\}$
  - **Agreement**: no 2 same new names
  - **Termination**: invokation of $rename()$ by a correct process terminates
Renaming Algorithm: Properties

- \( M = n + f \)
- \( \lambda = n - t - 1 \)
- \( t = m + f, \ m \geq 0, \ f \geq 0 \)

<table>
<thead>
<tr>
<th>total # of faults</th>
<th>( t = m + f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )-constrained crashes</td>
<td>( m )</td>
</tr>
<tr>
<td>any-time crashes</td>
<td>( f )</td>
</tr>
</tbody>
</table>

[Herlihy, Shavit, 93]: Impossible with \( f + 1 \) any-time crash failures.
Parameters $f$ and $m$ allow the user to tune the proportion of each type of crash failures and the size of the new name space.

\[
M = n + t
\]

\[
M = n + \left\lceil \frac{t}{2} \right\rceil
\]

\[
M = n
\]
Renaming Algorithm: Shared Registers

- **PART** $[1 \ldots n]$: snapshot object, initially [down, \ldots, down]

- **RENAMEING**$_f$: $(n + f)$-renaming object that:
  - tolerates $\leq f$ any-time crash failures
  - does not require participation

  *e.g.* [Attiya, Welch, 04]
operation $rename(id_i)$ is

1. $PART\text{.write}(up)$;  % signal participation
2. repeat
3. $part_i := PART\text{.snapshot}()$;  % wait for $n - t$
4. $count_i := |\{x \text{ such that } part_i[x] = up\}|$;  % participants
5. until $count_i \geq n - t$ end repeat;
6. $newName_i := RENAMING.f.rename(id_i)$;  % get new name
7. return $(newName_i)$
operation rename(id_i) is

(1) \(PART.write(up)\); % signal participation
(2) repeat
(3) \(part_i := PART.snapshot()\); % wait for \(n - t\)
(4) \(count_i := |\{x \text{ such that } part_i[x] = up\}|\); % participants
(5) until \(count_i \geq n - t\) end repeat;
(6) \(newName_i := \text{RENAMEING}_f.rename(id_i)\); % get new name
(7) return(newName_i);
Renaming Algorithm: Proof

(1) \textit{PART}.write(up);

(2) repeat

(3) \texttt{part}_i := \textit{PART}.snapshot();

(4) \texttt{count}_i := |\{x \text{ such that } \texttt{part}_i[x] = \text{up}\}|;

(5) until \texttt{count}_i \geq n - t end repeat;

\[ a \leq t \text{ crashes + participation required} \]
\[ \leadsto \text{eventually } \texttt{count}_i \geq n - t \text{ at every correct process } p_i \]
Renaming Algorithm: Proof

(1) \( PART.write(up); \)
(2) repeat
(3) \( part_i := PART.snapshot(); \)
(4) \( count_i := |\{x \text{ such that } part_i[x] = up\}|; \)
(5) until \( count_i \geq n - t \) end repeat;

\[ \begin{align*}
\text{a} & \quad \leq t \text{ crashes } + \text{ participation required} \\
& \quad \leadsto \text{ eventually } count_i \geq n - t \text{ at every correct process } p_i \\
\text{b} & \quad n - t > \lambda \leadsto \text{ no } \lambda\text{-constrained crashes in } RENAMING_f \\
& \quad \leadsto \leq f \text{ crashes in } RENAMING_f
\end{align*} \]
Renaming Algorithm: Proof

(1) \( \text{PART}.\text{write}(\text{up}); \)

(2) repeat

(3) \( \text{part}_i := \text{PART}.\text{snapshot}(); \)

(4) \( \text{count}_i := |\{x \text{ such that } \text{part}_i[x] = \text{up}\}|; \)

(5) until \( \text{count}_i \geq n - t \) end repeat;

\[ \textbf{a} \leq t \text{ crashes } + \text{ participation required} \]
\[ \implies \text{ eventually } \text{count}_i \geq n - t \text{ at every correct process } p_i \]

\[ \textbf{b} \quad n - t > \lambda \implies \text{ no } \lambda\text{-constrained crashes in } \text{RENAMING}_f \]
\[ \implies \leq f \text{ crashes in } \text{RENAMING}_f \]

\[ \textbf{c} \quad \text{ participation not required for } \text{RENAMING}_f + \text{ properties of } \text{RENAMING}_f \]
\[ \implies \text{ validity, agreement, } \& \text{ termination } \]
Generalization to One-Shot Concurrent Objects

Transform $OB = \text{one-shot object tolerating } < X \text{ any-time crashes, participation not required}$

- $\lambda = n - t - 1$
- $t = m + f, \ m \geq 0, 0 \leq f \leq X$

<table>
<thead>
<tr>
<th>total # of faults</th>
<th>$t = m + f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$-constrained crashes</td>
<td>$m$</td>
</tr>
<tr>
<td>any-time crashes</td>
<td>$f \leq X$</td>
</tr>
</tbody>
</table>

operation $\text{op}(in_i)$ is

1. $\text{PART}.\text{write}(up)$;
2. repeat
3. $part_i := \text{PART}.\text{snapshot}()$;
4. $count_i := |\{ x \text{ such that } part_i[x] = up \}|$;
5. until $count_i \geq n - t$ end repeat;
6. $res_i := OB.\text{op}(in_i)$;
7. return($res_i$);