CPSC 465/565 Theory of Distributed Systems

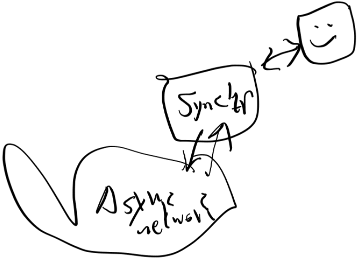
James Aspnes

2023-09-13

Today's exciting topics

- Synchronizers
- The session problem

Synchronizers (Awerbuch 1985)



Goal: Simulate synchronous message-passing protocol in an asynchronous system.

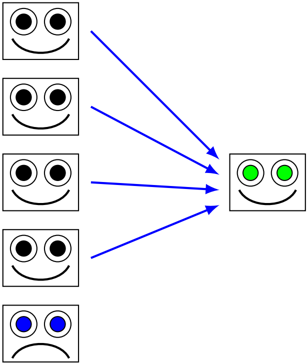
Necessary assumption: No failures!

Synchronizer interface

Synchronizer acts as bridge between simulated synchronous protocol and asynchronous network:

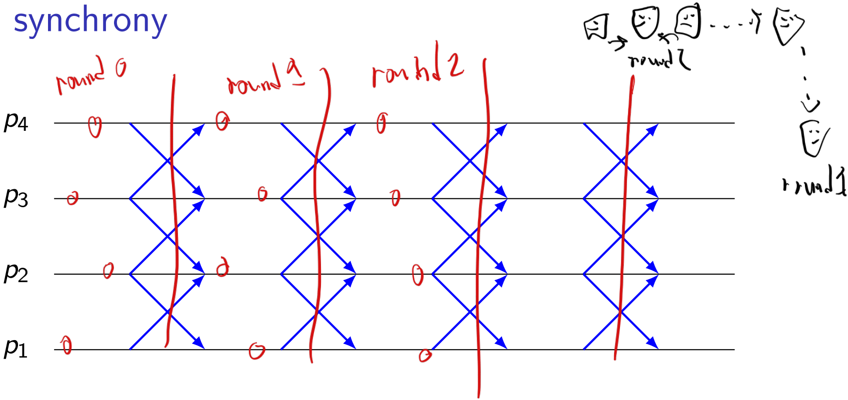
1. Synchronizer collects incoming messages for round r.
2. Synchronizer simulates synchronous delivery event del(i, S).
3. Synchronizer distributes resulting messages to neighbors for round $r + 1$.

Local synchrony



- My simulated delivery event for round $r + 1$ includes all messages sent to me in round r.
- Execution is indistinguishable from synchronous to simulated processes.

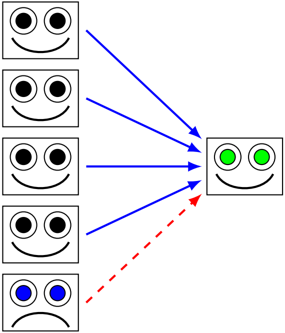
Global synchrony



- Nobody sends in $r + 1$ until everybody receives messages sent in r.
- Execution is indistinguishable from synchronous to outside observer.

Global synchrony implies local synchrony, so global is harder.

The alpha synchronizer (Awerbuch 1985)

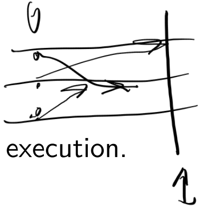


- How do I know when I have all my incoming messages?
- Replace missing message with noMsg(r).
- In each round, wait for real message or noMsg from all neighbors.

This gives a *local* synchronizer.

Proof: Immediate from waiting rule.

Time complexity of alpha synchronizer



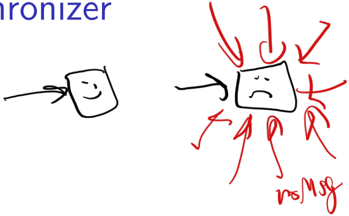
Complexity depends on length T of synchronous execution.

Time complexity is exactly T. Proof:

- Claim: All messages send in round r (including noMsg) are delivered by time $r + 1$.
- Proof (by induction on r):
  - Base case: Each round-0 message (or noMsg) sent by 0
    - $\Rightarrow$ Arrives by 1.
  - Induction step:
    - Round $r - 1$ messages (or noMsg) delivered by r (ind hyp).
    - I get all incoming messages for round r by r.
    - I send outgoing messages (or noMsg) by r.
    - My neighbors receive these messages by $r + 1$.

