UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Lecture 12: Randomized Asynchronous Agreement

CS 539 / ECE 526

Distributed Algorithms

Announcements

- PS2 graded and solution sketch uploaded. Regrade requests due tonight.
- P3 due this Friday
- Review lecture next Monday
- Midterm in lecture next Wednesday
  - In class, on Canvas, open book

What can we do about FLP?

- Consider easier problems
- Randomization
- Consider easier models (partial synchrony)

- Agreement, total order bcast, and replication possible in psync or async with randomization
  - Single-value broadcast still impossible

Ben-Or Protocol

- The first randomized async agreement (1983)
  - Tolerate $f < n/2$ crash faults
    - Best possible in psync / async
  - Binary inputs
    - Asynchronous Binary Agreement (ABA)

- Supposed to be simple (still is relatively), but turns out to be much trickier

Ben-Or ABA Protocol

- Party j has input x_j. Will keep updating x_j

  Iteration r:

  – Round 1: party j sends (r, vote1, x_j) to all
    - Wait for n-f such msgs

  – Round 2: if all n-f vote1 are for the same x, party j sends (r, vote2, x); else, sends (r, vote2, $\perp$)
    - Wait for n-f such msgs

  – If all n-f vote2 are for the same x, then decide x; Else if there is one vote2 for x, then set x_j = x; Else, set x_j to 0 or 1 randomly. Go to iteration r+1.

Ben-Or ABA Protocol

- Party j has input $x_{j.}$ Will keep updating x_j

  Iteration r:

  – $(y, g) \leftarrow GA_r(x_j)$ // 2-round GA

  – If $g == 1$, then decide y;

  Else if $y \neq \perp$, $x_j \leftarrow y$;

  Else, $x_j \leftarrow$ a random bit

  – Go to iteration r+1

Recall Graded Agreement (GA)

- Party j has input x_j:
  - Round 1: party j sends (vote1, x_j)
    - Wait for n-f vote1 msgs
  - Round 2: if all n-f vote1 are for the same x, party j sends (vote2, x); else, sends (vote2, $\perp$)
    - Wait for n-f vote2 msgs
  - If all n-f vote2 are for the same x, then output (x, 1); Else if there is one vote2 for x, then output (x, 0); Else, output ($\perp$, 0).

Recall Graded Agreement (GA)

- n parties, each with an input, up to f faulty
- Each party outputs value y and "grade" bit g
  - g is roughly "confidence"
- Liveness: everyone outputs
- Validity: same inputs x → all output (x, 1)
- Two safety guarantees:
  - S2: One outputs (y, 1), all output (y, *)
  - S3: No (y, *) and (y', *) for $y \neq y'$, $y \neq \perp$, $y' \neq \perp$

Termination Gadget

- As written, the protocol does not have a termination rule

- Termination gadget for crash faults:
  - Upon deciding x, send “decide x” to all and terminate
  - Upon receiving “decide x”, send “decide x” to all and terminate

Ben-Or ABA Correctness

- Validity: same inputs x → (x, 1) from GA
  - In fact, if all parties start iteration r with $x_j = x$, then all decide x in iteration r
- Safety:

  One party decides y (for whom GA_r outputs (y, 1))

  → GA_r outputs (y, *) for all parties (GA S2)

  → All set x_j to y → All decide y in iteration r+1

Ben-Or ABA Correctness

- Liveness:
  - GA S3: No (y, *) and (y', *) for $y \neq y'$, $y \neq \perp$, $y' \neq \perp$
  - Possible outcomes: all \$ (coin), 0 and \$, or 1 and \$
  - If everyone happens to get the same coin flip AND it happens to equal those who adopt GA output …
    - … this happens with exponentially small prob
  - then all parties will start with the same value and will decide in next iteration

Ben-Or ABA Efficiency

- Efficient asynchronous randomized protocols use a "common coin" subroutine

Iteration r:

– $(y, g) \leftarrow GA_r(x_j)$ // 2-round GA

– If $g == 1$, then decide y;

Else if $y \neq \perp$, $x_j \leftarrow y$;

Else, $x_j \leftarrow C_r$ // C_r: r-th common coin

– Go to iteration r+1

Ben-Or ABA Efficiency

- With a common coin, want to argue: with probability $\geq \frac{1}{2}$, coin = non-$\perp$ GA output
  - At most one non-$\perp$ GA output
- Hence, decide in expected 2 iterations !?
- Turns out it's not so simple

Ben-Or ABA Liveness

- Let us take a closer look at the protocol

  Iteration r:

