Tight Bounds for Anonymous Adopt-Commit Objects

Faith Ellen
University of Toronto

joint work with Jim Aspnes
to appear at SPAA 2011
Consensus
Consensus

- **termination**: each nonfaulty process outputs a value
- **agreement**: all outputs are the same
- **validity**: every output is an input
consensus using only r/w registers:

there is no deterministic algorithm that tolerates 1 process crash in an asynchronous system [FLP, LA]

there are randomized algorithms that tolerate any number of process crashes in asynchronous systems [A, AB, AC, AH, C, CIL]

termination: each nonfaulty process outputs a value with probability 1
randomized consensus algorithms

convergence: if all inputs are v, all outputs are (commit,v)

coherence: if some output is (commit,v), every output is (commit,v) or (adopt,v)

probabilistic agreement: all outputs are the same with probability \( \Delta > 0 \)
\[ A = \text{expected step complexity of adopt-commit} \]

\[ C = \text{expected step complexity of conciliator} = O(\log n) \]

if \( \triangle \) is constant, expected step complexity of consensus is \( O(A + C) \)
m-valued adopt-commit objects

- $O(n)$ deterministic [Gafni]
- $O(\log m)$ deterministic, anonymous [Aspnes]

Anonymous: all processes run the same code

- $O(\min(n, \log m / \log \log m))$ deterministic, anonymous and matching randomized, anonymous lower bound
m-valued adopt-commit object

\text{adoptCommit}(u), u \in [1,m]

possible outputs: \{((\text{adopt},v)| v \in [1,m]) \}

U \{(\text{commit},v)| v \in [1,m]\}

termination: each nonfaulty process outputs a value

validity:: every output is an input

convergence: if all inputs are v, all outputs are (commit,v)

coherence: if some output is (commit,v), every output is (commit,v) or (adopt,v)
m-valued conflict detector

\text{check}(v), v \text{ in } [1,m]
possible outputs: \{true, false\}

termination: each nonfaulty process outputs a value
in every execution in which all \text{check} operations have the same input, they all output false

in every execution that contains \text{check}(v) \text{ and } \text{check}(v'), at least one of them outputs true
a conflict detector from an adopt-commit object

\[
\text{check}(v) \\
(d,v') := \text{adoptCommit}(v) \\
\text{if } (d,v') = (\text{commit},v) \\
\text{then return false} \\
\text{else return true}
\]
an adopt-commit object from a conflict detector and registers

\[
\text{adoptCommit}(v) \\
\text{if check}(v) \text{ then conflict := true} \\
\quad \text{else } u := \text{proposal} \\
\text{if } u = 0 \text{ then proposal := } v \\
\quad \text{else } v := u \\
b := \text{conflict} \\
\text{if } b \text{ then return (adopt,v) } \\
\quad \text{else return (commit,v)}
\]
conflict detector from registers

\[
\text{check}(v)
\]
\[
w: \text{for } i := 1 \text{ to } n \text{ do }
\]
\[
\quad \text{if done then goto } r
\]
\[
\quad M[i] := v
\]
\[
done := \text{true}
\]
\[
r: \text{for } i := 1 \text{ to } n \text{ do }
\]
\[
\quad \text{if } M[i] \neq v
\]
\[
\quad \text{then return true}
\]
\[
\text{return false}
\]

\[
\begin{array}{ccccccc}
\text{done} & \text{initially false} \\
M[1..n] & \text{all initially 0} \\
\end{array}
\]
a conflict detector from registers

\[
\text{check}(v) \\
w: \text{ for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if done then goto } r \\
\quad M[i] := v \\
\text{done} := \text{true} \\
r: \text{ for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if } M[i] \neq v \\
\quad \text{then return true} \\
\text{return false}
\]
a conflict detector from registers

\[
\text{check}(v) \\
\text{w: for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if done then goto } r \\
\quad M[i] := v \\
\text{done := true} \\
\text{r: for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if } M[i] \neq v \\
\quad \text{then return true} \\
\text{return false}
\]
a conflict detector from registers

check(v)
w: for i := 1 to n do
    if done then goto r
    M[i] := v
done := true
r: for i := 1 to n do
    if M[i] ≠ v
    then return true
return false
done
initially false
M[1..n]
all initially 0
**a conflict detector from registers**

**check(\(v\))**

\(w: \text{for } i := 1 \text{ to } n \text{ do} \)

\[
\text{if done then goto r} \\
M[i] := v
\]

\(\text{done := true} \)