This is the best we can hope for.

Message complexity of alpha synchronizer



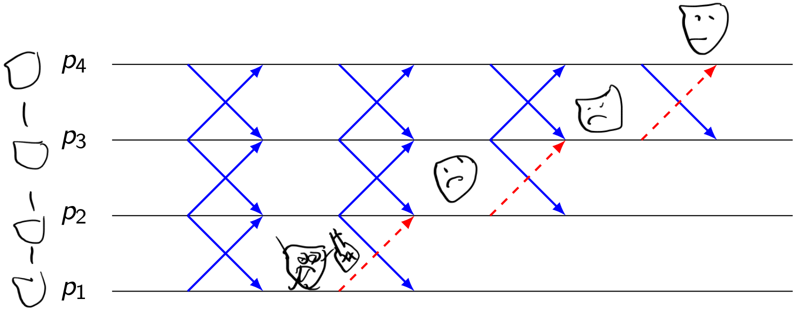
Message complexity may be bad!

- Every process sends a real message or noMsg every round.
- Length T synchronous execution $\Rightarrow 2T|E|$ messages. ← 2

Example: id-based synchronous leader election protocol.

- Lowest-id process passes n messages around the ring.
- $T = (\min \mathrm{id} + 1) \cdot n$.
- So message complexity $= (\min \mathrm{id}) \cdot n^2 =$ arbitrarily huge.

Deviation from global synchrony



- Stuck process might not stop distant processes immediately.
- $\Rightarrow$ No global synchrony in worst case.
- Can show by induction that deviation $\leq$ distance.

The beta synchronzier (also Awerbuch 1985)

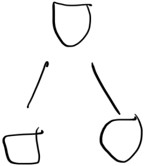
Assumes rooted spanning tree with leader at root.

1. Send/ack phase
   - Each process sends its round r messages.
   - Receiver responds with ack.
   - Sender waits for ack for each message sent.
2. Convergecast phase
   - Wait for done from all children.
   - Send done to parent.
3. Broadcast phase
   - Root waits for all expected ack/done.
   - Root broadcasts Go through tree.

Global synchrony:

- All messages delivered in send/ack phase
- Send/ack phases separated by broadcast/convergecast.

Complexity of the beta syncrhonizer



Messages:

1. $2M$ for send/ack
2. $+(n-1)$ per round for convergecast
3. $+(n-1)$ per round for broadcast

$= 2M + 2(n-1)T = O(M + T)$. (Maybe better than alpha.)

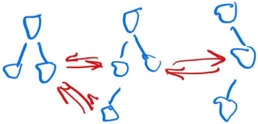
2

Time:

1. 2 per round for send/ack
2. $+ \leq D$ for convergecast
3. $+ \leq D$ for broadcast

$\leq T \cdot (D + 2)$. (Much worse than alpha.)

Hybrid: the gamma synchronizer (still Awerbuch 1985)



- Divide network into clusters.
- Run beta within each cluster, alpha between clusters.
- A node is done (for beta convergecast) only if
  - It gets ack for each in-cluster message it sent.
  - It gets message or noMsg from each out-of-cluster neighbor.

This does *not* give global synchrony.

But it may give better trade-off between messages and time if the graph is clumpy enough.

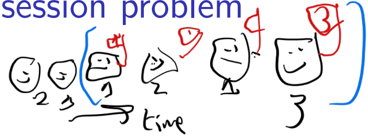
Lower bound on global synchronizers

- Intuition: Can't synchronize globally without global communication.
- It takes D time to get a message across a network.
- So $\geq D$ time to do one round of a global synchronizer.

Formalizing this intuition is tricky.

The session problem

$n=3$



$(1\ 2\ 2\ 2\ 1\ 3)$

session = every ≥ 1

Described by (Arjomandi, Fischer, and Lynch 1983).

We'll do a lower bound of (Attiya and Mavronicolas 1994).

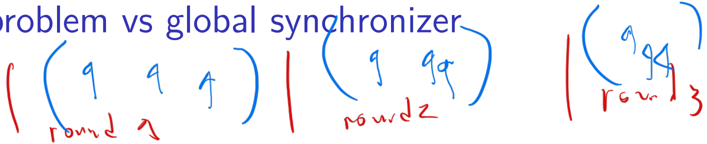
Definition:

- Every process has a **special action** (local computation event).
- Session = minimal interval in which every process does at least one special action.
- Goal: Pack as many sessions as possible into a given time bound.

5-session alg $\Rightarrow$ ≥ 5 sessions

what is time complexity?

