



hochschule mannheim

# Understanding Eventual Consistency

MSI Presentation SS2014

Horst Schneider, Patrick Beedgen

Hochschule Mannheim

June 17th, 2014

# Introduction

*" ...the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value"*  
–W. Vogels (2009)

# Introduction

## Interpretations of Eventual Consistency

### Interpretation 1

*"When you read data[...], the response might not reflect the results of a recently completed write operation. The response might include some stale data. Consistency across all copies of the data is usually reached within a second; so if you repeat your read request after a short time, the response returns the latest data."*

### Interpretation 2

*"This sort of system we term "single writer eventual consistency". So what are its properties?*

- (1) A client could read stale data.*
- (2) The client could see out-of-order write operations. [...] So this is our weakest form of consistency - eventually consistent with out of order reads in the short term."*

# Introduction

## Interpretations of Eventual Consistency

### DynamoDB Documentation

*"When you read data[...], the response might not reflect the results of a recently completed write operation. The response might include **some stale data**. Consistency across all copies of the data is **usually reached within a second**; so if you repeat your read request after a short time, the response returns the latest data."*

### MongoDB Documentation

*"This sort of system we term "single writer eventual consistency". So what are its properties?*

*(1)**A client could read stale data.***

*(2)**The client could see out-of-order write operations.[...]***

*So this is our weakest form of consistency - eventually consistent with **out of order reads** in the short term."*

# Problems

- disparate and low-level formalisms  
*consistency model is tied to system implementation*

# Problems

- disparate and low-level formalisms  
*consistency model is tied to system implementation*
- weak guarantees  
*in realistic scenarios updates **never** stop*

# Problems

- disparate and low-level formalisms  
*consistency model is tied to system implementation*
- weak guarantees  
*in realistic scenarios updates **never** stop*
- conflict resolution policies  
*resolution of conflicts in multiple replicas*

# Problems

- disparate and low-level formalisms  
*consistency model is tied to system implementation*
- weak guarantees  
*in realistic scenarios updates **never** stop*
- conflict resolution policies  
*resolution of conflicts in multiple replicas*
- combinations of different consistency levels  
*strong consistency may be needed at certain times*



## Problems

- disparate and low-level formalisms  
*consistency model is tied to system implementation*
- weak guarantees  
*in realistic scenarios updates **never** stop*
- conflict resolution policies  
*resolution of conflicts in multiple replicas*
- combinations of different consistency levels  
*strong consistency may be needed at certain times*

⇒ some sort of formalism is needed to define semantics of Eventual Consistency

# Agenda

- ① Replicated Data Types
- ② Axiomatic Specification Framework
- ③ Consistency Strengthening Interfaces
- ④ Conclusion / Discussion

# Replicated Data Types

- a replicated database stores **objects**  $\text{Obj} = \{x, y, \dots\}$

## Replicated Data Types

- a replicated database stores **objects**  $\text{Obj} = \{x, y, \dots\}$
- every object  $x \in \text{Obj}$  has
  - a **value**  $\in \text{Val}$
  - a **type**  $\text{type}(x) \leftrightarrow \tau$
  - **operations**  $\text{Op}_{\text{type}(x)}$  that a client can perform on it

## Replicated Data Types

- a replicated database stores **objects**  $\text{Obj} = \{x, y, \dots\}$
- every object  $x \in \text{Obj}$  has
  - a **value**  $\in \text{Val}$
  - a **type**  $\text{type}(x) \leftrightarrow \tau$
  - **operations**  $\text{Op}_{\text{type}(x)}$  that a client can perform on it
- two examples: Int Register **intreg**, Counter **ctr**

$$\begin{aligned}\text{Op}_{\text{ctr}} &= \{\text{rd}, \text{inc}\} \\ \text{Op}_{\text{intreg}} &= \{\text{rd}, \text{wr}(k) \mid k \in \mathbb{Z}\}\end{aligned}$$

# Replicated Data Types

## Sequential Data Type Specification

in a *strongly consistent system*, the semantics of a data type can be specified by a function:

$$S_\tau : \text{Op}_\tau^+ \rightarrow \text{Val}$$

# Replicated Data Types

## Sequential Data Type Specification

in a *strongly consistent system*, the semantics of a data type can be specified by a function:

$$S_{\tau} : \text{Op}_{\tau}^{+} \rightarrow \text{Val}$$

example:

$$S_{\text{intreg}}(\sigma \text{ wr}(k)) = S_{\text{ctr}}(\sigma \text{ inc}) = \perp;$$

