Verification of Peer-to-Peer Algorithms: A Case Study

Rana Bakhshi
Dilian Gurov

Royal Institute of Technology (KTH),
Sweden
What is P2P?

- Distributed
  - No centralized control
  - Nodes are symmetric in function
- Dynamic
  - Nodes to join and leave the network
- Structured (optional)
  - Network topology maintenance needed
Chord: Protocol

• Provides P2P lookup service
• Given a key, it maps the key onto a node
• Maintains routing information as nodes join and leave the system
  – stabilization algorithm
• and other features…
Chord: Protocol

- **Chord ring**
  - ID are ordered on a ID circle

- **Two basic neighbours**
  - predecessor and successor

- **Successor**
  - the first node
  - ID is more or equal to the current
Chord: Stabilization algorithm

• All nodes information must be up to date
  • for correctly executing lookups
• Each node periodically runs a stabilization algorithm
  • to update the successor information

• Case study
  • Verification of the stabilization algorithm
Chord: Stabilization algorithm

- n runs `$join()`:
  1. predecessor = nil
  2. n acquires $n_s$ as successor from $n'$

- n runs `$stab()`:
  1. n notifies $n_s$ being the new predecessor
  2. $n_s$ acquires n as its predecessor

- $n_p$ runs `$stab()`:
  1. $n_p$ asks $n_s$ for its predecessor (now n)
  2. $n_p$ acquires n as its successor
  3. $n_p$ notifies n
  4. n acquires $n_p$ as its predecessor

- predecessor and successor information are now updated
Motivation

• A reasonable level of abstraction at which to perform the verification
  – Correctness proofs
    • sketch at the high level of abstraction
    • tend to provide no operational semantics
  – Model checking techniques
    • not directly applicable
    • systems are inherently dynamic
    • have infinite state behaviour
Modelling

• Assumptions
  – No finger tables and successors’ lists
  – Pure Join Model

• Formalism
  – $\pi$-calculus

• Strategy
  – establishing weak bisimulation
Ring behaviour:

- A node performs an output on its channel
- and passes the token to its successor
Specification

\[ \text{Ring}(k_i, m) \triangleq \ldots + \sum_{k_j \in \mathcal{I} \setminus m} \tau \cdot \text{Ring}(k_i, m \oplus k_j) \]

- Ring behaviour:
  - A new node joins the ring
  - and passes the token to its successor
Implementation

- A collection of concurrent processes
  - nodes $A$
  - nodes $P$ ready to join a network
  - a token

\[ Impl(k, m) \triangleq (\overrightarrow{vin^{k_i}}, \overrightarrow{in^{k_i}}, \overrightarrow{in^{k_l}})((\overrightarrow{in^{k_i}} \cdot 0)(\prod_{k_i \in I \setminus m} k_0 \in m') \sum_{k_i \in I \setminus m} P(\overrightarrow{in^{k_j}}, \overrightarrow{in^{k_0}})) \]

\[ \prod_{k_i' \in m'' \subseteq m} A(\overrightarrow{id^{k_i'}}, \overrightarrow{in^{k_i'}}, s^{k_i'}, p^{k_i'}) \prod_{k_l \in m' \subseteq m} A(\overrightarrow{id^{k_l}}, \overrightarrow{in^{k_l}}, s^{k_l}, p^{k_l}) \]
Implementation

\[ \text{Impl}(k_i, m) \triangleq \ldots | \left( \prod_{k_j \in \mathcal{I} \setminus m} \sum_{k_v \in m'} P(\mathbf{in}^{k_j}_i, \mathbf{in}^{k_v}_4) \right) | \ldots \]

- **Process P**
  - A node that is not in the network but may join it
  - Implements \textit{join()}

\[ P(\mathbf{in}^{k_j}_i, \mathbf{in}^{k_i}_4) \triangleq (\nu \mathbf{in}^{k_j}_i) \left( \mathbf{in}^{k_i}_4 \langle \mathbf{in}^{k_j}_i \rangle . \mathbf{in}^{k_j}_4 (\mathbf{in}^z) . A(id^{k_j}_i, \mathbf{in}^{k_j}_i, \mathbf{in}^{k_z}_i, \bot) \right) \]
Implementation

\[ Impl(k_i, m) \triangleq \ldots \prod_{k_i \in m' \subseteq m} A(id_{k_i}, in_{k_i}, s_{k_i}, p_{k_i}) \]

- **Process A**
  - 5 ports for different type of messages and 1 port for output ID
  - stores info of itself and neighbours
  - implements \textit{stab()}
Conclusion

- Extends results on lookup algorithm for the static case
- A model for P2P networks with a ring topology
  - Process \textit{Ring}
- $\pi$-calculus offers a suitable theory
Future Work

• Other aspects of P2P networks
  – including presence of failures in the network
  – adding finger tables and successors lists

• Theorem proving systems
  – models can be formalized
  – the bisimulation proof can be carried
  – e.g. Isabelle/HOL
Thank you!
Chord: Stabilization algorithm

% new node joins a Chord ring containing node n'
join() →
n'! {'find_successor',n}
receive
    {'find_successor',n'} →
        succ := n'; pred := nil;
end.

% in parallel with

loop
receive
    {'find_successor',n'} →
        if n'∈(n,succ) →
            n'! {'find_successor',succ}
              n'∉(n,succ) →
        succ! {'find_successor',n'}
end
end.

% stabilization algorithm

stab() →
succ! {'req_predecessor',n}
receive
    {'resp_predecessor',n'} →
        if n'∈(n,succ) →
            succ := n';
        succ! {'notify',n}
end.