Session problem vs global synchronizer



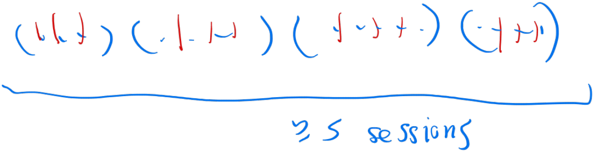
Upper bound on global synchronizer $\Rightarrow$ upper bound on session time.

- Suppose we have a global synchronizer
- Make special action = computation event to deliver synchronous messages.
- Each synchronous round = one session.

Example: beta synchronizer gives one session every $2D + 2$ steps.

Contrapositive: lower bound on session time $\Rightarrow$ lower bound on global synchronizer.

Lower bound on the session problem



Theorem: Any asynchronous s-session protocol takes $> (s-1)D$ time in the worst case.

Proof outline: Given synchronous $(s-1)D$-time execution, apply causal shuffle to get only $s-1$ sessions.

Details of proof

$(s-1)D$ time total

Start with synchronous schedule

$\leq D$ rounds

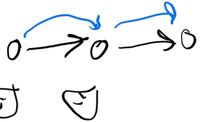
γ_i

$$\gamma = \gamma_1\gamma_2\gamma_3 \ldots \gamma_{s-1}\delta$$

where

- γ is synchronous execution.
- Each γ_i is $\leq D$ rounds.
- (Possibly empty) suffix δ has no special actions.

2 rounds



Let p and q be processes at distance D.

Claim: For any event e_p of p and e_q of q in γ_i,

- $e_p \not\Rightarrow_\gamma e_q$.
- $e_q \not\Rightarrow_\gamma e_p$.

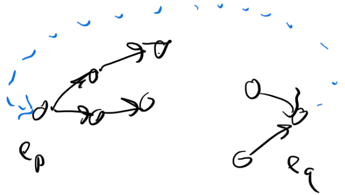
Proof: Longest message chain in D rounds can't reach q from p or vice versa.

The causal shuffle

$\underbrace{\boxed{pppp}}_{\alpha_i}$ $\underbrace{\boxed{qqqq}}_{\beta_i}$

Claim: Can reorder γ_i as $\alpha_i\beta_i$ where

- No q events in α_i
- No p events in β_i



Proof:

For all events e_p of p and e_q of q in γ_i:

1. Have $e_q \not\Rightarrow_{\gamma_i} e_p$.
2. Adding $e_p < e_q$ doesn't create a contradiction.
3. Order γ_i using $\Rightarrow_{\gamma_i} \cup \{e_p < e_q\}$.
4. This is a causal shuffle $\Rightarrow$ indistinguishable to all processes.

Similarly, can reorder γ_i as $\beta_i\alpha_i$ with same constraints.

The bad schedule

$$\underbrace{ppp}_{\alpha_1}\underbrace{qqq}_{\beta_1}\ \underbrace{qq}_{\beta_2}\ \underbrace{ppp}_{\alpha_2}$$

- For i odd, reorder γ_i as $\alpha_i\beta_i$
- For i even, reorder γ_i as $\beta_i\alpha_i$.

New schedule is

$$\gamma' = \overbrace{\alpha_1}^{p}\overbrace{\beta_1\beta_2}^{q}\overbrace{\alpha_2\alpha_3}^{p}\overbrace{\beta_3\beta_4}^{q}\alpha_4\ldots\alpha_{s-1}\beta_{s-1}\delta.$$

- No session inside just α segments or just β segments.
- $\Rightarrow$ Every session includes at least one boundary $\alpha_i\beta_i$ or $\beta_i\alpha_i$.
- $\Rightarrow$ At most $s-1$ sessions.

Note: It's possible that γ' takes more than $(s-1)D$ time.

All we show is that any algorithm that guarantees s sessions in all executions does $> (s-1)D$ time in at least one execution.

Tightness of the lower bound

Consider length-D path:

$$0 \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow \cdots \longleftrightarrow (D-1) \longleftrightarrow D$$

Algorithm: Pass a token back and forth:

- Upon receiving token, do special action and pass it on.
- For endpoints, do two special actions then send back.

This gives s sessions in sD time.

Summary

- Alpha synchronizer gives local synchrony with no slowdown.
- Beta synchronizer gives global synchrony with $O(D)$ slowdown.
- $O(D)$ is best possible due to session problem.

Critical assumption: No failures! 

Starting next week: Failures!