(e.g.  $\sigma = \{\text{rd rd wr}(5) \text{ wr}(6) \text{ rd}\}$  or  $\sigma = \{\text{rd rd inc inc rd}\}$  )

# Replicated Data Types

## Sequential Data Type Specification

in a *strongly consistent system*, the semantics of a data type can be specified by a function:

$$S_{\tau} : \text{Op}_{\tau}^{+} \rightarrow \text{Val}$$

example:

$$\begin{aligned} S_{\text{intreg}}(\sigma \text{ wr}(k)) &= S_{\text{ctr}}(\sigma \text{ inc}) = \perp; \\ S_{\text{ctr}}(\sigma \text{ rd}) &= (\text{number of inc operations in } \sigma); \end{aligned}$$

(e.g.  $\sigma = \{\text{rd rd wr}(5) \text{ wr}(6) \text{ rd}\}$  or  $\sigma = \{\text{rd rd inc inc rd}\}$  )



# Replicated Data Types

## Sequential Data Type Specification

in a *strongly consistent system*, the semantics of a data type can be specified by a function:

$$S_\tau : \text{Op}_\tau^+ \rightarrow \text{Val}$$

example:

$$S_{\text{intreg}}(\sigma \text{ wr}(k)) = S_{\text{ctr}}(\sigma \text{ inc}) = \perp;$$

$$S_{\text{ctr}}(\sigma \text{ rd}) = (\text{number of inc operations in } \sigma);$$

$$S_{\text{intreg}}(\sigma \text{ rd}) = k; \text{ if } \text{wr}(0)\sigma = \sigma_1 \text{ wr}(k) \sigma_2 \text{ and } \\ \sigma_2 \text{ does not contain wr operations}$$

(e.g.  $\sigma = \{\text{rd rd wr}(5) \text{ wr}(6) \text{ rd}\}$  or  $\sigma = \{\text{rd rd inc inc rd}\}$  )

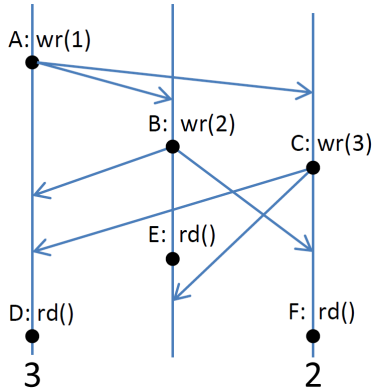
# Replicated Data Types

## Semantics of Eventual Consistency

- semantics of eventually consistent systems are harder to formalize
- concurrent operations on the same object happen on multiple replicas
- each replica executes operations immediately, updating other replicas later → **conflicts**
- different conflict resolution strategies for replicated data types

