Distributed Synchronization

«A man with one clock knows what time it is – a man with two is never sure.»
Synchronous vs Asynchronous

Synchronous algorithms simple to design and reason about:

Send...

... receive...

... compute.
Synchronous vs Asynchronous

Asynchronous systems hard:
- Remember BFS? Agree when next phase, counters, …

Idea: How to emulate synchronous in asynchronous?
Local Synchronization

In round i:
- send message to all neighbors including usual packets of round i (of sync. algo, if any) plus also include round information
- go to next round when all neighbors have sent round i infos too

Idea of α-synchronizer: works with clock pulses!
Definitions

Synchronizer
A (distributed) synchronizer generates clock pulses (PULSE) at each node of the network.

Valid Clock Pulse
A pulse generated at some node $v$ is valid, iff it is generated after $v$ received all the messages of the synchronous algorithm sent to $v$ by its neighbors in the previous pulses.
Synchronizer

A (distributed) synchronizer generates clock pulses at each node of the network.

What is overhead? When to generate valid pulses?

synchronous algorithm sent to v by its neighbors in the previous pulses.
Overhead of Using Synchronizers

Overhead: Message needed for each pulse, independent of protocol data!

**Overhead**

Say $T(A)$ and $M(A)$ are time and message complexity of synchronous algorithm $A$, and $T(S)$ and $M(S)$ are complexities of a synchronizer for each pulse. Moreover, $T_{\text{init}}(S)$ and $M_{\text{init}}(S)$ to set up synchronizer. Then:

$$T = T_{\text{init}}(S) + T(A)*(1+T(S)), \quad M = M_{\text{init}}(S) + M(A) + T(A)*M(S)$$

**Proof.**

- Setup synchronizer clear: one time cost
- Each round additionally costs time $T(S)$ and messages $M(S)$

In the following, we will ignore initialization costs.
Definitions

**Safe Node**

A node $v$ is **safe** wrt certain clock pulse if all messages of the synchronous algorithm sent by $v$ in that pulse have already arrived *at their destination*.

$v$ not safe wrt 10

$v$ safe wrt 10
Synchronizer

Next Pulse

If all neighbors of a node $v$ are safe wrt the current clock pulse of $v$, the next pulse can be generated for $v$ (valid!).

Proof.

If all neighbors of $v$ are safe wrt that pulse, $v$ has received all messages of the given pulse! So if $v$ generates a new pulse now, it must be valid!

How to detect? ACKs (reached dest!), at most factor 2 more messages
The Local Synchronizer $\alpha$

**Synchronizer $\alpha$**

At node $v$:
- wait until $v$ is safe (learn via ACKs)
- send SAFE to all neighbors
- wait until $v$ receives SAFE messages from all neighbors
- start new PULSE

Via ACKs, node detects when it is safe (= all its messages arrived at destination), the reports it to neighbors.
When all neighbors are safe (= all their neighbors including $v$ got their messages), it continues.

No initialization needed!

Overhead?
Analysis of Local Synchronizer $\alpha$

**Synchronizer $\alpha$**

Overhead per synchronous round:
- $T(\alpha) = O(1)$
- $M(\alpha) = O(m)$

Why?
- Communication only between neighbors
- As soon as all neighbors of a node $v$ become safe, $v$ will know after one additional round
- Every edge sees at most 6 messages (PULSE, SAFE, ACK)
  (but unfortunately, maybe in reality no message would have been sent in sync algo)

Happy?
Analysis of Local Synchronizer $\alpha$

**Synchronizer $\alpha$**

Overhead per synchronous round:
- $T(\alpha) = O(1)$
- $M(\alpha) = O(m)$

Time complexity good, but too much communication!

Ideas? Recall «Flooding»...
The Global Synchronizer $\beta$

Idea: make global synchronization:
Pre-compute a leader and a spanning tree: convergecast to next phase!

SAFE: round $x$ done!
Idea: make global synchronization:
Pre-compute a **leader** and a **spanning tree**: convergecast to next phase!

PULSE: Go to next round x+1
The Global Synchronizer $\beta$

Idea: make global synchronization:
Pre-compute a leader and a spanning tree: convergecast to next phase!

Synchronizer $\beta$

At node $v$:
- wait until $v$ is safe
- wait until $v$ receives SAFE message from all its children in tree
- send SAFE message to parent in $T$
- wait until PULSE received from parent
- send PULSE to children
- start PULSE

Complexity?
The Global Synchronizer $\beta$

**Synchronizer $\beta$**

Complexities per synchronous round:

- $T(\beta) = O(\text{diam } T) = O(n)$
- $M(\beta) = O(n)$

Why?
- $T(\beta)$ and $M(\beta)$: follow from convergecast
- Initialization: improved GHS gives $T_{\text{init}}(\beta) = O(n)$, $M_{\text{init}}(\beta) = O(m + n \log n)$

Local synchronizer faster, global synchronizer less messages!
How to combine?
The Hybrid Synchronizer $\gamma$

Idea: Make $\beta$ in dense areas and $\alpha$ in sparse areas
The Hybrid Synchronizer $\gamma$

Idea:

Partition network into small-diameter clusters.
The Hybrid Synchronizer $\gamma$

Idea:

Partition network into small-diameter clusters.

Dense but small diameter, so do synchronizer $\beta$!
The Hybrid Synchronizer $\gamma$

Idea:

Between clusters, do synchronizer $\alpha$!
(See it as graph where clusters collapsed.)

Partition network into small-diameter clusters.
The Hybrid Synchronizer γ

Idea:

Each cluster has leader and BFS spanning tree!
Idea:

- **Intra-cluster tree edge**
- **Intra-cluster edge** (not tree)
- **Inter-cluster edge**
- **Edge between clusters**
The Hybrid Synchronizer γ

Idea:

Partition network into small-diameter clusters.

