236358
Distributed Graph Algorithms

Spring 2017
Class by Keren Censor-Hillel
Asynchronous model

• No timing guarantees on delivery of messages

• **Complexity measures:**
  – Number of messages
  – Time: worst case number of time units assuming each message takes at most single unit

• **Reminder:** We saw two asynchronous algorithms for constructing a BFS tree
Synchronizers

• Designing algorithms for synchronous systems is easier
  – Synchrony improves predictability

• Can we construct a simulator, to which we feed a synchronous algorithm, and we get an asynchronous algorithm?

• Such a simulator is called a synchronizer.
Synchronizers – Take I

Send \((r+1)\)-message after receiving all round-\(r\) messages

\(v\) will wait forever if \(u\) never sends a round-\(r\) message

Sending an empty message increases message complexity
Synchronizers – Formal

• Requirements from a synchronizer SYNCH:
  
  – Given a synchronous algorithm $S$, SYNCH produces an asynchronous algorithm $A$
  
  – For every execution $\pi_S$ of $S$ on a graph $G$ with inputs $IN$, $A$ produces an execution $\pi_A$ of $A$
Synchronizers – Formal

• Every node $v$ maintains a \textit{round} $r_v$ variable

• The local state of any local variable $X_v$ in $\pi_A$ when $r_v = r$ is the same as its local state at the beginning of round $r$ in $\pi_S$
Synchronizers – Formal

• The original message sent/received by $v$ to $w$ in $\pi_A$ when $r_v = r$ is the same as the message it sends/receives in round $r$ of $\pi_S$

• The output of $v$ in $\pi_A$ is the same as its output in $\pi_S$
Synchronizers - Complexity

• The synchronizer SYNCH may perform some setup stage, requiring $M_{\text{init}}(\text{SYNCH})$ messages and $T_{\text{init}}(\text{SYNCH})$ time.

• Every round requires $M_{\text{round}}(\text{SYNCH})$ messages and $T_{\text{round}}(\text{SYNCH})$ time.
Synchronizers - Complexity

- The message and time complexities of the asynchronous algorithm $A$ are:

$$M(A) \leq M_{\text{init}}(\text{SYNCH}) + M(S) + T(S) \cdot M_{\text{round}}(\text{SYNCH})$$

$$T(A) \leq T_{\text{init}}(\text{SYNCH}) + T(S) \cdot T_{\text{round}}(\text{SYNCH})$$
Synchronizers – Take I

Send \((r+1)\)-message after receiving all round-\(r\) messages

\(v\) will wait forever if
\(u\) never sends a round-\(r\) message

Sending an empty message increases message complexity

\[ M_{\text{round}}(\text{SYNCH}) = O(m) \]
\[ T_{\text{round}}(\text{SYNCH}) = O(1) \]
## Synchronizers

<table>
<thead>
<tr>
<th>Synchronizer</th>
<th>$M_{\text{round}}(\text{SYNCH})$</th>
<th>$T_{\text{round}}(\text{SYNCH})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty messages</td>
<td>$O(m)$</td>
<td>1</td>
</tr>
</tbody>
</table>
Synchronizers – Acks

Sending ACKs:

- \( v \) sends its messages to neighbors
- Neighbors send ACKs to \( v \)
- \( v \) informs receiving all ACKs (\( v \) is safe)
- \( v \) sends round-(\( r+1 \)) messages when all neighbors are safe
Synchronizers – Acks

• **Correctness**: If \( v \) receives a *safe* message from every \( u \) in \( N(v) \), then every such \( u \) received an ACK from all nodes in \( N(u) \) to which it sent messages.

• In particular, for every \( u \) in \( N(v) \), either \( v \) received the message from \( u \), or \( u \) did not send any message to \( v \).
Synchronizers – Acks

• Hence, when $v$ sends messages for the next round, it has a correct state from the previous round.
Synchronizers – Acks

- **Complexity**: Still need a (safe) message from every neighbor
- Message overhead is $M_{\text{round}}(\text{SYNCH}) = O(m)$
- Time overhead is $T_{\text{round}}(\text{SYNCH}) = O(1)$
## Synchronizers

<table>
<thead>
<tr>
<th>Synchronizer</th>
<th>$M_{\text{round}}(\text{SYNCH})$</th>
<th>$T_{\text{round}}(\text{SYNCH})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty messages</td>
<td>$O(m)$</td>
<td>1</td>
</tr>
<tr>
<td>ACKs + safe</td>
<td>$O(m)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Synchronizers – Spanning tree

$E_T$ is a rooted spanning tree
- Nodes send messages and ACKs
- A leaf $u$ sends a *safe* message to its parent after receiving all ACKs
- An inner node send a *safe* message to its parent after receiving all ACKs AND all *safe* messages from its children
Synchronizers – Spanning tree

$E_T$ is a rooted spanning tree

- Root sends **all_safe** message down the tree after receiving all ACKs AND all **safe** messages from children
Synchronizers – Spanning tree

- **Correctness**: By induction: If $u$ receives a safe message from every child in $E_T$, then every node $w$ in the subtree of $u$ received an ACK from all nodes in $N(w)$ to which it sent messages.
Synchronizers – Spanning tree

- Hence, when the root \( v \) sends the *all_safe* message, all nodes have received ACKs, and so all nodes have received the messages sent to them.

