A comprehensive study of Convergent and Commutative Replicated Data Types

Marc Shapiro, Nuno Preguica, Carlos Baquero, Marek Zawirski

[Research Report] RR-7506, Inria, 2011

2023/10/18　Delight B3 kekeho

概要

CRDT

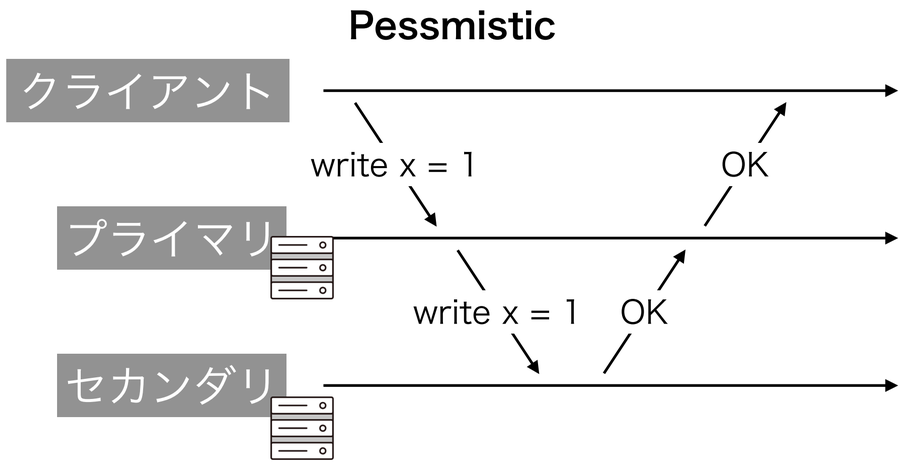
- 全てのレプリカ間で、一貫した最終状態に常に収束するデータ型として、2種類のConflict-free Replicated Data Typeを提案
  - Convergent Replicated Data Type (**State-Based**)
  - Commutative Replication Data Type (**Operation-Based**)
- カウンタ、Set、グラフなどいくつかのCRDT型を紹介

CRDTの概要

背景

分散システムにおけるReplication

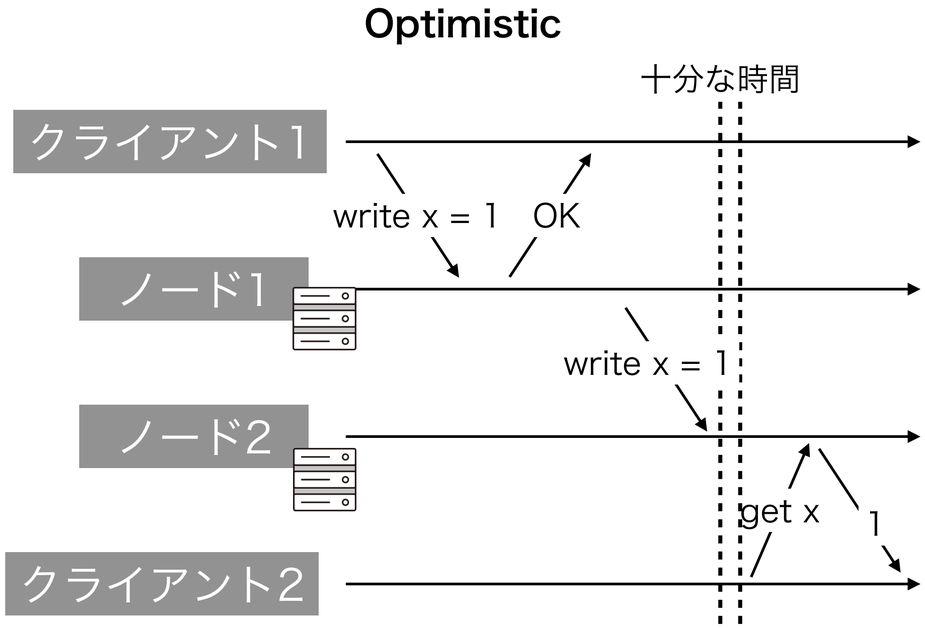
- 分散システムでは、Availabilityを高くするために**Replication**を行う
  - Replication: データのコピーを複数のプロセス(ノード)で保持する
- Replication手法は、Pessimistic/Optimisticの二種類に大別される
- **Pessimistic(悲観的) Replication**
  - プライマリは、データの書き込みを行ったあと同期的にセカンダリに変更を書き込む
  - ノード間のデータで一貫性が保てる一方、広域分散システムにおいて優れたパフォーマンス・Availabilityを出すことが難しい



