Contention-Related Crash Failures

Anaïs Durand*, Michel Raynal* †, and Gadi Taubenfeld‡

* IRISA, Université de Rennes, France
† Polytechnic University, Hong Kong
‡ Interdisciplinary Center, Herzliya, Israel

October 1st, 2018
Computational Model

- Asynchronous deterministic system
- $n$ processes $p_1, \ldots, p_n$
- Atomic read/write registers
- Process crashes
- Participation required
- **Contension** = \# processes that accessed a shared register

- \( \lambda \) = predefined contention threshold

- 2 kinds of crash failures:
  - \( \lambda \)-**constrained** crash failures:
    - No \( \lambda \)-constrained crashes
  - "**any-time**" crash failures:
**Contestation-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 \(n \geq 2\ell + k - 2\).
Motivation

Consider a problem $P$ that can be solved with $t$ any-time crash failures.

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

$t_1$ \textit{\lambda-constrained}

and

$t_2$ \textit{any-time}

crash failures, with $t_1 + t_2 > t$?

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

- **One-shot object**

- **Operation** `propose(v)`: propose value $v$ and return a decided value

- **Properties:**
  - **Validity**: decided value = proposed value
  - **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.
Parameters $f$ and $m$ allow the user to \textbf{tune} the proportion of each type of crash failures.

\begin{align*}
    m_f &= k - 1 \\
    t &= \max\; \#\text{any-time} (= k-1) \\
    m_f &= 2k - 2 \\
    t &= \max\; \#\lambda\text{-constrained} (= 2k-2)
\end{align*}
\textit{k-Set Agreement: Shared Registers (1/2)}

- \textbf{DEC}: atomic register, initially $\perp$

- \textbf{PART}[1\ldots n]: snapshot object, initially \([\text{down, \ldots, down}]\)

- Atomic (linearizable) operations \textit{write()} and \textit{snapshot()}

- $\approx$ array of single-writer multi-reader atomic register \textbf{PART}[1\ldots n] such that:
  - $p_i$ invokes \textit{write($v$)} = writes $v$ into \textbf{PART}[$i$]
  - $p_i$ invokes \textit{snapshot()} = obtains the value of the array \textbf{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 propose($i$) is

(1) \texttt{PART.write(up);}  
\% signal participation
operation \textit{propose}(in_i) \text{ is}

1. \texttt{PART.write(up)}; \hspace{2cm} \text{\% signal participation}

2. \texttt{repeat}
3. \hspace{1cm} \texttt{part}_i := \texttt{PART.snapshot}(); \hspace{2cm} \text{\% wait for } n - t
4. \hspace{1cm} \texttt{count}_i := |\{x \text{ such that } \texttt{part}_i[x] = up\}|; \hspace{2cm} \text{\% participants}
5. \hspace{1cm} \texttt{until count}_i \geq n - t \text{ end repeat};
operation propose($in_i$) is

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

(2) repeat

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

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

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

(6) if \( \text{count}_i \leq n - k \) then

(7) \( \text{group}_i := 2; \) \hfill \% split processes into groups

(8) else

(9) \( \text{group}_i := 1; \) \hfill \% \( \rightsquigarrow \text{MUTEX}[2] \) (m-mutex)

(10) end if

\( \rightsquigarrow \text{MUTEX}[1] \) (f-mutex)
operation \textit{propose}(\textit{in}_i) \text{ is}

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

(2) \hspace{1em} \text{repeat}

(3) \hspace{1em} \text{\textit{part}_i} := \text{\textit{PART}.snapshot}(); \hspace{1em} \% \text{ wait for } n - t

(4) \hspace{1em} \text{\textit{count}_i} := |\{x \text{ such that } \text{\textit{part}_i}[x] = \text{\textit{up}}\}|; \hspace{1em} \% \text{ participants}

(5) \hspace{1em} \text{until } \text{\textit{count}_i} \geq n - t \text{ end repeat;}

(6) \hspace{1em} \text{if } \text{\textit{count}_i} \leq n - k \text{ then} \hspace{1em} \% \text{ split processes into groups}

