



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."*

## The Problem

- 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

## 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:

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

(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:

$$\begin{aligned} S_{\text{ctr}}(\sigma \text{rd}) &= (\text{number of inc operations in } \sigma); \\ S_{\text{intreg}}(\sigma \text{rd}) &= k; \text{ if } \text{wr}(0)\sigma = \sigma_1 \text{wr}(k)\sigma_2 \text{ and} \\ &\quad \sigma_2 \text{ does not contain wr operations} \end{aligned}$$

(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:

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

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

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

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

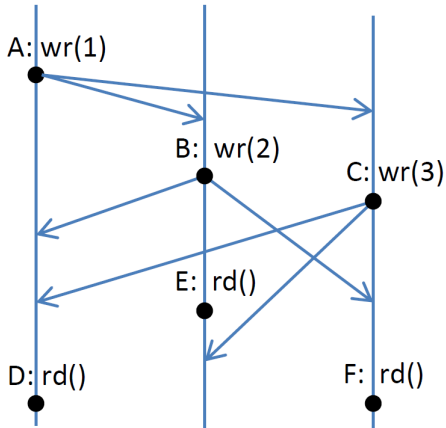
# Replicated Data Types

## 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
- different implementation strategies for replicated data types

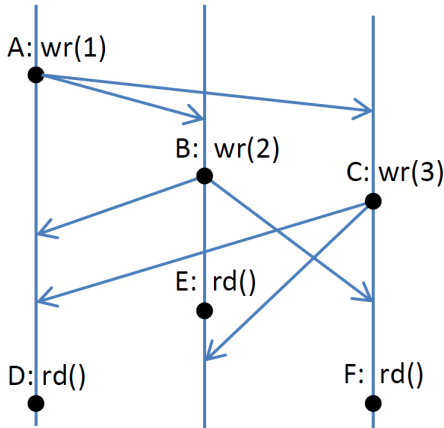
# Replicated Data Types

## Conflict Resolution Strategies



# Replicated Data Types

## Conflict Resolution Strategies

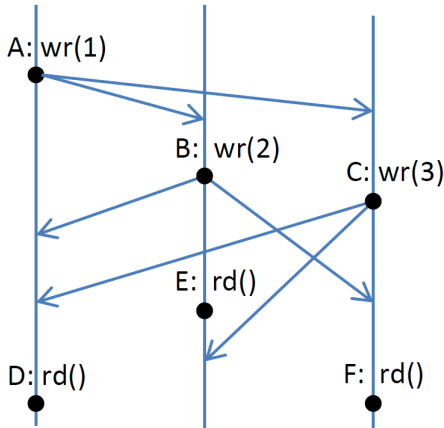


- 1 Make concurrent operations commutative



# Replicated Data Types

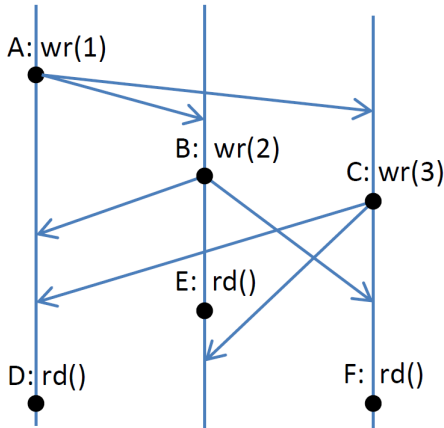
## Conflict Resolution Strategies



- ① Make concurrent operations commutative
- ② Order concurrent operations

# Replicated Data Types

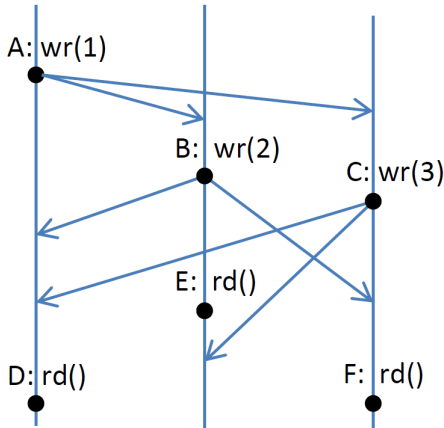
## Conflict Resolution Strategies



- 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** 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** and returns a **value**

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

- operation context  $C$  adds **visibility** and **arbitration relations** to preceding operations:

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

# 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** and returns a **value**

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

- operation context  $C$  adds **visibility** and **arbitration relations** to preceding operations:

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

# 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** and returns a **value**

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

- operation context  $C$  adds **visibility** and **arbitration relations** to preceding operations:

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



# 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**

$$F_{\text{integ}}(\text{inc}, V, \text{vis}, \text{ar}) = S_{\text{integ}}(V^{\text{ar}} f)$$

# Axiomatic Specification Framework

## Session and Action

- clients wish to perform operations in a common context
- **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

- clients wish to perform operations in a common context
- **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 these replicated data types we can define multiple forms of eventual consistency
  - Basic eventual consistency
  - Ordering guarantees
  - On-demand consistency strengthening
- Every form contains multiple axioms

# Axiomatic Specification Framework

## Basic Eventual Consistency Axioms

- Axioms a database implementation has to apply to offer basic eventual consistency
- Well-Formedness Axioms
  - SOwf, ARwf, VISwf
- Data Type Axiom
  - RVAL
- Basic Eventual Consistency axioms
  - EVENTUAL
  - THINAIR

# Axiomatic Specification Framework

## Session guarantees

- Axioms that ensure that databases stay consistent within a single session with a client
- !RYW!, MR, WYRV, WFRA, MWV, MWA



# Axiomatic Specification Framework

## Causal Consistency Axioms

- POCV, POCA, COCV, COCA

# Conclusion

- the paper provides a way to **precisely specify eventually consistent systems** in a common notation
- **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

# Discussion