From Linearizability to Eventual Consistency

Radha Jagadeesan  James Riely,
College of CDM, DePaul University, Chicago

Dec 5, 2016. Compositionality workshop
Organization of talk

Context of problem: Distributed data structures.

Problem: Correctness.

Compositionality and abstraction.

DePaul CDM Tech Report, 2016. “From Linearizability to Eventual Consistency”.

Radha Jagadeesan, James Riely, College of CDM, DePaul University, Chicago
From Linearizability to Eventual Consistency
Sequential interfaces.

eg. Integer Set.

Mutators: +0 [Add] and −0 [remove]. Return type VOID.

Accessor: ✓1, ✗1. Returns a boolean. Do not alter the state of the object.

Example traces.

\( \times 0 \hspace{1cm} +0 \hspace{1cm} ✓0 \hspace{1cm} \times 1 \)

\(+0 \hspace{1cm} +1 \hspace{1cm} ✓0 \hspace{1cm} ✓1 \hspace{1cm} -1 \hspace{1cm} ✓0 \hspace{1cm} ✗1\)
Distributed (implementation of) Set.

\[ add(0); ?0; ?1; ?1 \parallel add(1); ?1; ?0; ?0 \]

\[ \rightarrow +0 \rightarrow √0 \rightarrow X1 \rightarrow \bullet \rightarrow √1 \]

\[ \rightarrow +1 \rightarrow √1 \rightarrow X0 \rightarrow \bullet \rightarrow √1 \]
add (0); ?0; ?1; ?1 || add (1); ?1; ?0; ?0
Serialization affects performance and scalability
cap theorem : can’t have all three [Gilbert and Lynch 2002]

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Every read receives the most recent write or an error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>Every request receives a response</td>
</tr>
<tr>
<td>Partition tolerance</td>
<td>The system operates despite arbitrary messages loss</td>
</tr>
</tbody>
</table>
Resolving conflicts among mutators. Observed Remove Set. or-set: “Add wins"

Specification : +0–0+0
Short digresssion. Distributed text editors

[Attiya, Burckhardt, Gotsman, Morrison, Yang, and Zawirski, 2016]

Mutators: \( \forall a, \ a < b, \ a > b, \ -a \)  
Accessors: \( \exists a_1 \cdots a_n \)

‘Deletion wins” (compare to ORSET)

\[
\begin{align*}
!c; \ b < c; \ d > c; \ ?bcd; \ a < b; \ e > d; \ -b; \ -d; \ ?ace
\end{align*}
\]
Resume: or-set examples
Resume: or-set examples

Radha Jagadeesan, James Riely, College of CDM, DePaul University, Chicago
From Linearizability to Eventual Consistency
or-set: non-behaviors

\[ +0 \rightarrow \sqrt{1} \rightarrow x1 \rightarrow +1 \]
or-set: non-behaviors

\[
\rightarrow +0 \rightarrow \boxed{\sqrt{1}} \rightarrow \times 1 \rightarrow \\
\rightarrow +1 \rightarrow \\
\rightarrow \sqrt{0} \rightarrow +0
\]
or-set: non-behaviors

\[ +0 \rightarrow \checkmark 1 \rightarrow \times 1 \rightarrow \]

\[ +1 \rightarrow \checkmark 0 \rightarrow +0 \]

Radha Jagadeesan, James Riely, College of CDM, DePaul University, Chicago
From Linearizability to Eventual Consistency
In what sense does the or-set implement a Set?

When is implementation $U$ valid for a specification $\Sigma$:

$U \sqsubseteq \Sigma$

What are the constraints?
Constraint 1: Compositionality (a)

[Herlihy, Wing 1990] Given two separate and independent sets:

\[ L_{\Sigma_1} \cap L_{\Sigma_2} = \emptyset. \]

and two implementations, each of which is correct individually:

\[ U_1 \sqsubseteq \Sigma_1, U_2 \sqsubseteq \Sigma_2 \]

we want:

\[ U_1 \parallel U_2 \sqsubseteq \Sigma_1 \parallel \Sigma_2 \]
Constraint 2: Compositionality (b)

[Filipovic, O Hearn, Rinetzky, Yang 2009]
Let $\mathcal{P}$ be the graph implementation, which is a client of the two sets (for vertices, edges). We want:

\[
(\mathcal{P} \parallel (\Sigma_1 \parallel \Sigma_2)) \setminus (L_{\Sigma_1} \cup L_{\Sigma_2}) \sqsubseteq T
\]

implies

\[
(\mathcal{P} \parallel (U_1 \parallel U_2)) \setminus (L_{\Sigma_1} \cup L_{\Sigma_2}) \sqsubseteq T.
\]
Constraint 3: Coherence with the sequential specification

**Single threaded semantics:** A correct implementation should behave according to the sequential semantics if accessed at a single replica.

**Permutation equivalence:** “If all sequential permutations of updates lead to equivalent states, then it should also hold that concurrent executions of the updates lead to equivalent states.