(7) \hspace{1em} \text{\textit{group}_i} := 2; \hspace{1em} \% \leadsto \text{MUTEX}[2] (m-mutex)

(8) \hspace{1em} \text{else} \hspace{1em} \% \leadsto \text{MUTEX}[1] (f-mutex)

(9) \hspace{1em} \text{\textit{group}_i} := 1;

(10) \hspace{1em} \text{end if}

(11) \hspace{1em} \text{launch in } // \text{ the threads } T_1 \text{ and } T_2;
\textbf{k-Set Agreement Algorithm (2/2)}

thread $T_1$ is

(1) loop forever
(2) \textbf{if} $DEC \not= \perp$ \textbf{then}
(3) \textbf{return}(DEC);
(4) \textbf{end if};
(5) \textbf{end loop};

% wait for a decided value
k-Set Agreement Algorithm (2/2)

thread $T_1$ is

(1) loop forever
(2) if $DEC \neq \bot$ then
(3) return($DEC$);
(4) end if;
(5) end loop;

% wait for a decided value

thread $T_2$ is

(6) if $group_i = 1 \lor m > 0$ then
(7) $MUTEX[group_i].acquire()$;
(8) if $DEC = \bot$ then
(9) $DEC := in_i$;
(10) end if
(11) $MUTEX[group_i].release()$;
(12) return($DEC$);
(13) end if;

% decide a value if enters its CS
thread $T_1$ is
(1) loop forever
(2) if $DEC \neq \bot$ then
(3) return$(DEC)$;
(4) end if;
(5) end loop;

thread $T_2$ is
(6) if $\text{group}_i = 1 \lor m > 0$ then
(7) $\text{MUTEX}[\text{group}_i].\text{acquire}()$;
(8) if $DEC = \bot$ then
(9) $DEC := in_i$;
(10) end if
(11) $\text{MUTEX}[\text{group}_i].\text{release}()$;
(12) return$(DEC)$;
(13) end if;

\[ \text{Decided value} = DEC \]
\textbf{k-Set Agreement Algorithm: Validity & Agreement}

Thread $T_1$ is

1. loop forever
2. if $DEC \neq \bot$ then
3.     return($DEC$);
4. end if;
5. end loop;

Thread $T_2$ is

6. if $\text{group}_i = 1 \lor m > 0$ then
7.     $\text{MUTEX}[\text{group}_i].\text{acquire}()$;
8.     if $DEC = \bot$ then
9.         $DEC := in_i$;
10.    end if
11. $\text{MUTEX}[\text{group}_i].\text{release}()$;
12. return($DEC$);
13. end if;

\begin{itemize}
  \item[a] Decided value = $DEC$
  \item[b] $DEC$ assigned to proposed values $in_i$ in CS
\end{itemize}
thread $T_1$ is
(1) loop forever
(2) if $DEC \neq \perp$ then
(3) return($DEC$);
(4) end if;
(5) end loop;

thread $T_2$ is
(6) if $\text{group}_i = 1 \lor m > 0$ then
(7) $\text{MUTEX}[\text{group}_i].\text{acquire}()$;
(8) if $DEC = \perp$ then
(9) $DEC := \text{in}_i$;
(10) end if
(11) $\text{MUTEX}[\text{group}_i].\text{release}()$;
(12) return($DEC$);
(13) end if;

\begin{itemize}
  \item[(a)] Decided value $= DEC$
  \item[(b)] $DEC$ assigned to proposed values $\text{in}_i$ in CS
  \item[(c)] $\text{MUTEX}[1] \rightsquigarrow \leq f \neq \text{values}$
  $\text{MUTEX}[2] \rightsquigarrow \leq m \neq \text{values}$
  $\Rightarrow \leq f + m = k$ decided values
\end{itemize}
**k-Set Agreement Algorithm: Termination**

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;

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

\( \leadsto \) eventually \( \text{count}_i \geq n - t \) at every correct process \( p_i \)
**(k-Set Agreement Algorithm: Termination)**

