Chapar: Certified Causally Consistent Distributed Key-Value Stores

Mohsen Lesani with CJ Bell and Adam Chlipala

PLV, CSAIL, MIT
Replicated Distributed Stores

- Fault tolerance, Partition tolerance
- Availability, Responsiveness
Replicated Distributed Stores

- Fault tolerance, Partition tolerance
- Availability, Responsiveness
- Consistency?
CAP Theorem

Impossibility of
○ Consistency
○ Availability
○ Partition Tolerance

**Theorem (CAP)**

*In a network subject to communication failures, it is impossible to implement an atomic (linearizable) read/write shared memory that guarantees a response to every request.*

Even without failures, responsiveness is in contrast to consistency.

Gilbert & Lynch. ACM SIGACT News ’02
Example: Status Update

0 $\rightarrow$
- $\text{put}(\text{Pic}, \mathcal{S});$
- $\text{put}(\text{Post}, \mathcal{C})$

1 $\rightarrow$
- $\text{post} \leftarrow \text{get}(\text{Post});$
- $\text{photo} \leftarrow \text{get}(\text{Pic});$
- $\text{assert}(\text{post} = \mathcal{C} \Rightarrow \text{photo} \neq \bot)$

Alice
▷ uploads a new photo
▷ announces it to her friends

Bob
▷ checks Alice’s post
▷ then loads her photo

Chapar: Certified Causally Consistent Distributed Stores
Mohsen Lesani
Example: Lost Ring

0 →
   put(Alice, ?);
   put(Alice, ✰)
1 →
   post ← get(Alice);
   if post = ✰ then
       put(Bob, ✰)
2 →
   post ← get(Bob);
   post' ← get(Alice);
   assert (post = ✰ ⇒ post' ≠ ?)

Alice
  ▷ “I’ve lost my ring”
  ▷ “Found it!”

Bob
  ▷ “Glad to hear it!”

Carol
Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani

Abstract Causal Operational Semantics

Programs

Algorithms
Chapar Framework

Abstract Causal Operational Semantics

Algorithm Verification

Causally Consistent
Chapar Framework

Abstract Causal Operational Semantics

Client Verification

Programs

Causally Content

Algorithm Verification

Algorithms

Causally Consistently
Abstract Operational Semantics

(1,1), {}

1 - \text{put}(k_1,v_1)

2

3
Abstract Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Abstract Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani

(1,1), {}  (1,2), {(1,1)}

{(1,1)}

put(k1,v1)  put(k2,v2)

(1,1), {}

{(1,1)}

(1,2), {(1,1)}

put(k2,v2)

1

2

3
Abstract Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Abstract Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Abstract Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani

Abstract Operational Semantics

1. \( (1,1), \{\} \)
   - \( \text{put}(k_1, v_1) \)
   - \( \text{put}(k_2, v_2) \)

2. \( (1,2), \{(1,1)\} \)
   - \( k_1 \rightarrow v_1, (1,1) \)
   - \( k_2 \rightarrow v_2, ((1,1), (1,2)) \)
   - \( \text{get}(k_2) \)
   - \( k_1 \rightarrow v_1, (1,1) \)
   - \( k_2 \rightarrow v_2, ((1,1), (1,2)) \)
   - \( (2,1), \{(1,1), (1,2)\} \)

3. \( (2,1) \)
   - \( \text{update} \)
Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Instrumented Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
put(k₁, v₁)
put(k₂, v₂)

σ₁
σ₁'
σ₁''

m₁₂, (1, 1)
m₁₃, (1, 1)
m₂₂, (1, 2)
m₂₃, (1, 2)
guard(m12)
update(m12)
put(k2,v2)put(k1,v1)
Instrumented Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Instrumented Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Instrumented Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Instrumented Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores

Mohsen Lesani
Instrumented Concrete Operational Semantics

Chapar: Certified Causally Consistent Distributed Stores
Mohsen Lesani
An algorithm is *well receptive* if and only if

- there exists a function $Rec$ such that
  - $Rec$ returns the number of updates that the input state has received from the input node i.e.
    - Init: The initial state has not received any update from any node.
    - Step: After a step, the number of received messages from a node $n$ is incremented only if the step is a put by node $n$ itself or is an update that applies the next expected update from node $n$.
    - Cause: If an update of a put is being applied that is causally dependent on another put with the identifier $(n', c')$, then the number of received messages from node $n'$ is greater than or equal to $c'$.
  - Seq: If we treat updates as simple puts, then the algorithm refines a sequential map.
Implementations

