236610
Distributed Graph Algorithms

Spring 2018
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
Asynchronous BFS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message complexity</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update-based</td>
<td>$O(nm)$</td>
<td>$O(D)$</td>
</tr>
<tr>
<td>Root-controlled</td>
<td>$O(m+nD)$</td>
<td>$O(D^2)$</td>
</tr>
</tbody>
</table>
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 \texttt{SYNCH}:

  – Given a synchronous algorithm \texttt{S}, \texttt{SYNCH} produces an asynchronous algorithm \texttt{A}

  – For every execution $\pi_S$ of \texttt{S} on a graph \texttt{G} with inputs \texttt{IN}, \texttt{A} produces an execution $\pi_A$ of \texttt{A}
Synchronizers – Formal

• Every node $v$ maintains a 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 \( \text{SYNCH} \) 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 $\mathcal{N}(v)$, then every such $u$ received an ACK from all nodes in $\mathcal{N}(u)$ to which it sent messages.

• In particular, for every $u$ in $\mathcal{N}(v)$, 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
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 $T_{round}(SYNCH) = O(depth(E_T))$
## 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>
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
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 $\text{dist}_{ES}(u,v) = t$ has a $w$ in $N_{ES}(v)$ for which $\text{dist}_{ES}(w,u) = t-1$.

- 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 $k \leq T_{\text{round}}(\text{SYNCH})$
GOSSIP model

In each round, **contact a single neighbor** to exchange information with

**Why?**

- Initiating communication may be expensive
- Reduce network traffic
Information Spreading

• Each node needs to learn the inputs of all other nodes

• Also called:
  – All-to-all Dissemination
  – Rumor Spreading
  – ...

Keren Censor-Hillel, Spring 2018
Information Spreading

In LOCAL model takes $D$ rounds

In GOSSIP model:
- Deterministic round-robin algorithm: $\Delta \cdot D$
  May need to contact all neighbors before reaching bottleneck
- Randomized algorithm?

3 rounds in LOCAL model
$O(n)$ rounds in GOSSIP model
Randomized Information Spreading

Randomized:

• Complete graph – $O(\log n)$ rounds
Information spreading

– The problem with the barbell graph is that it has
  • Large degrees, and
  • Bad connectivity

– Large degree alone is not a problem

– Bad connectivity alone is not a problem

$O(\log n)$ rounds

$\Theta(n)$, but we cannot hope for anything better
Randomized information spreading

• General graphs analyzed in terms of their conductance $\Phi$

\[ 0 \leq \Phi \leq 1, \text{ measure of connectivity} \]

\[ \Phi = \min_S \varphi(S) \]

\[ \varphi(S) = \text{Number of edges touching } S \]

$\Phi \approx 1/n^2$
Conductance - Examples

• Clique

\[ \Phi(\text{clique}) = \left( \frac{n \cdot n}{2 \cdot 2} \right) / \left( \frac{n}{2} \cdot n \right) = O(1) \]

• Path

\[ \Phi(\text{path}) = O\left( \frac{1}{n} \right) \]
Randomized information spreading

\(O\left(\frac{\log(n)}{\Phi_G}\right)\) rounds for all graphs \(G\)

\(O(n^2\log(n))\)
Information Spreading - Examples

- **Clique**
  \[ \Phi(\text{clique}) = O(1) \]
  Information spreading in \( O(\log n) \) rounds

- **Path**
  \[ \Phi(\text{path}) = O\left(\frac{1}{n}\right) \]
  Information spreading in \( O(n \log n) \) rounds