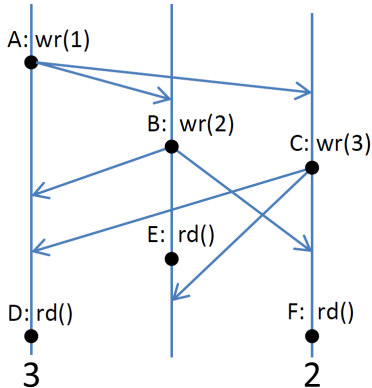
# Replicated Data Types

## Conflict Resolution Strategies



# Replicated Data Types

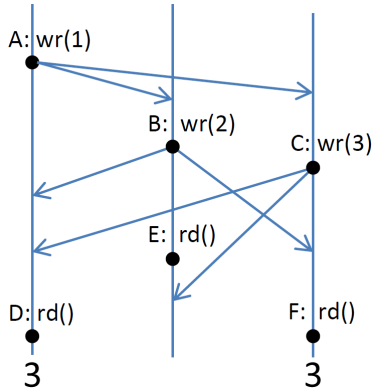
## Conflict Resolution Strategies



- ① make concurrent operations commutative

# Replicated Data Types

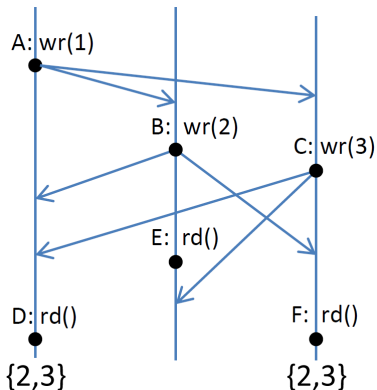
## Conflict Resolution Strategies



- ① make concurrent operations commutative
- ② order concurrent operations

# Replicated Data Types

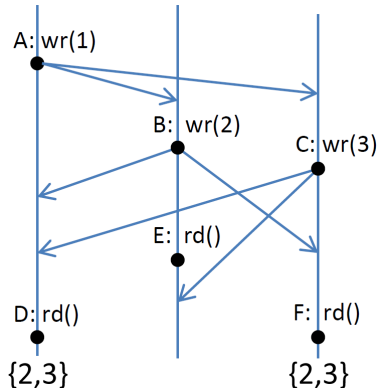
## Conflict Resolution Strategies



- ① make concurrent operations commutative
- ② order concurrent operations
- ③ flag conflicts (let the user decide)

# Replicated Data Types

## Conflict Resolution Strategies



- ① make concurrent operations commutative
- ② order concurrent operations
- ③ flag conflicts (let the user decide)
- ④ resolve conflicts semantically

# Replicated Data Types

## Replicated Data Type Specification

- $S_\tau$  is not strong enough to formalize these strategies
- **visibility** and **order** of preceding operations have to be included



# Replicated Data Types

## Replicated Data Type Specification

- $S_\tau$  is not strong enough to formalize these strategies
- **visibility** and **order** of preceding operations have to be included
- $F_\tau$ : takes an **operation context**  $C$  and returns a value

$$F_\tau(C) \in \text{Val}$$

# Replicated Data Types

## Replicated Data Type Specification

- $S_\tau$  is not strong enough to formalize these strategies
- **visibility** and **order** of preceding operations have to be included
- $F_\tau$ : takes an **operation context**  $C$  and returns a value

$$F_\tau(C) \in \text{Val}$$

- $C$  provides preceding operations with **visibility** and **arbitration relations**:

$$C = (f, V, \text{ar}, \text{vis})$$

# Replicated Data Types

## Replicated Data Type Specification

- $S_\tau$  is not strong enough to formalize these strategies
- **visibility** and **order** of preceding operations have to be included
- $F_\tau$ : takes an **operation context**  $C$  and returns a value

$$F_\tau(C) \in \text{Val}$$

- $C$  provides preceding operations with **visibility** and **arbitration relations**:

$$\begin{aligned} C &= (f, V, \text{ar}, \text{vis}) \\ u &\xrightarrow{\text{vis}} v, \text{vis} \subseteq V \times V \\ u &\xrightarrow{\text{ar}} v, \text{ar} \subseteq V \times V \end{aligned}$$

# Replicated Data Types

## Replicated Data Type Specification

example: Strategy **Make Concurrent Calls Commutative**

$$F_{\text{ctr}}(\text{inc}, V, \text{vis}, \text{ar}) = \perp;$$

$$F_{\text{ctr}}(\text{rd}, V, \text{vis}, \text{ar}) = (\text{the number of inc operations in } V);$$

# Replicated Data Types

## Replicated Data Type Specification

example: Strategy **Make Concurrent Calls Commutative**

$$F_{\text{ctr}}(\text{inc}, V, \text{vis}, \text{ar}) = \perp;$$

$$F_{\text{ctr}}(\text{rd}, V, \text{vis}, \text{ar}) = (\text{the number of inc operations in } V);$$

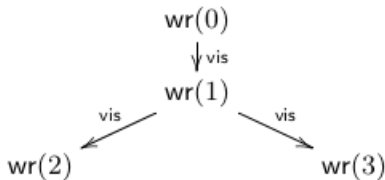
example: Strategy **Order Concurrent Operations**

$$F_{\text{intreg}}(f, V, \text{vis}, \text{ar}) = S_{\text{intreg}}(V^{\text{ar}} f) \\ (S_{\tau} : \text{Op}_{\tau}^{+} \rightarrow \text{Val})$$