- Causal Memory
- Eiger

Ahamad et. al. Distributed Computing ’95
Lloyd et. al. NSDI ’13
Algorithm 1

\[ A_1 (\text{Algorithm 1}) \]

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (\text{store}: K \rightarrow V, \text{clock}: N \rightarrow C) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (\text{unode}: N, \text{uclock}: N \rightarrow C) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init (( v_0 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ret} (\lambda k. v_0, \lambda n. 0) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>put (self, this)(( k, v ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (s, c) \leftarrow \text{this}; )</td>
</tr>
<tr>
<td>( c' \leftarrow c[\text{self} \mapsto c[\text{self}] + 1]; )</td>
</tr>
<tr>
<td>( s' \leftarrow s[k \mapsto v]; )</td>
</tr>
<tr>
<td>( \text{ret} ((s', c'), (\text{self}, c')) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>get (self, this)(( k ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (s, c) \leftarrow \text{this}; )</td>
</tr>
<tr>
<td>( \text{ret} (s[k], (s, c)) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>guard (self, this)(( k, v, u ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (s, c) \leftarrow \text{this}; )</td>
</tr>
<tr>
<td>( (n', c') \leftarrow u; )</td>
</tr>
<tr>
<td>( \text{ret for all} (\lambda n. n \neq n' \Rightarrow c'[n] \leq c[n]) N )</td>
</tr>
<tr>
<td>( \land c'[n'] = c[n'] + 1 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>update (self, this)(( k, v, u ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (s, c) \leftarrow \text{this}; )</td>
</tr>
<tr>
<td>( (n', c') \leftarrow u; )</td>
</tr>
<tr>
<td>( c'' \leftarrow c[n' \mapsto c'[n']]; )</td>
</tr>
<tr>
<td>( s'' \leftarrow s[k \mapsto v]; )</td>
</tr>
<tr>
<td>( \text{ret} (s'', c'') )</td>
</tr>
<tr>
<td>A₂  (Algorithm 2)</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td><strong>State</strong></td>
</tr>
<tr>
<td>(store: $K \rightarrow (V, N, C)$, rec: $N \rightarrow C$, clock: $C$, dep: List[(N, C)])</td>
</tr>
<tr>
<td><strong>Update</strong></td>
</tr>
<tr>
<td>(unode: $N$, uclock: $C$, udep: List[(N, C)])</td>
</tr>
<tr>
<td><strong>init</strong> ($v₀$)</td>
</tr>
<tr>
<td>ret ($λk.(v₀, n₀, 0), λn.0, 0, [])$</td>
</tr>
<tr>
<td><strong>put</strong> (self, this)(k, v)</td>
</tr>
<tr>
<td>(s, r, c, d) ← this; c' ← c + 1; s' ← s[k ← (v, self, c')]; r' ← r[self ← c']; d' ← (self, c') :: []; ret ((s', r', c', d'), (self, c', d))</td>
</tr>
<tr>
<td><strong>get</strong> (self, this)(k)</td>
</tr>
<tr>
<td>(s, r, c, d) ← this; (v, n', c') ← s[k]; d' ← if n' ≠ n₀ then (n', c') :: d else d; ret (v, (s, r, c, d'))</td>
</tr>
<tr>
<td><strong>guard</strong> (self, this)(k, v, u)</td>
</tr>
<tr>
<td>(_, r, _, <em>) ← this; (</em>, _, d') ← u; ret forall ($λ(n', c'). r(n') ≥ c'$) d'</td>
</tr>
<tr>
<td><strong>update</strong> (self, this)(k, v, u)</td>
</tr>
<tr>
<td>(s, r, c, d) ← this; (n', c', _) ← u; s' ← s[k ← (v, n', c')]; r' ← r[n' ← c']; ret (s', r', c, d)</td>
</tr>
</tbody>
</table>

Lloyd et. al. NSDI '03
Chapar: Certified Causally Consistent Distributed Stores  Mohsen Lesani
Experimental Results

Chapar: Certified Causally Consistent Distributed Stores
Mohsen Lesani
Thanks for your attendance