\(r: \text{for } i := 1 \text{ to } n \text{ do} \)

\[
\text{if } M[i] \neq v \\
\text{then return true}
\]

return false

done

<table>
<thead>
<tr>
<th>(M)</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
<th>(2)</th>
<th>(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>done</td>
<td>(f)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
a conflict detector from registers

\[
\text{check}(v) \\
\text{w: for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if done then goto r} \\
\quad M[i] := v \\
\text{done} := \text{true} \\
\text{r: for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if } M[i] \neq v \\
\quad \text{then return true} \\
\quad \text{return false}
\]
a conflict detector from registers

check(v)
w: for i := 1 to n do
   if done then goto r
   M[i] := v
done := true
r: for i := 1 to n do
   if M[i] ≠ v
      then return true
return false

done
initially false
M[1..n]
all initially 0
a conflict detector from registers

check(v)

w: for i := 1 to n do
    if done then goto r
    M[i] := v

done := true

r: for i := 1 to n do
    if M[i] ≠ v
      then return true

return false

M

M[1..n]
all initially 0

done
initially false

2 2 2 2 2 2 0

done
a conflict detector from registers

\texttt{check}(v)
\texttt{w: for } i := 1 \text{ to } n \text{ do }
    \text{if done then goto r}
    M[i] := v
\text{done} := \text{true}
\texttt{r: for } i := 1 \text{ to } n \text{ do }
    \text{if } M[i] \neq v
    \text{then return true}
\text{return false}

done
initially false
M[1..n]
all initially 0

\begin{array}{cccccc}
3 & 2 & 2 & 2 & 2 & 0 \\
\end{array}

\begin{array}{c}
done
\end{array}
a conflict detector from registers

check(v)

w: for i := 1 to n do
if done then goto r
M[i] := v

done := true

r: for i := 1 to n do
if M[i] ≠ v
then return true

return false

done initially false
M[1..n] all initially 0
a conflict detector from registers

\[
\text{check}(v) \\
\text{w: for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if done then goto r} \\
\quad M[i] := v \\
\text{done} := \text{true} \\
\text{r: for } i := 1 \text{ to } n \text{ do} \\
\quad \text{if } M[i] \neq v \\
\quad \text{then return true} \\
\text{return false}
\]
a conflict detector from registers

\[ \text{check}(v) \]

\[ w: \text{for } i := 1 \text{ to } n \text{ do } \]

\[ \text{if done then goto } r \]

\[ M[i] := v \]

\[ \text{done} := \text{true} \]

\[ r: \text{for } i := 1 \text{ to } n \text{ do } \]

\[ \text{if } M[i] \neq v \]

\[ \text{then return true} \]

return false

done

initially false

\[ M[1..n] \]

all initially 0

3 3 2 2 2 2

done
a conflict detector from registers

check(v)

w: for i := 1 to n do
  if done then goto r
  M[i] := v

done := true

r: for i := 1 to n do
  if M[i] ≠ v
    then return true

return false

done initially false
M[1..n] all initially 0

\[
\begin{bmatrix}
3 & 3 & 3 & 3 & 2 & 2 & 2 \\
\end{bmatrix}
\]

done
a 2-valued conflict detector from registers

\[
\text{check}(v) \\
M[v] := v \\
\text{if } v = 1 \\
\text{then } x := M[2] \\
\text{else } x := M[1] \\
\text{if } x \neq 0 \\
\text{then return true} \\
\text{else return false}
\]
a 2-valued conflict detector from registers

check(v)

M[v] := v

if v = 1
then x := M[2]
else x := M[1]

if x ≠ 0
then return true
else return false

M[1..2]
both initially 0

M

1 0
a 2-valued conflict detector from registers

check(v)
M[v] := v
if v = 1
then x := M[2]
else x := M[1]
if x ≠ 0
then return true
else return false

M[1..2]
both initially 0
an m-valued conflict detector from registers

check(v)
for i := 1 to k do
    x := M[π_\{i\}]
    if x = 0
        then M[π_\{i\}] := v
    else if x ≠ v
        then return true
    return false

Π_{i=1}^{m} k is O(\log m / \log \log m)
an \( m \)-valued conflict detector from registers

check(\( v \))

for \( i := 1 \) to \( 3 \) do

\( x := M[\prod_v[i]] \)

if \( x = 0 \)

then \( M[\prod_v[i]] := v \)

else if \( x \neq v \)

then return true

return false

\[ \prod_1 = [1 \, 2 \, 3] \]
\[ \prod_2 = [1 \, 3 \, 2] \]
\[ \prod_3 = [2 \, 1 \, 3] \]
\[ \prod_4 = [2 \, 3 \, 1] \]
\[ \prod_5 = [3 \, 1 \, 2] \]
\[ \prod_6 = [3 \, 2 \, 1] \]