# Replicated Data Types

## Replicated Data Type Specification

example: Strategy **Flag Conflicts**



writes on a multi value register

# Axiomatic Specification Framework

## Session and Action

- a single client may do several changes to the same object
- **sessions** provide a way to track client identity for operations
- an **action** is a tuple  $(e, s, [x.f : k])$ 
  - $e$ : unique identifier
  - $s$ : session id  $\in \text{SId}$
  - $[x.f : k]$ : object, operation and return value

# Axiomatic Specification Framework

## Session and Action

- a single client may do several changes to the same object
- **sessions** provide a way to track client identity for operations
- an **action** is a tuple  $(e, s, [x.f : k])$ 
  - $e$ : unique identifier
  - $s$ : session id  $\in \text{SId}$
  - $[x.f : k]$ : object, operation and return value

example:

$$a = (1af3c, 17, [x.rd : k]); \text{type}(x) = \text{intreg}$$



# Axiomatic Specification Framework

## History and Execution

- the set of all actions that happen in a database is denoted as  $\text{Act}$
- a **history**  $(A, \text{so})$  is a set of actions  $A \subseteq \text{Act}$  and a **session order** relation  $\text{so} \subseteq A \times A$
- an **execution**  $X = (A, \text{so}, \text{vis}, \text{ar})$  enhances the history with visibility and arbitration relations
- we can now extract an operation context for any action in any session, providing a deterministic return value

# Axiomatic Specification Framework

## Levels of Eventual Consistency

- with replicated data types we can define multiple forms of Eventual Consistency
  - Basic Eventual Consistency
  - Session Guarantees
  - Causality
- every form contains multiple axioms
- more axioms mean stronger consistency

# Axiomatic Specification Framework

## Basic Eventual Consistency Axioms

- axioms a database implementation has to enforce to offer  
**Basic Eventual Consistency**
- Well Formedness Axioms
  - SOwf: *so is the union of transitive, irreflexive and total orders on actions by each session*
  - VISwf:  $\forall a, b. a \xrightarrow{\text{vis}} b \Rightarrow \text{obj}(a) = \text{obj}(b)$
  - ARwf:  $\forall a, b. a \xrightarrow{\text{ar}} b \Rightarrow \text{obj}(a) = \text{obj}(b)$

# Axiomatic Specification Framework

## Basic Eventual Consistency Axioms

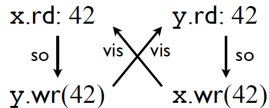
- Basic Eventual Consistency axioms:
  - THINAIR:  $so \cup vis$  is anticyclic

# Axiomatic Specification Framework

## Basic Eventual Consistency Axioms

- Basic Eventual Consistency axioms:
  - THINAIR:  $so \cup vis$  is anticyclic

**not** possible in THINAIR:



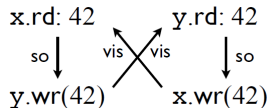
# Axiomatic Specification Framework

## Basic Eventual Consistency Axioms

- Basic Eventual Consistency axioms:

- THINAIR:  $\text{so} \cup \text{vis}$  is anticyclic

**not** possible in THINAIR:

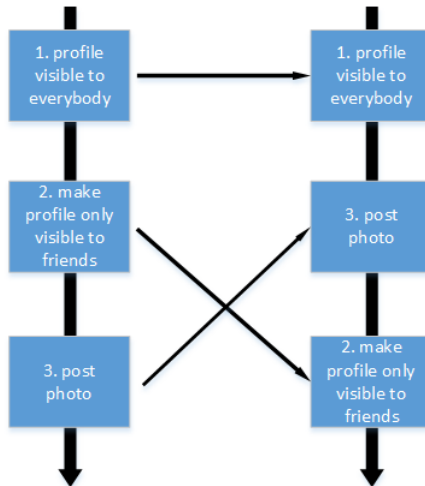


- EVENTUAL:

$$\forall a \in A. \neg(\exists \text{infinitely many } b \in A. \text{same}(a, b)) \wedge \neg(a \xrightarrow{\text{vis}} b)$$

# Axiomatic Specification Framework

Problem with basic eventual consistency



# Axiomatic Specification Framework

## Session guarantees

- with Basic Eventual Consistency we still might be reading values out of order
- axioms that formalise that all operation within a session keep the current context consistent:
  - Read Your Writes: *An operation sees all previous operations by the same session*
  - Writes Follow Reads in Visibility: *Arbitration orders an operation after other operations previously seen by the same session*
  - ... etc.



