DEIM2022 J34-1

Coordination Avoidance のためのセグメント分解手法

葛木　優太† 　中園　翔†† 　鬼塚　真†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1–5
†† NTT コンピュータ&データサイエンス研究所　〒 180–8585 東京都武蔵野市緑町 3–9–11
E-mail: †{katsuragi.yuta,onizuka}@ist.osaka-u.ac.jp, ††syou.nakazono.nu@hco.ntt.co.jp

あらまし　データベースでのトランザクション処理において，対象のオブジェクトが Invariant confluent であることは，調停フリーな実行でアプリケーション固有の不変条件を保つための必要十分条件である．Invariant confluent ではない場合であっても，オブジェクトを 1 つ以上の Invariant confluent なセグメントに分解することで，セグメント内では調停フリーな実行が可能になる．しかし，Invariant confluent ではないオブジェクトを Invariant confluent なセグメントに分解するためのアルゴリズムは存在しない．本稿では，Invariant confluent ではないオブジェクトを Invariant confluent となることが保証されているセグメントに分解する手法を提案する．提案手法は (1) Invariant confluent の十分条件を満たすように不変条件を制限してセグメントを作成する技術と，(2) その存在により Invariant confluent にならないことが保証されるトランザクションをすべて制限してセグメントを作成する技術を用いて，セグメント分解を求める．提案手法により，Invariant confluent ではないオブジェクトを Invariant confluent なセグメントに分解が可能であることを複数の具体例を用いて確認した．

キーワード　トランザクション処理，Coordination Avoidance

1　はじめに

データベースのトランザクション処理において，高可用性，低遅延，高いスケーラビリティを達成するためには調停を削減することが重要である．しかし，調停フリーなトランザクション実行では，アプリケーション固有の不変条件 (Invariant) を守ることができない場合がある．アプリケーション固有の不変条件を保つためのシステムが提案されている [1–7]．そこでは，調停フリーな実行が可能か否か，調停が必要になる場合はいつかという問題を扱う．

Coordination Avoidance [6,7] ではオブジェクトが Invariant confluent であることが，調停フリーな実行でアプリケーション固有の不変条件を保つための必要十分条件であることが示されている．オブジェクトが Invariant confluent ではない場合は，トランザクションの実行において調停が必要となる．そのような場合において，調停フリーなトランザクション実行を可能にするための概念として，セグメントが提案されている [7]．セグメントはオブジェクトに対して実行可能なトランザクション，不変条件を制限した対象を表す．オブジェクトが Invariant confluent となるセグメントに分解することで，セグメント内では調停フリーな実行が可能になる．セグメントの移動や，セグメントに属さないトランザクションの実行では調停を行う．しかし，既存の研究 [7] では Invariant confluent ではないオブジェクトに対して，Invariant confluent なセグメントから構成されるセグメント分解を求めるためのアルゴリズムは示されておらず，そのようなセグメント分解を求めることはアプリケーション設計者の手に委ねられていた．

そこで本稿では Invariant confluent ではないオブジェクトに対して，Invariant confluent なセグメントから構成されるセグメント分解を求めるためのアルゴリズムを提案する．提案手法は，不変条件を制限してセグメントを作成する手法と，実行可能なトランザクション集合を制限してセグメントを作成する手法からなり，それぞれにより作成されたセグメントを用いてセグメント分解を作成する．1 つめの不変条件を制限する手法では，与えられた不変条件を，Invariant confluent の十分条件である Invariant closed を満たす部分不変条件に制限する．この際，Invariant confluent ではない場合には異なるセグメントの部分不変条件に所属すべき状態が存在することを活用する．2 つめのトランザクション集合を制限する手法では，Invariant confluent の反例に到達させうるトランザクションをトランザクション集合から除去することで Invariant confluent となることが期待されるセグメントを作成する．提案手法を複数の具体例に適用し，Invariant confluent なセグメントからなるセグメント分解が求められることを確認した．

本稿の構成は次のとおりである．まず，2 章で事前知識について説明する．3 章でセグメント分解を求めるための提案手法の詳細について述べ，4 章で提案手法を具体例に適用した結果を示す．5 章で関連研究についてまとめ，6 章で本稿をまとめる．