背景

分散システムにおけるReplication

- **Optimistic(楽観的) Replication**
  - ノードは、書き込みを受け付けたあと、非同期的に他のノードに変更を送信する。
  - Eventual Consistency (結果整合性)が許容できるアプリケーションで有用
    - **Eventual Consistency**: 十分な時間が経過した後、更新がシステムを通じて伝播し、最終的にすべてのプロセス(ノード)に適用されることを保証するモデル
  - Partition耐性があり、高いAvailabilityを維持できる。パフォーマンスがよい。
  - データの衝突が起きたら、その都度修正することになる。このメカニズムは一般的に複雑である。



CRDTの概要

- 本論文では、**Conflict-free Replicated Data Type: CRDT**を提案
- CRDTのレプリカは、正しく逐次実行された場合と等しくなるように収束する
- **Strong Eventual Consistency**を満たす
  - 全てのプロセス(ノード)が同じ更新を受け取ったときに、その順序に関係なく、同じ状態になることを保証するモデル
  - 別途コンセンサスを取らなくてOK
- 高いAvailabilityとPartition耐性を持つ

データモデル

- **Atom**
  - immutableなデータ型
  - プロセス間でコピーが可能
  - 整数・文字列・Set・Tupleなど
- **Object**
  - mutableなデータ型
  - アイデンティティ(識別子を指す?)・初期状態・操作(Operations)を持ち、コンテンツ(Payloadと呼ばれる)として任意の数のAtom, Objectを内包している

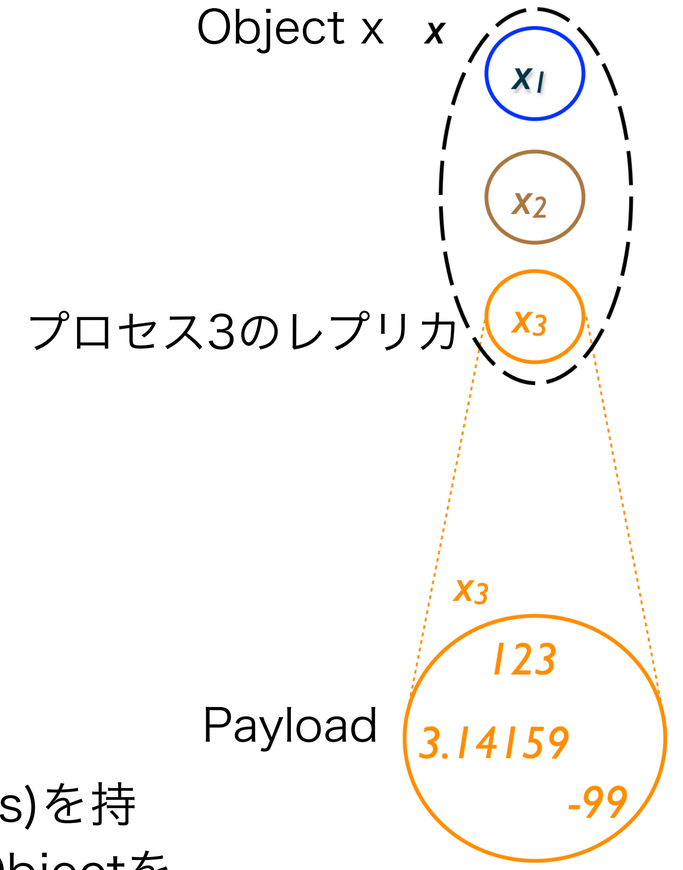


Figure 1: Object

Operations

- **Operation**: Objectに対する更新操作
- クライアントは、1つのノード上のレプリカにOperationを適用し、レプリカは更新を他のレプリカに非同期的に伝播させる
  - 更新の伝え方によって、2種類のCRDTがある
    - **Operation-based CRDT**
    - **State-based CRDT**

Operation-based CRDT

- Operation自体を他のノードに伝播させていく方式
- Commutative Replicated Data Type(CmRDT)とも呼ばれる
- Operationが可換(Commutative)であることが条件
  - g(f(x)) == f(g(x))
  - これによって、Strong Eventual Consistencyを満たすことができる