\( M \)
an $m$-valued conflict detector from registers

\[
\text{check}(v) \\
\text{for } i := 1 \text{ to } 3 \text{ do} \\
\quad x := M[\Pi_v[i]] \\
\quad \text{if } x = 0 \\
\quad \quad \text{then } M[\Pi_v[i]] := v \\
\quad \text{else if } x \neq v \\
\quad \quad \text{then return true} \\
\text{return false}
\]
an $m$-valued conflict detector from registers

\[ \text{check}(v) \]

for $i := 1$ to $3$ do

\[ x := M[\Pi_v[i]] \]

if $x = 0$

then $M[\Pi_v[i]] := v$

else if $x \neq v$

then return true

return false

$\Pi_1 = [1 \ 2 \ 3]$

$\Pi_2 = [1 \ 3 \ 2]$

$\Pi_3 = [2 \ 1 \ 3]$

$\Pi_4 = [2 \ 3 \ 1]$

$\Pi_5 = [3 \ 1 \ 2]$

$\Pi_6 = [3 \ 2 \ 1]$

\[ M = \begin{array}{ccc} 1 & 1 & 0 \end{array} \]
an $m$-valued conflict detector from registers

check($v$)
for $i := 1$ to $3$ do
  $x := M[\pi_1[i]]$
  if $x = 0$
    then $M[\pi_1[i]] := v$
  else if $x \neq v$
    then return true
return false

$\pi_1 = [1 \ 2 \ 3]$  
$\pi_2 = [1 \ 3 \ 2]$  
$\pi_3 = [2 \ 1 \ 3]$  
$\pi_4 = [2 \ 3 \ 1]$  
$\pi_5 = [3 \ 1 \ 2]$  
$\pi_6 = [3 \ 2 \ 1]$
an m-valued conflict detector from registers

check(v)
for i := 1 to 3 do
    x := $M[p_i^v[i]]$
    if x = 0
        then $M[p_i^v[i]] := v$
    else if x ≠ v
        then return true
    return false
an m-valued conflict detector from registers

check(v)
for i := 1 to 3 do
    x := M[Πv[i]]
    if x = 0
        then M[Πv[i]] := v
    else if x ≠ v
        then return true
return false

Π1 = [1 2 3]
Π2 = [1 3 2]
Π3 = [2 1 3]
Π4 = [2 3 1]
Π5 = [3 1 2]
Π6 = [3 2 1]
an m-valued conflict detector from registers

check(v)
for i := 1 to 3 do
  x := M[π_v[i]]
  if x = 0
    then M[π_v[i]] := v
  else if x ≠ v
    then return true
return false

π_1 = [1 2 3]
π_2 = [1 3 2]
π_3 = [2 1 3]
π_4 = [2 3 1]
π_5 = [3 1 2]
π_6 = [3 2 1]

M
\[ \begin{array}{c|ccc}
    & 0 & 4 & 4 \\
\end{array} \]
an m-valued conflict detector from registers

\[
\text{check}(v) \\
\text{for } i := 1 \text{ to } 3 \text{ do} \\
\quad x := M[\pi_v[i]] \\
\quad \text{if } x = 0 \text{ then } M[\pi_v[i]] := v \\
\quad \text{else if } x \neq v \text{ then return true} \\
\text{return false}
\]

\[
\pi_1 = [1 \ 2 \ 3] \\
\pi_2 = [1 \ 3 \ 2] \\
\pi_3 = [2 \ 1 \ 3] \\
\pi_4 = [2 \ 3 \ 1] \\
\pi_5 = [3 \ 1 \ 2] \\
\pi_6 = [3 \ 2 \ 1]
\]

\[
M = \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix}
\]
an \( m \)-valued conflict detector from registers

\[
\text{check}(v) \\
\text{for } i := 1 \text{ to } 3 \text{ do} \\
\quad x := M[\pi_v[i]] \\
\quad \text{if } x = 0 \\
\quad \quad \text{then } M[\pi_v[i]] := v \\
\quad \text{else if } x \neq v \\
\quad \quad \text{then return true} \\
\text{return false}
\]
an m-valued conflict detector from registers

check(v)
for i := 1 to 3 do
    x := M[Π_v[i]]
    if x = 0
        then M[Π_v[i]] := v
    else if x ≠ v
        then return true
return false