Cluster safe if all its nodes safe.
The Hybrid Synchronizer $\gamma$

Idea:
1. Phase 1: Apply Synchronizer $\beta$ in each cluster; when done inform leaders in neighbor clusters
2. Phase 2: Generate next pulse when neighbor clusters are safe (Synchronizer $\alpha$)

Synchronizer $\gamma$

For node $v$:
- wait until $v$ is safe
- wait until $v$ receives SAFE from all children in intra-cluster tree
- send SAFE to parent in tree
- wait for CLUSTERSAFE message from parent
- send CLUSTERSAFE to children
- wait until NEIGHBORSAFE received from all incident inter-cluster edges and children in intra-cluster
- send NEIGHBORSAFE to parent
- wait for PULSE and forward
The Hybrid Synchronizer $\gamma$

Idea:
1. Phase 1: Apply Synchronizer $\beta$ in each cluster; when done inform leaders in neighbor clusters
2. Phase 2: Generate next pulse when neighbor clusters are safe (Synchronizer $\alpha$)

Synchronizer $\gamma$

For node $v$:
- wait until $v$ is safe
- wait until $v$ receives SAFE from all children in intra-cluster tree
- send SAFE to parent in tree
- wait for CLUSTERSAFE message from parent
- send CLUSTERSAFE to children
- wait until NEIGHBORSAFE received from all incident inter-cluster edges and children in intra-cluster
- send NEIGHBORSAFE to parent
- wait for PULSE and forward

Complexity?
**Synchronizer $\gamma$**

Let $m_c$ be number of inter-cluster edges and let $k$ be the maximum cluster radius (max dist leaf to leader). Then:

$$T(\gamma) = O(k)$$
$$M(\gamma) = O(n+m_c)$$

Why?
Synchronizer $\gamma$

Let $m_c$ be number of intercluster edges and let $k$ be the maximum cluster radius (max dist leaf to leader). Then:

$$T(\gamma) = O(k)$$
$$M(\gamma) = O(n + m_c)$$

Why?

- **Time:**
  - Depth of tree enough (among all neighbor clusters).

- **Messages:**
  - intra-cluster tree edges see SAFE, CLUSTERSAFE, NEIGHBORSAFE, PLUSE, plus ACKs. It’s trees, so $O(n)$.
  - one NEIGHBORSAFE for each inter-cluster edge
Synchronizer $\gamma$

Let $m_c$ be number of intercluster edges and let $k$ be the maximum cluster radius (max dist leaf to leader).

Then:

\[ T(\gamma) = O(k) \]
\[ M(\gamma) = O(n + m_c) \]

**Why?**

- **Time:**
  - Depth of tree enough (among all neighbor clusters).

- **Messages:**
  - intra-cluster tree edges see SAFE, CLUSTERSAFE, NEIGHBORSAFE, PLUSE, plus ACKs. It’s trees, so $O(n)$.
  - one NEIGHBORSAFE for each inter-cluster edge

Want to minimize $k$ and $m_c$: How to partition?
Possible Partitions

- What happens if each single node is its own partition?
- What if everything is one big cluster?
Possible Partitions

- What happens if each single node is its own partition?
- What if everything is one big cluster?

**What’s our synchronizers!**
Network Partition

Idea:
1. Construct one cluster after another; start cluster at random non-covered node
2. Grow as long as “growth significant” (factor $\rho$)

Define: $B(v,r) = \text{Ball of radius } r \text{ around } v$

Cluster Construction

while unprocessed nodes:
  select arbitrary unprocessed node $v$
  $r:=0$
  while $|B(v,r+1)| > \rho \times |B(v,r)|$ do
    $r := r+1$
  end while
  makeCluster($B(v,r)$)
end while
Partition Properties

The resulting network partition:
(1) consists of clusters of radius at most $\log_\rho n$
(2) at most $(\rho - 1)n$ intercluster edges
Quality of Partition

Partition Properties

The resulting network partition:

1. consists of clusters of radius at most \( \log_\rho n \)
2. at most \((\rho - 1)^*n\) intercluster edges

Why?

- Cluster radius:
  - Radius grows only if cluster size increases by factor \( \rho \)
  - As there are at most \( n \) nodes, this can happen at most \( \log_\rho n \) times

- Intercluster edges:
  - Intercluster edge connects node at border of cluster with node outside cluster
  - Consider cluster \( C = B(v, r) \): we know \( |B(v, r+1)| \leq \rho \cdot |B(v, r)| \).
  - The number of nodes adjacent to a cluster can hence be at most \( |B(v, r+1) \setminus B(v, r)| \leq \rho \cdot |C| - |C| \)
  - There is only one intercluster edge per cluster-pair, so at most \( \rho \cdot |C| - |C| \) per cluster
  - Summing over all cluster (\( n \) nodes in total): \( \sum (\rho - 1)^* |C| = (\rho - 1)^* \sum |C| = (\rho - 1)^* n \)
Partition Properties

The resulting network partition:

1. consists of clusters of radius at most $\log_\rho n$
2. at most $(\rho - 1)n$ intercluster edges

Asymptotically optimal tradeoff!
Partition Properties

The resulting network partition:
1. consists of clusters of radius at most $\log_\rho n$
2. at most $(\rho - 1) n$ intercluster edges

Example: $\rho = 2$
Logarithmic time, linear number of intercluster edges

Example: $\rho = n^{1/k}$
k time, $O(n^{1+1/k})$ intercluster edges
Remarks

- Try to make synchronous BFS algorithm asynchronous with $\gamma$ synchronizer

- Bad about our solution: all nodes need to participate in synchronization even if they are “active” only during small algo period
End of Lecture