- Thus, for the next round, all nodes have a correct state from the previous round.
Synchronizers – Spanning tree

• **Complexity**: safe and all_safe messages are sent only on edges of $E_T$
• Message overhead is $M_{round}(SYNCH) = O(n)$
• But the time costs as the depth of the tree
• Time overhead is $T_{round}(SYNCH) = O(\text{depth}(E_T))$
## Synchronizers

<table>
<thead>
<tr>
<th>Synchronizer</th>
<th>$M_{\text{round}}(\text{SYNCH})$</th>
<th>$T_{\text{round}}(\text{SYNCH})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty messages</td>
<td>$O(m)$</td>
<td>1</td>
</tr>
<tr>
<td>ACKs + safe</td>
<td>$O(m)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Spanning tree $E_T$</td>
<td>$O(n)$</td>
<td>$O(\text{depth}(E_T))$</td>
</tr>
</tbody>
</table>

Keren Censor-Hillel, Spring 2017
Spanners

• Given $G=(V,E)$ and $E_S$ in $E$, a subgraph $S=(V,E_S)$ is called a **k-spanner** of $G$ if:
  – For every $u,v$ in $V$:
    $$\text{dist}_S(u,v) \leq k \cdot \text{dist}_G(u,v)$$

• $k$ is called the **stretch** of the spanner
Synchronizers – Spanners

$E_S$ is a $k$-spanner with $m_S$ edges

• Nodes send messages and ACKs

• After receiving all ACKs, repeat for $k$ iterations:
  – Send safe messages in the spanner
  – Wait for safe messages in the spanner

2-spanner
Synchronizers – Spanners

• **Correctness:** For every node $v$, by induction, after iteration $t$, every node $u$ such that $\text{dist}_{ES}(u,v) \leq t$ has received all ACKs.

• Base case, $t=0$: $v$ received all ACKs

• **Induction hypothesis:** after iteration $t-1$, every node $u$ such that $\text{dist}_{ES}(u,v) \leq t-1$ has received all ACKs.
Synchronizers – Spanners

- **Induction step**: Every node $u$ such that $\text{dist}_{\text{ES}}(u,v) = t$ has a $w$ in $\text{N}_{\text{ES}}(v)$ for which $\text{dist}_{\text{ES}}(w,u) = t-1$.

- When $v$ receives a *safe* message from $w$ in iteration $t$, then by the induction hypothesis for $w$, $u$ has received all ACKs.
Synchronizers – Spanners

• For every neighbor $w$ in $N(v)$, it holds that $\text{dist}_{ES}(w,v) \leq k$, because $E_S$ is a $k$-spanner.

• After $k$ iterations, every $w$ in $N(v)$ has received all ACKs, so $v$ received all the messages sent to it.
Synchronizers – Spanners

- **Complexity**: Every round requires $k$ iterations, in each iteration a message is sent on every spanner edge.
  - $M_{\text{round}}(\text{SYNCH}) = O(km_s)$
  - $T_{\text{round}}(\text{SYNCH}) = O(k)$
## Synchronizers

<table>
<thead>
<tr>
<th>Synchronizer</th>
<th>$M_{\text{round}}(\text{SYNCH})$</th>
<th>$T_{\text{round}}(\text{SYNCH})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty messages</td>
<td>$O(m)$</td>
<td>1</td>
</tr>
<tr>
<td>ACKs + safe</td>
<td>$O(m)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Spanning tree $E_T$</td>
<td>$O(n)$</td>
<td>$O(\text{depth}(E_T))$</td>
</tr>
<tr>
<td>k-spanner $E_S$</td>
<td>$O(km_S)$</td>
<td>$O(k)$</td>
</tr>
</tbody>
</table>
Constructing a Spanner from a Synchronizer

**SYNCH** is a synchronizer

Mark all edges used

Information has to pass between each pair of neighbors

Gives a spanner $S$ with $m_S \leq M_{\text{round}}(\text{SYNCH})$ and $k \leq T_{\text{round}}(\text{SYNCH})$