  – $(y_r, g_r) \leftarrow GA_r(x_j)$ // 2-round GA

  – If $g_r == 1$, then decide y_r;

  Else $x_j \leftarrow y_r$ $(\neq\perp)$ or C_r // C_r: r-th common coin

  – Go to iteration r+1

- Claim: $\Pr[C_r = y_r] = 1/2$
  – Implicit assumption: y_r is independent of C_r
  – The adversarial network can manipulate message delivery order to make sure $y_r \neq C_r$

Ben-Or ABA Protocol (n=3, f=1)

- Party j has input $x_{j.}$ Will keep updating x_j

  Iteration r:

  – Round 1: party j sends (r, vote1, x_j)
    - Wait for n-f = 2 such msgs

  – Round 2: if all n-f=2 vote1 are for the same x, party j sends (r, vote2, x); else, sends (r, vote2, ?)
    - Wait for n-f = 2 such msgs

  – If all n-f=2 vote2 are for the same x, then decide x; Else if there is one vote2 for x, then set x_j = x; Else, set x_j to 0 or 1 randomly. Go to iteration r+1.

Ben-Or ABA Liveness

- Ben-Or ABA not live if using common coin

– Initially:	0	1	v
– GA_1:	$\perp$	$1-C_1$	$1-C_1$ if $C_1 = C_2$ $\perp$ if $C_1 \neq C_2$
– Adopt:	C_1	$1-C_1$	$1-C_2$
– GA_2:	$\perp$	$1-C_2$	
– Adopt:	C_2	$1-C_2$	

- With prob ½, an adversarial network can create an infinite run
  - Need $1-C_1 = v$

Ben-Or ABA Liveness

- Ben-Or ABA not live if using common coin
- Miraculously, it is live with local coins, but very inefficient and requires a complex proof
- Can we make Ben-Or work for common coin?
  - Yes! [Abraham-BenDavid-Yandamuri, 2022]
  - Idea: prevent manipulation of GA outputs after any party outputs from GA

Need Additional Property in GA

- Liveness: everyone outputs
- Validity: same inputs x → all output (x,1)
- Two safety:
  - S2: One outputs (y, 1), all output (y, *)
  - S3: No (y, *) and (y', *) for $y \neq y'$, $y \neq \perp$, $y' \neq \perp$
- Binding: once a party outputs, all parties can only output ($\perp$, *) or (y, *) (for some $y \in \{0,1\}$)
  - "$\perp$/y-valent" as opposed to "tri-valent"

GA with Binding

- Party j has input x_j:
  - Round 1: party j sends (vote1, x_j)
    - Wait for n-f vote1 msgs
  - Round 2: if all n-f vote1 are for the same x, party j sends (vote2, x); else, sends (vote2, ⊥)
    - Wait for n-f vote2 msgs
  - Round 3: if all n-f vote2 are for the same x, party j sends (vote3, x); else, sends (vote3, ⊥)
    - Wait for n-f vote3 msgs
  - If all n-f vote3 are for the same x, then output (x, 1); Else if there is one vote3 for x, then output (x, 0); Else, output (⊥, 0).

GA with Binding

- Liveness, validity proofs similar as before
- Safety: quorum intersection → at most one non-⊥ vote2 and vote3 → both S2 and S3
- Binding: once a party outputs, all parties can only output (⊥, *) or (y, *) (for some y ∈ {0,1})
  - Consider the first time some party sends vote3
    - This is before any party outputs
  - Consider the n-f vote2 received by this party
  - If one has v ≠ ⊥ → no other non-⊥ vote2 → binding
  - If n-f (vote2, ⊥) → everyone receives a (vote2, ⊥) → everyone sends (vote3, ⊥) → binding (only ⊥)

Modern Ben-Or ABA Correctness

- Liveness:
  - Possible outcomes: all \$ (coin), 0 and \$, or 1 and \$
    - By the time coin is revealed, the outcome for this iteration is already fixed (GA binding)
  - If all coins == adopted value (if any), then all parties start with same value and decide in next iteration
    - 2^{-n} prob with local coin, ½ prob with common coin

Modern Ben-Or ABA Efficiency

- In expectation, $O(2^n)$ iterations with local coins and $O(1)$ iterations with common coin

- Rounds: $O(1)$ times expected # of iteration
- Communication complexity: $O(n^2)$ times expected # of iteration

Summary

- Ben-Or protocol: randomized asynchronous binary agreement tolerating $f < n/2$ crash
- Randomization circumvents FLP
- Also circumvents $f+1$ round lower bound