**Client-server linearizability:** Any execution of a correct implementation on a client-server system should be linearizable.
An implementation $U$ is valid if it is simulated by the above automaton.
The linearizability automaton is too restrictive: does not simulate many desired behaviors.

Not *linearizable*: no way to place both $X_0, X_1$ in $+0 +1$ while preserving order.

What is the correct formalization?
Relaxing linearizability: Eventual consistency

But, states of all the replicas eventually converge when all the messages have been delivered. cf. \textit{quiescent consistency}

Suffices for “shopping cart”.

Radha Jagadeesan, James Riely, College of CDM, DePaul University, Chicago
From Linearizability to Eventual Consistency
Consistency at non-quiescent states??

Not enough constraints: eg. “permutation equivalence” not enforced.
Prior Work

Abandon sequential specifications
[Bouajjani, Enea, Hamza 2014]
[Burckhardt, Gotsman, Yang, Zawirski 2014]

😊 Only sequential specifications are canonical

Permutation based
[Burckhardt, Leijen, Fähndrich, Sagiv. 2012]
[Jagadeesan, Riely 2015]

😢 Too restrictive
Our approach: liberalize the linearizability automaton

Two ingredients.

(a) Quotient states under observational equivalence
(b) Time as a partial order
   Prefixes to subsequences
   Explicate and disentangle dependencies
Quotient states under observational equivalence

In linearizability state machine, states are sequences of methods.
[Brookes 96]: Two sequences are equivalent if they yield the same sequence of states of the data structure, upto stuttering. In set:

\[ +0+0 \sim +0 \quad +0\check{0} \sim +0 \]

and the equivalence classes for a set over one element 0 are:

\[ +0, +0-0, +0-0+0, +0-0+0-0, \ldots \]
Time as a partial order.

In the linearizability automaton, time is linear.
Strict prefix ordering

Radha Jagadeesan, James Riely, College of CDM, DePaul University, Chicago
From Linearizability to Eventual Consistency
Time as a partial order: prefixes to subsequence

Radha Jagadeesan, James Riely, College of CDM, DePaul University, Chicago
From Linearizability to Eventual Consistency
Time as a partial order. Disentangling dependencies

The linearizability automaton is insensitive to independence.

In binary set \([+0, -0, \checkmark 0, \times 0, \times 1, \checkmark 1]\), the two values are independent, i.e. a trace for a binary set is valid iff its projection to 0 (resp. 1) is valid.

More generally, enrich specification with notion of conflict: 

\[
\text{set} : +0 \# \checkmark 0, +0 \# \times 0, +1 \# \checkmark 1, +1 \# \times 1, \checkmark 0 \# \times 0, \checkmark 1 \# \times 1 \ldots
\]

Distributed text editors: Two labels from this alphabet are in conflict iff they mention overlapping sets of text identifiers, or if one is a query and the other is a remove.

\[
b < c \# a < b, ?c e \# -a \ldots
\]
Disentangling dependencies: set

Specification: \((+0-0+0) \parallel (+1-1+1)\)
Disentangling dependencies: Distributed text editor

Specification: !c; b<c; d>c; ?bcd; a<b; e>d; -b; -d; ?ace
Disentangling dependencies: the set automaton.
Our correctness criterion: \( U \sqsubseteq \Sigma \)

\( \Sigma \): Incorporate "quotienting of the label sequences under observational equivalence", and "time as a partial order".

is generated *purely* from the standard sequential specification.

An implementation \( U \) is valid if it is simulated by \( \Sigma \).
The set automaton
Alternative characterization. Via a direct definition.

Coherence with the sequential specification. Single threaded semantics, Permutation equivalence and Client-server linearizability.

Expressiveness. Addresses the CRDT examples.
Results (2)

**Composition.** Given two separate and independent sets, $L_{\Sigma_1} \cap L_{\Sigma_2} = \emptyset$ and $U_1 \subseteq \Sigma_1$, $U_2 \subseteq \Sigma_2$, we have:

$$U_1 \parallel U_2 \subseteq \Sigma_1 \parallel \Sigma_2$$

**Abstraction.** Let $\mathcal{P}$ be the graph implementation, which is a client of the two sets (for vertices, edges). Then:

$$(\mathcal{P} \parallel (\Sigma_1 \parallel \Sigma_2)) \setminus (L_{\Sigma_1} \cup L_{\Sigma_2}) \subseteq T$$

implies

$$(\mathcal{P} \parallel (U_1 \parallel U_2)) \setminus (L_{\Sigma_1} \cup L_{\Sigma_2}) \subseteq T.$$
Results (3). calm clients can program sequentially


\[ path(@Src, Dest) : - path(@Src, X), link(@X, Dest) \]

BUT: “non-monotonic reasoning in general requires global barriers”. eg. state change, counting aggregates...

\[ \text{toggle}(1) : - \text{state}(0) \]
\[ \text{toggle}(0) : - \text{state}(1) \]
\[ \text{state}(X)\text{@next}: - \text{toggle}(X) \]

No “races” between concurrent mutators and mutators/accessors.
QUESTIONS??

For full details refer to:
DePaul CDM Tech Report, 2016. “From Linearizability to Eventual Consistency”.