(6) if $\text{count}_i \leq n - k$ then
(7) $\text{group}_i := 2$;
(8) else
(9) $\text{group}_i := 1$;
(10) end if

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

\[b \quad \text{group}_i = 1 \Rightarrow \text{count}_i > n - k = \lambda\]
\[\leadsto \text{no } \lambda\text{-constrained crashes among participants of group 1}\]
\[\leadsto \leq f - 1 \text{ crashes in } f\text{-mutex } \textit{MUTEX}[1]\]
\( k \)-Set Agreement Algorithm: Termination

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

\[ b \quad \text{group}_i = 1 \Rightarrow \text{count}_i > n - k = \lambda \]
\[ \Rightarrow \text{ no } \lambda\text{-constrained crashes among participants of group 1} \]
\[ \Rightarrow \leq f - 1 \text{ crashes in } f\text{-mutex } \text{MUTEX}[1] \]

\[ c \quad \text{If } \geq 1 \text{ correct process } \in \text{ group 1 } \Rightarrow \geq 1 \text{ of them decides} \]
\[ \text{otherwise (some maths) } \Rightarrow \leq m - 1 \text{ crashes in } m\text{-mutex } \text{MUTEX}[2] \]
\[ \& \geq 1 \text{ correct process in group 2 } \Rightarrow \geq 1 \text{ of them decides} \]
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.

\[
\begin{align*}
m_f^0 &= \lceil t/2 \rceil \\
m_f^0 &= \lfloor t/2 \rfloor \\
m_f^0 &= 0 \\
m_f^0 &= t \\
M &= n + t \\
M &= n + \lceil t/2 \rceil \\
M &= n \\
\text{max} \ #\text{any-time} (= t) \\
\text{max} \ #\lambda\text{-constrained} (= t)
\end{align*}
\]
Renaming Algorithm: Shared Registers

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

- **RENAME\(\text{ING}_f\):** \((n + f)\)-renaming object that:
  - tolerates \(\leq f\) any-time crash failures
  - does not require participation

  *e.g.* [Attiya, Welch, 04]
Renaming Algorithm

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;

(newName$_i := RENAMING f.rename(id_i)$;          % get new name
return (newName$_i$))
Renaming Algorithm

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

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

\textcolor{red}{a} \leq t \text{ crashes} + \text{ participation required}

\leadsto \text{ eventually } \textit{count}_i \geq n - t \text{ at every correct process } \textit{p}_i
Renaming Algorithm: Proof

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

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

\( b \ n - t > \lambda \leadsto \) no \( \lambda \)-constrained crashes in \( \text{RENAME}ING_f \)
\( \leadsto \leq f \) crashes in \( \text{RENAME}ING_f \)
Renaming Algorithm: Proof

(1) \textit{PART}.\texttt{write}(\textit{up});
(2) \textbf{repeat}
(3) \hspace{1em} \textit{part}_i := \textit{PART}.\texttt{snapshot}();
(4) \hspace{1em} \textit{count}_i := |\{x \text{ such that } \textit{part}_i[x] = \textit{up}\}|;
(5) \hspace{1em} \textbf{until } \textit{count}_i \geq n - t \textbf{ end repeat} ;

\begin{itemize}
  \item[a] \hspace{1em} \leq t \text{ crashes } + \text{ participation required } \implies \text{ eventually } \textit{count}_i \geq n - t \text{ at every correct process } p_i \\
  \item[b] \hspace{1em} n - t > \lambda \implies \text{ no } \lambda\text{-constrained crashes in } \textit{RENAMEING}_f \implies \leq f \text{ crashes in } \textit{RENAMEING}_f \\
  \item[c] \hspace{1em} \text{ participation not required for } \textit{RENAMEING}_f + \text{ properties of } \textit{RENAMEING}_f \implies \text{ validity, agreement, & termination}
\end{itemize}
Conclusion

- Notion of \textit{contention-related} crash failures

- Allows to circumvent impossibility results

- Future work:
  - Tight bounds?
  - General algorithm for $k$-set agreement, $k \geq 1$. 
Thank you for your attention!

Do you have any question?
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** $op(in_i)$ is

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