\[
\begin{align*}
\Pi_1 &= [1 \ 2 \ 3] \\
\Pi_2 &= [1 \ 3 \ 2] \\
\Pi_3 &= [2 \ 1 \ 3] \\
\Pi_4 &= [2 \ 3 \ 1] \\
\Pi_5 &= [3 \ 1 \ 2] \\
\Pi_6 &= [3 \ 2 \ 1] \\
\end{align*}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
\end{bmatrix}
\]
an $m$-valued conflict detector

from registers

\[\begin{align*}
\Pi_1 &= [1 \ 2 \ 3] \\
\Pi_2 &= [1 \ 3 \ 2] \\
\Pi_3 &= [2 \ 1 \ 3] \\
\Pi_4 &= [2 \ 3 \ 1] \\
\Pi_5 &= [3 \ 1 \ 2] \\
\Pi_6 &= [3 \ 2 \ 1]
\end{align*}\]

\[M = \begin{bmatrix} 1 & 4 & 0 \end{bmatrix}\]

check($v$)

for $i := 1$ to $3$ do

\[
x := M[\Pi_v[i]]
\]

if $x = 0$

then $M[\Pi_v[i]] := v$

else if $x \neq v$

then return true

return false

for any $u \neq v$, $i$ is before $j$ in $\Pi_u$ and $j$ is before $i$ in $\Pi_v$, for some $i \neq j$
Ω(\min(n, \log m / \log \log m)) lower bound on step complexity of anonymous m-valued conflict detectors for n processes
\( E(v) = \) solo execution of \( \text{check}(v) \)
\( W(v) = \) registers written to in \( E(v) \)
\( R(v) = \) registers read from, but not written to, in \( E(v) \)

\( R3, R2, W5, W3, R1, W5, R2, W6, R3 \)
\[ E(v) = \text{solo execution of} \; \text{check}(v) \]

\[ W(v) = \text{registers written to in} \; E(v) \]

\[ R(v) = \text{registers read from, but not written to, in} \; E(v) \]

\[ R3, \; R2, \; W5, \; W3, \; R1, \; W5, \; R2, \; W6, \; R3 \]

\[ \pi(v) = \text{permutation of} \; W(v) \cup R(v) \]

\[ \text{arranged according to first writes to registers in} \; W(v) \; \text{and last reads from registers in} \; R(v) \]
$$E(v) = \text{solo execution of check}(v)$$

$$W(v) = \text{registers written to in } E(v)$$

$$R(v) = \text{registers read from, but not written to, in } E(v)$$

$$\{R_3, R_2, W_5, W_3, R_1, W_5, R_2, W_6, R_3\}$$

$$\pi(v) = \text{permutation of } W(v) \cup R(v)$$

arranged according to first writes to registers in $$W(v)$$ and last reads from registers in $$R(v)$$

$$[5,3,1,2,6]$$
\[ E(v) = \text{solo execution of } \text{check}(v) \]
\[ W(v) = \text{registers written to in } E(v) \]
\[ R(v) = \text{registers read from, but not written to, in } E(v) \]

**LEMMA 1** If \( |E(v)| + |E(u)| \leq n \) then there exist \( i,j \) in \((W(v) \cup R(v)) \cap (W(u) \cup R(u))\) that occur in different orders in \( \pi(v) \) and \( \pi(u) \).
Proof: Suppose all $i,j$ in $(W(v) \cup R(v)) \cap (W(u) \cup R(u))$ occur in the same orders in $\pi(v)$ and $\pi(u)$.

$E(v) = R_3, R_2, W_5, W_3, R_1, W_5, R_2, W_6, R_3$

$\pi(v) = [5,3,1,2,6]$

$E(u) = R_5, R_1, W_5, R_3, R_4, R_1, W_7, W_2, R_5, W_2$

$\pi(u) = [5,3,4,1,7,2]$

The adversary can construct an execution $E'$ that is indistinguishable from $E(v)$ to $p$ and indistinguishable from $E(u)$ to $q$. 
Proof: Suppose all \( i,j \) in \((W(v) \cup R(v)) \cap (W(u) \cup R(u))\) occur in the same orders in \( \pi(v) \) and \( \pi(u) \).

\[
E(v) = R_3, R_2, W_5, W_3, R_1, W_5, R_2, W_6, R_3
\]

\[
\pi(v) = [5,3,1,2,6]
\]

\[
E(u) = R_5, R_1, W_5, R_3, R_4, R_1, W_7, W_2, R_5, W_2
\]

\[
\pi(u) = [5,3,4,1,7,2]
\]