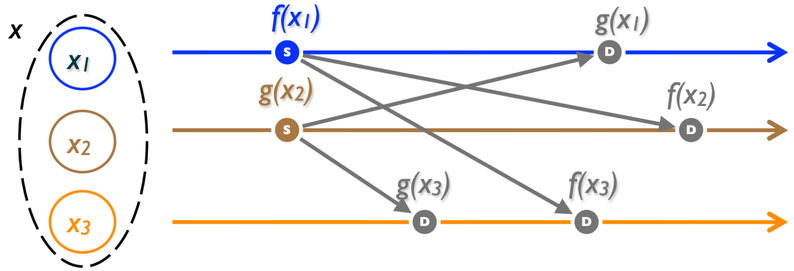


Figure 6: Operation-Based Replication

State-based CRDT

- Operation適用後のコンテンツ(State)を伝播させていく方式
- Convergent Replicated Data Type(CvRDT)とも呼ばれる
- 各レプリカは、受け取ったState自体を自身のStateにマージ(Join)する
- データがJoin Semilattice構造になっている必要がある。Join Semilatticeは以下の3つの性質を持つ。(∨はJoin)
  - 冪等性: $a \vee a = a$
  - 結合性: $a \vee (b \vee c) = (a \vee b) \vee c$
  - 可換性: $a \vee b = b \vee a$
  - これによって、Strong Eventual Consistencyを満たすことができる

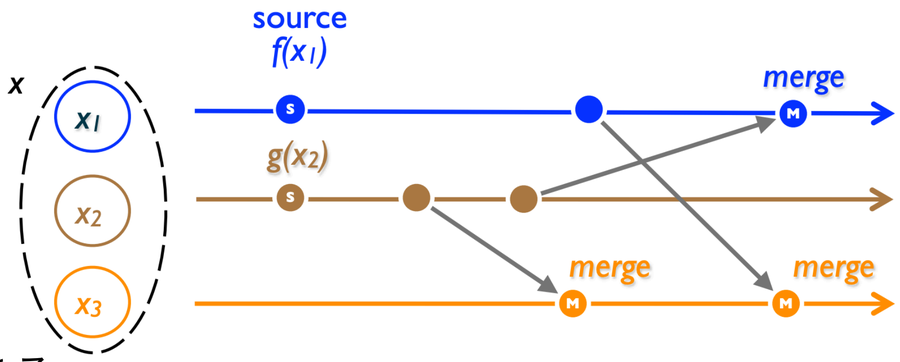


Figure 4: State-based replication

Portfolio

いろいろなCRDT

- いろいろなCRDT Objectがある。本スライドではその一部を紹介する。
  - Counter
  - Set
  - Register
  - その他
    - Graphなど

Counter

Operation-based Counter

- 各ノードが持つレプリカの構造: カウンタの値(初期値は0)
- Operation: Increment/Decrement
- Increment・Decrementは可換であるので、書くレプリカは受け取った順にOperationを適用するだけで、最終的な状態が収束する

Specification 5 op-based Counter

```
1:  payload integer i
2:      initial 0
3:  query value () : integer j
4:      let j = i
5:  update increment ()
6:      downstream ()
7:          i := i + 1
8:  update decrement ()
9:      downstream ()
10:         i := i − 1
```

Counter

State-based Counter

- **G-Counter** (Grow only counter)
  - インクリメントだけができるカウンタ
  - 各ノードが持つレプリカの構造: {“node名”, 値}の配列
  - マージ処理: 各ノード名毎に最大値を取る
    - 例: [{“Node1”: 5, “Node2”: 2}] ∨ [{“Node1”: 4, “Node2”: 4}] = [{“Node1”: 5, “Node2”: 4}]
    - Join Semilatticeである
  - それぞれのノードの値を合計したものがカウンタの値
    - 例: [{“Node1”: 5, “Node2”: 4}] => 9