2　事前準備

本章では，[7] に基づき，Invariant confluent，Invariant closed，セグメント，Segmented invariant confluent の定義とシステムモデルについて述べる．

オブジェクトを $O = (S, \sqcup)$，トランザクション集合を T，ア

プリケーション固有の不変条件を I ($\subset S$)[1] とする．ここで，S は状態集合を表し，$\sqcup : S \times S \to S$ は 2 つの状態同士をマージする関数を表す．本稿では，マージ $\sqcup$ は冪等かつ可換であるとする．$s_0 \in S$ を O の初期状態とする．本稿では，マージによってのみ不変条件を満たさない状態へ遷移することを想定する [7].

2.1 Invariant confluent [7]

Invariant confluent の定義について述べるためにまず $\text{reachable}_{(s_0,T,I)}$ [7] の定義を示す．$\text{reachable}_{(s_0,T,I)}$ は初期状態 s_0，トランザクション集合 T，不変条件 I に依存する．$\text{reachable}_{(s_0,T,I)}$ は状態集合上に定められる述語であり，トランザクションやマージの実行により，初期状態から到達可能な状態を表現する．

定義 1 ($\text{reachable}_{(s_0,T,I)}$)**.** $\text{reachable}_{(s_0,T,I)}$ は次の条件を満たす最小の述語として再帰的に定義される．ある状態 s について，$\text{reachable}_{(s_0,T,I)}(s)$ が真であることを単に $\text{reachable}_{(s_0,T,I)}(s)$ と表記する．

- $\text{reachable}_{(s_0,T,I)}(s_0)$ である．
- 任意の状態 $s \in S$ と，トランザクション $t \in T$ について，$\text{reachable}_{(s_0,T,I)}(s)$ であり，$t(s) \in I$ が成り立つとき，$\text{reachable}_{(s_0,T,I)}(t(s))$ である．
- $\text{reachable}_{(s_0,T,I)}(s_1)$ かつ $\text{reachable}_{(s_0,T,I)}(s_2)$ であれば，$\text{reachable}_{(s_0,T,I)}(s_1 \sqcup s_2)$ である．

以下，$\text{reachable}_{(s_0,T,I)}$ を満たす状態を単に到達可能な状態とも表現する．

次に，Invariant confluent の定義について述べる．オブジェクトが Invariant confluent であることは，そのオブジェクトに対するトランザクションを調停フリーで実行して不変条件を保つための必要十分条件である [6]．Invariant confluent は 到達可能性を用いて定義され，s_0, T, I に依存する．s_0, T, I に関して Invariant confluent であることを (s_0, T, I)-confluent と表記する．

定義 2 ((s_0, T, I)-confluent)**.** オブジェクト O が (s_0, T, I)-confluent であるとは，

$$\{\, s \mid s \in S, \text{reachable}_{(s_0,T,I)}(s) \,\} \subset I \qquad (1)$$

を満たすことをいう．

調停フリーなトランザクション実行のためには Invariant confluent は望ましい性質であるが，与えられたオブジェクトが Invariant confluent であるかを決定することは一般に決定不能であることが知られている [7]．Invariant confluent の十分条件で，検査が容易な条件に Invariant closed がある．次に Invariant closed の定義を示し，Invariant closed が Invariant confluent の十分条件であることについて述べる．Invariant closed は不変条件 I に依存しており，I に関して Invariant closed であることを I-closed と表記する．

定義 3 (I-closed)**.** オブジェクト O が I-closed であるとは，任意の $s_1, s_2 \in I$ に対して，$s_1 \sqcup s_2 \in I$ を満たすことをいう．

以下，文脈から，対象のオブジェクトを省略しても誤解が生じない場合には，不変条件が Invariant closed とも書く．

Invariant closed は不変条件を満たす状態同士のマージが常に不変条件を保つことを表す．そのため，Invariant closed であれば到達可能な状態は常に不変条件を満たすことが保証され，Invariant confluent となる (定理 1).