The adversary can construct an execution \( E' \) that is indistinguishable from \( E(v) \) to \( p \) and indistinguishable from \( E(u) \) to \( q \).
\[ E(v) = R_3, R_2, W_5, W_3, R_1, W_5, R_2, W_6, R_3 \]

\[ E(u) = R_5, R_1, W_5, R_3, R_4, R_1, W_7, W_2, R_5, W_2 \]

\[ W_5, R_3, W_3, R_1, R_1, R_2 \]

\[ R_i \text{ is scheduled immediately before corresponding } R_i/W_i \]
\[ E(v) = R_3, R_2, W_5, W_3, R_1, W_5, R_2, W_6, R_3 \]

\[ E(u) = R_5, R_1, W_5, R_3, R_4, R_1, W_7, W_2, R_5, W_2 \]

*Ri* is scheduled immediately before corresponding *Ri/Wi*

*Wi* is scheduled immediately after corresponding *Ri/Wi*
\[ E(v) = R3, R2, W5, W3, R1, W5, R2, W6, R3 \]
\[ E(u) = R5, R1, W5, R3, R4, R1, W7, W2, R5, W2 \]

\[ R_i \text{ is scheduled immediately before corresponding } R_i/W_i \]

\[ W_i \text{ is scheduled immediately after corresponding } R_i/W_i \]

\[ R/W's \text{ between successive } R/W's \text{ and } R'/W's \text{ between successive } R/W's \text{ are interleaved arbitrarily} \]
R3,R2,R5,R1,W5,W5,R3,W3,R4,R1,R1,W7,W5,R2,W2, W6,R5,R3,W2

Problem: q may read a value written by p or p may read a value written by q

Solution: add clones.

A clone of q is a process with the same input (and code) as q, which is run in lockstep with q, until immediately before some write. The clone performs that write later to ensure that q reads the value it last wrote to that register.
For each $i$ in $W(v) \cap W(u)$:
add one clone of $q$ for each $R_i$ by $q$
after its first $W_i$ and
add one clone of $p$ for each $R_i$ by $p$
after its first $W_i$

This ensures that any read of $M[i]$ after the first two writes of $M[i]
will see the same value in $E'$ it saw in $E(v)$ or $E(u)$.
For each $i$ in $R(v) \cap W(u)$:
all $R_i$'s, $R_i$ by $p$ occur before the first write $W_i$ by $q$ and, hence read 0.

For each $i$ in $W(v) \cap R(u)$:
all $R_i$'s, $R_i$ by $q$ occur before the first write $R_i$ by $p$ and, hence read 0.
LEMMA 2 Let $\pi(1),...,\pi(m)$ be finite sequences without repetition such that, for every two sequences, $\pi(v)$ and $\pi(u)$, there exist elements $i$ and $j$ that occur in $\pi(v)$ and $\pi(u)$ in different orders. Then

$$\sum \left\{ \frac{1}{|\pi(v)|!} : v = 1,...,m \right\} \leq 1.$$
THEOREM The worst case step complexity of any deterministic anonymous m-valued conflict detector for n processes is $\Omega(\min(n, \log m / \log \log m))$. 
Proof: Let $t = \max \{|E(v)| : v = 1, \ldots, m\}$. Then $|\pi(v)| \leq |E(v)| \leq t$.

If $t > n/2$, the claim is true. Otherwise, for all $v \neq u$, $|E(v)| + |E(u)| \leq n$.

By Lemma 1, for all $u$ and $v$, there exist elements $i$ and $j$ that occur in $\pi(v)$ and $\pi(u)$ in different orders. Hence, $m/t! = \sum \{1/t! : v = 1, \ldots, m\} \leq \sum \{1/|\pi(v)|! : v = 1, \ldots, m\} \leq 1$, by Lemma 2. So $m \leq t!$ and $t$ is $\Omega(\log m / \log \log m)$. 
COROLLARY Any anonymous randomized m-valued conflict detectors for n processes has $\Omega(\min(n, \log m / \log \log m))$ step complexity with probability 1 against an oblivious adversary.
Suppose not. For each $v = 1, \ldots, m$, there is a sequence of coin flips such that some solo execution $E(v)$ by a process with input $v$ takes at most $t$ steps, where $t \leq n/2$ and $t! \leq m$. The proof of the theorem constructs an execution $E'$ in which two processes with different inputs both perform check and return false. This violates correctness.
THEOREM? Any anonymous randomized m-valued conflict detectors for n processes has $\Omega(\min(n, \log m \div \log \log m))$ step complexity with probability 1 against an oblivious adversary.