# Axiomatic Specification Framework

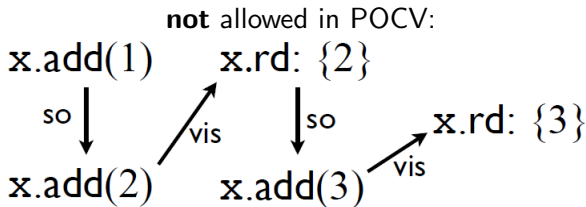
## Causality Axioms

- Per-object-causal-visibility: *POCV guarantees that an operation sees all operations on the same object that causally affect it*

# Axiomatic Specification Framework

## Causality Axioms

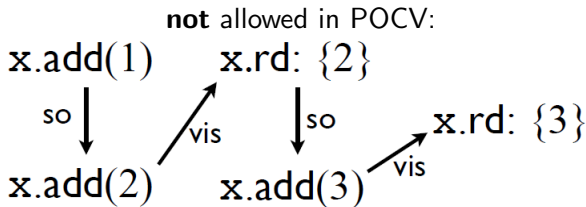
- Per-object-causal-visibility: *POCV guarantees that an operation sees all operations on the same object that causally affect it*



# Axiomatic Specification Framework

## Causality Axioms

- Per-object-causal-visibility: *POCV guarantees that an operation sees all operations on the same object that causally affect it*



- Per-object-causal-arbitration: *POCA correspondingly restricts the arbitration relation*

# Consistency Strengthening Interfaces

## Online Shopping

- almost everybody does online shopping
  - we put items in our shopping cart
  - we pay them
  - we continue shopping.. or not

# Consistency Strengthening Interfaces

## Online Shopping

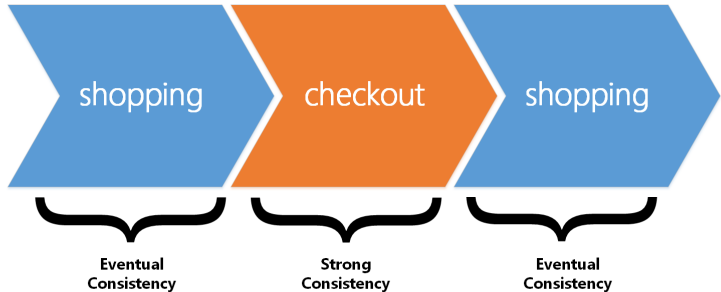
- almost everybody does online shopping
  - we put items in our shopping cart
  - we pay them
  - we continue shopping.. or not



# Consistency Strengthening Interfaces

## Online Shopping

- almost everybody does online shopping
  - we put items in our shopping cart
  - we pay them
  - we continue shopping.. or not



# Consistency Strengthening Interfaces

## Consistency Annotations

- every **action** accepted by the database has to be marked with a **consistency annotation**

$$(e, s, [x.f_\mu : k])_\mu \in \{ORD, CSL\}$$

- either **ordinary** or **causal**
- ordinary actions behave like we defined previously
- causal actions make all operations performed before the annotations visible to all previous actions

# Consistency Strengthening Interfaces

## Fences

- instead of annotating every single action, a **fence** can be used
- a **fence** is an **action** where the executing replica forces all its updates on every other replica in the cluster

$$\text{action } a = (e, s, \text{fence})$$

- the execution of other actions is halted until all replicas acknowledge the receipt



# Consistency Strengthening Interfaces

## Fences

- instead of annotating every single action, a **fence** can be used
- a **fence** is an **action** where the executing replica forces all its updates on every other replica in the cluster

$$\text{action } a = (e, s, \text{fence})$$

- the execution of other actions is halted until all replicas acknowledge the receipt
- the database behaves like a **CP** database !

# Conclusion

- the paper provides a formal way to **precisely specify eventually consistent systems**
- **every aspect of a system is covered**, from data types to client interaction
- specifications are **independent of implementation details**
- still **very theoretical**, no tools available to map between specifications and implementation
- the framework is **not suitable for programmers**, as it is very abstract and not easily understandable and applicable
- the paper is still "work in progress"

# Discussion