% in parallel with

loop
receive
    {'req_predecessor',n'} →
        n'! {'resp_predecessor',pred}
        {'notify',n'} →
            if pred = nil ∨ n'∈(pred,n) →
                pred := n';
end
end.
new node join a Chord ring containing node n'

join() ->
n'! {"find_successor",n}
receive
  {"find_successor",n'} ->
  succ := n'; pred := nil;
end.

%in parallel with

loop
  receive
    {"find_successor",n'} ->
    if n'∈(n,succ] ->
      n'! {"find_successor",succ}
      n'∉(n,succ] ->
      succ! {"find_successor",n'}
  end
end.

%stabilization algorithm

stab() ->
succ!{"req_predecessor",n}
receive
  {"resp_predecessor",n'} ->
  if n'∈(n,succ) ->
    succ := n';
    succ!{"notify",n}
  if (n'∉(n,succ) ∧ (n'≠n)) ->
    succ!{"notify",n}
end.

%in parallel with

loop
  receive
    {"req_predecessor",n'} ->
    n'!{"resp_predecessor",pred}
    {"notify",n'} ->
    if pred = nil ∨ n'∈(pred,n) ->
      pred := n';
      if (pred = n) ∧ (succ = n)
      ->
        succ := n';
        succ!{"notify",n}
    end
end.
## \( \pi \text{-calculus} \)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u, v )</td>
<td>( o, p, x )</td>
<td>names</td>
</tr>
<tr>
<td></td>
<td>( in, id )</td>
<td>channels</td>
</tr>
<tr>
<td></td>
<td>( \bot )</td>
<td>undefined value</td>
</tr>
<tr>
<td></td>
<td>( k_i, k_j )</td>
<td>integers, IDs</td>
</tr>
<tr>
<td></td>
<td>( \text{succ}(x, m) )</td>
<td>successor</td>
</tr>
<tr>
<td></td>
<td>( \text{pred}(x, m) )</td>
<td>predecessor</td>
</tr>
<tr>
<td>( e, e', e'' )</td>
<td>( u )</td>
<td>expressions</td>
</tr>
<tr>
<td>( \phi, \psi )</td>
<td>( e = e' )</td>
<td>boolean tests</td>
</tr>
<tr>
<td></td>
<td>( e \in (e', e'') )</td>
<td>interval check</td>
</tr>
<tr>
<td>( m, m', m'' )</td>
<td>sets, subsets</td>
<td></td>
</tr>
<tr>
<td>( \pi )</td>
<td>( \tau \mid \overline{p}(\overline{v}) \mid p(\overline{v}) )</td>
<td>prefix</td>
</tr>
<tr>
<td></td>
<td>( (\overline{v}).P )</td>
<td>abstractions</td>
</tr>
<tr>
<td></td>
<td>( \overline{u} \langle \overline{v} \rangle .P )</td>
<td>concretions</td>
</tr>
<tr>
<td>( Q, R )</td>
<td>processes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( M )</td>
<td>summation</td>
</tr>
<tr>
<td></td>
<td>( (Q \mid R) )</td>
<td>parallel composition</td>
</tr>
<tr>
<td></td>
<td>( (\nu \overline{p}) Q )</td>
<td>restriction</td>
</tr>
<tr>
<td></td>
<td>( \text{if } \psi \text{ then } Q \text{ else } R )</td>
<td>if statement</td>
</tr>
<tr>
<td></td>
<td>( !Q )</td>
<td>replication</td>
</tr>
<tr>
<td></td>
<td>( Q \langle \overline{v} \rangle )</td>
<td>process constant</td>
</tr>
<tr>
<td>( M, M' )</td>
<td>( 0 )</td>
<td>inaction</td>
</tr>
<tr>
<td></td>
<td>( \pi.Q )</td>
<td>process action</td>
</tr>
<tr>
<td></td>
<td>( M + M' )</td>
<td>choice</td>
</tr>
</tbody>
</table>
\( \pi\text{-calculus} \)

\[
\text{succ}(x, \hat{m}) = \{ y \in \hat{m} \mid (y \downarrow x) = \min\{ (z \downarrow x) \mid z \in \hat{m} \} \land \hat{m} \subseteq I \}
\]

\[
\text{pred}(x, \hat{m}) = \{ y \in \hat{m} \mid (y \downarrow x) = \max\{ (z \downarrow x) \mid z \in \hat{m} \} \land \hat{m} \subseteq I \} \cup \{ x \notin \hat{m} \mid \bot \} \]
Implementation

\[
A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p) \triangleq \overline{in}^1(\overline{in}^b).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p) \\
+ \overline{in}^1(\overline{in}^x).\overline{in}^2(\overline{in}^p).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p) \\
+ \overline{in}^2(\overline{in}^x). (\text{if } z \in (s, o) \text{ then } \overline{in}^3(\overline{in}^b).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p) \\
\quad \text{else (if } z = o \text{ then } A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p) \\
\qquad \text{else } \overline{in}^3(\overline{in}^b).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p)) \\
+ \overline{in}^3(\overline{in}^x).(\text{if } p = \bot \lor z \in (p, o) \\
\quad \text{then (if } (p = o) \land (s = o) \text{ then } \overline{in}^3(\overline{in}^b).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^x) \\
\qquad \text{else } A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^x)) \\
\quad \text{else } A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^x)) \\
+ \overline{in}^4(\overline{in}^x).(\text{if } z \in (s, o] \text{ then } \overline{in}^4(\overline{in}^b).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p) \\
\qquad \text{else } \overline{in}^4(\overline{in}^x).A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p)) \\
+ \overline{in}^5.(id^o.A(id^o, \overline{in}^b, \overline{in}^x, \overline{in}^p)|in^5_0)
\]

where: \( \overline{in}^x = \{x, in^1_1, in^2_2, in^3_3, in^4_4, in^5_5\} \).