Counter

State-based Counter

- **PN-Counter**
  - インクリメントとデクリメント両方できるカウンタ
  - G-Counterを拡張して、足し算分(Positive)と引き算分(Negative)をそれぞれ保持する
  - 各ノードが持つレプリカの構造: {'node名', (Positive値: P, Negative値: N)}
  - マージ処理: 各ノード毎に、PとNそれぞれ最大値を取る
    - 例: [{“Node1”: (5, 2), “Node2”: (2, 3)}] ∨ [{“Node1”: (6, 4), “Node2”: (3, 2)}] = [{“Node1”: (6, 4), “Node2”: (3, 3)}]
  - 各ノードのP・Nそれぞれの合計値を計算する。その後、Pの合計値からNの合計値を引いたものがカウンタの値
    - 例: [{“Node1”: (6, 4), “Node2”: (3, 3)}] => 9 - 7 = 2

Set

State-based Set

- **G-Set**
  - 追加しかできないSet
  - レプリカの構造: 集合自身(初期値=空集合)
  - マージ処理: 和集合を取る
    - 例: {1, 2, 3} ∨ {1, 3, 4} = {1, 2, 3, 4}
    - Join Semilatticeである

Set

State-based Set

- **2P-Set**
  - 追加・削除ができるSet
  - 各ノードが持つレプリカの構造: (追加された要素の集合: A, 削除された要素の集合: R)
    - PN-Counterのように、追加された要素の集合と削除された要素の集合をそれぞれ保持することで実現
  - マージ処理: A同士・R同士それぞれで和集合を取る
    - 例: ({1, 2, 3}, {1}) ∨ ({2, 3, 4}, {2}) = ({1, 2, 3, 4}, {1, 2})
  - A - RがSetの状態
    - 例: ({1, 2, 3, 4}, {1, 2}) => {3, 4}
  - 注意: 一度削除されてしまった値は、二度と追加できない

Set

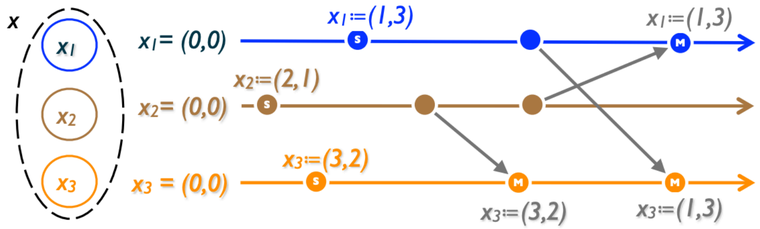
Operation-based Set

- **Op-based 2P Set**
  - レプリカの構造: (追加された要素の集合: A、削除された要素の集合: R)
  - Operation: Add(要素) / Remove(要素)
    - Operationを受け取った順にレプリカに適用する。
      - Add(要素): Aに要素を追加
      - Remove(要素): Rに要素を追加
  - A - RがSetの状態
    - 例: ({1, 2, 3, 4}, {1, 2}) => {3, 4}
  - 注意: 一度削除されてしまった値は、二度と追加できない

Register

State-based LWW-Register

- Register: Mutableな変数
- **Last Writer Wins (LLW) Register**
  - 最後に書き込まれた値が反映される
  - レプリカの構造: (レジスタの値: v、タイムスタンプ: t)
  - マージ処理: タイムスタンプが新しい方を反映する
    - 例: (“old”, 1) ∨ (“new”, 2) = (“new”, 2)
      - Join Semilatticeである



まとめ

CRDTおさらい

- **CRDT**: 高いAvailabilityとPartition耐性を持ち、Strong Eventual Consistency を持つデータ構造
- Operation-basedなものとState-basedなものがある
  - Operation-based: Operationが可換である必要がある
  - State-based: StateがJoin Semittrance構造を持つ必要がある
    - Join Semittrance: 冪等性・結合性・可換性
- Counter、Set、Register、Graphなど、様々なデータを表現できる

活用例

分散KVS

ライブラリ

アプリケーション

Room.sh

(Figma的なアプリ)

参考文献

- Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. [Research Report] RR-7506, Inria – Centre ParisRocquencourt; INRIA. 2011, pp.50. ffinria-00555588 https://inria.hal.science/inria-00555588/document
- Diving into Conflict-Free Replicated Data Types (CRDTs) - Redis https://redis.com/blog/diving-into-crdts/
- Eventual Consistency vs. Strong Eventual Consistency vs. Strong Consistency - Baeldung CS https://www.baeldung.com/cs/eventual-consistency-vs-strong-eventual-consistency-vs-strong-consistency