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 valuee“
–W. Vogels (2009)

„Zweites Zitat über Ev. Consistency “

The Problem

- The definitions are ambiguous
- Most big players claim to implement it
- Implementations can't be be compared… scientifically
- In real world distributed databases updates never stop

1. Replicated Data Types

2. Axiomatic Specification Framework

Replicated Data Types

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

Replicated Data Types

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

Replicated Data Types

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

$$\mathrm{Op}_{\mathrm{ctr}} = \{\mathrm{rd}, \mathrm{inc}\}$$
$$\mathrm{Op}_{\mathrm{intreg}} = \{\mathrm{rd}, \mathrm{wr}(k) | k \in \mathbb{Z}\}$$

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

$$S_\tau \ : \mathrm{Op}_\tau^+ \to \mathrm{Val}$$

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

$$S_\tau \ : \mathrm{Op}_\tau^+ \to \mathrm{Val}$$

Example:

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

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

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

$$S_\tau \ : \mathrm{Op}_\tau^+ \to \mathrm{Val}$$

Example:

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

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

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

$$S_\tau \ : \mathrm{Op}_\tau^+ \to \mathrm{Val}$$

Example:

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

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

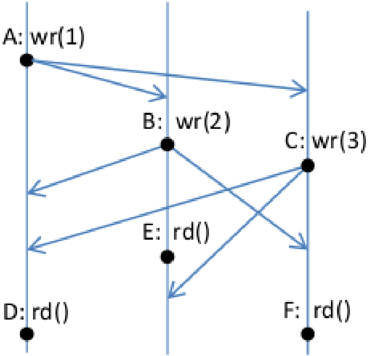
Replicated Data Types

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

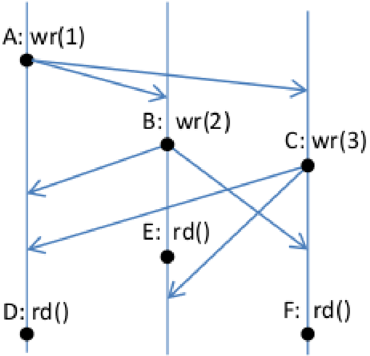
Replicated Data Types

Conflict Resolution Strategies

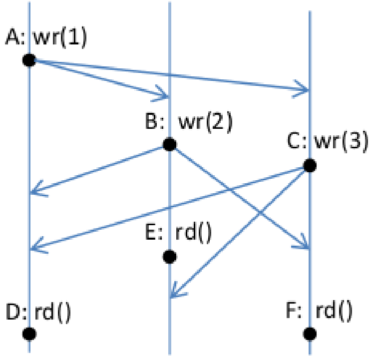


Replicated Data Types

Conflict Resolution Strategies



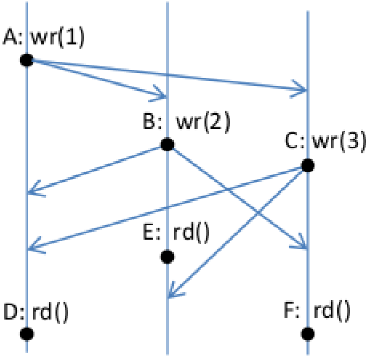
1. Make concurrent operations commutative



1. Make concurrent operations commutative
2. Order concurrent operations

Replicated Data Types

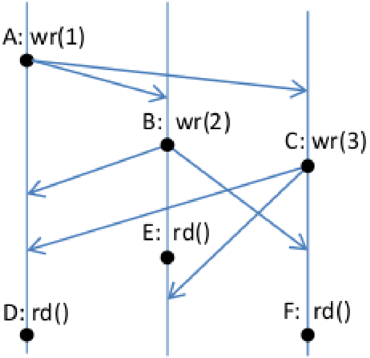
Conflict Resolution Strategies



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

- S_τ is not strong enough to formalize these strategies

- S_τ is not strong enough to formalize these strategies
- visibility and order of preceding operations have to be included

- S_τ is not strong enough to formalize these strategies
- visibility and order of preceding operations have to be included
- F_τ: takes an **operation context** and returns a **value**

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

- S_τ is not strong enough to formalize these strategies
- visibility and order of preceding operations have to be included
- F_τ: takes an **operation context** and returns a **value**

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

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

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

- S_τ is not strong enough to formalize these strategies
- visibility and order of preceding operations have to be included
- F_τ: takes an **operation context** and returns a **value**

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

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

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

- S_τ is not strong enough to formalize these strategies
- visibility and order of preceding operations have to be included
- F_τ: takes an **operation context** and returns a **value**

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

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

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

Example: Strategy **Make Concurrent Calls Commutative**

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

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

Example: Strategy **Make Concurrent Calls Commutative**

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

Example: Strategy **Order Concurrent Operations**

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

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 $\in \mathrm{SId}$
  - $[x.f : k]$: 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 $\in \mathrm{SId}$
  - $[x.f : k]$: object, operation and return value

Example:

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

Axiomatic Specification Framework

History and Execution

- the set of all actions that happen in a database is denoted as Act
- a **history** (A, so) is a set of actions $A \subseteq \mathrm{Act}$ and a **session order** relation $\mathrm{so} \subseteq A \times A$
- an **execution** $X = (A, \mathrm{so}, \mathrm{vis}, \mathrm{ar})$ 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
  - Ordering guarantees
  - on-demand consistency strengthening
- Every form contains multiple axioms

Axiomatic Specification Framework

Client Interaction Model

- Clients often wish to perform multiple operations within some context
- bla

Axiomatic Specification Framework

Basic Eventual Consistency Axioms

- Axioms a database has to fulfill to be eventual consistent
- SOWF, ARWF, VISWF, RVAL, 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

- 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