定理 1. オブジェクトが $s_0 \in I$ かつ I-closed を満たせば，(s_0, T, I)-confluent である．

オブジェクトが Invariant closed であるかどうかは，不変条件とマージによってのみ決まり，初期状態，トランザクション集合には依存しない．オブジェクトが Invariant closed であるかは Z3 [8] などの SMT ソルバを用いることで検査が可能である．

Invariant closed が Invariant confluent の十分条件である理由は，Invariant closed がトランザクションとマージによる到達可能性を考えていないことによる．次の例 1 に Invariant closed ではないが，Invariant confluent となる例を示す．

例 1 (Invariant closed の十分性)**.** $O = (\mathbf{Z} \times \mathbf{Z}, \sqcup)$, $s_0 = (0, 0)$, $T = \{\, t_{x+1}, t_{y-1} \,\}$, $I = \{\, (x, y) \mid xy \leqq 0 \,\}$ を考える．ただし，ここでマージ $\sqcup$ は要素ごとの max であるとする．O は I-closed ではない．$s_1 = (-1, 1)$, $s_2 = (1, -1) \in I$ は，$s_1 \sqcup s_2 = (1, 1) \notin I$ を満たし，反例となる．しかし，反例 s_1 は到達可能ではない．O が実際は (s_0, T, I)-confluent であることが次のことからわかる．O の到達可能な状態全体の集合は $\{\, s \mid s \in S, \text{reachable}_{(s_0,T,I)}(s) \,\} = \{\, (x, y) \mid x \geqq 0, y \leqq 0 \,\}$ であり，右辺は不変条件の部分集合となる．よって，O は (s_0, T, I)-confluent となる．

次の例 2 に，Invariant closed ではなく，Invariant confluent にもならない例を示す．この例では到達可能な状態が Invariant closed の反例となり，Invariant confluent とはならない．また，この例は単に Invariant confluent ではないオブジェクトの例にもなっている．

例 2 (Invariant closed ではなく，Invariant confluent でもない例)**.** 例 1 と同じトランザクション集合，不変条件のオブジェクトを考える．ただし，今回の場合は初期状態を $s_0 = (-42, 42)$ であるとする．O はやはり I-closed ではないが，その反例となる状態 $s_1 = (3, -2)$, $s_2 = (-2, 7)$ はともに到達可能である．このことから，到達可能で不変条件を満たさない状態 $(s_1 \sqcup s_2 = (3, 7) \notin I)$ が存在することがわかり，O が (s_0, T, I)-confluent ではないことがわかる．

2.2 セグメントと Segmented invariant confluent [7]

オブジェクトが Invariant confluent ではない場合，調停フ

1：本稿では $A \subset B$ は $A = B$ の場合を含むものとし，$A \subset B$ かつ $A \neq B$ であることを $A \subsetneqq B$ で表す．

リーなトランザクション実行の下で不変条件を守ることはできない．しかし，オブジェクトが Invariant confluent ではない場合であっても，トランザクション集合や不変条件の制限により到達可能性を制限すると Invariant confluent になり得る．そこで，実行可能なトランザクション，不変条件を制限した対象であるセグメントを用いて，オブジェクトのセグメントによる分解を考える．オブジェクトが Invariant confluent となるセグメントに分割することで，各セグメント内では調停フリーなトランザクションの実行が可能になる．セグメント間の遷移やセグメントに属さないトランザクションの実行については，調停を行う．以下ではまずセグメントとセグメント分解の定義を示した後，Segmented invariant confluent の定義，Segmented invariant confluent となるオブジェクトの具体例を示す．

定義 4 (セグメント)．不変条件の部分集合 I' $(\subset I)$ とトランザクション集合の部分集合 T' $(\subset T)$ とのペア (I', T') をセグメントと呼ぶ．

ここで，セグメントの定義には，オブジェクトがセグメント内で Invariant confluent となることは含まれていないことに注意する．I', T' を適切に選択すればセグメント内で Invariant confluent になり得る．次に，セグメント分解の定義を示す．

定義 5 (セグメント分解)．(I_i, T_i) $(i = 1, \ldots, n)$ がセグメントであるとき，セグメントの列 $\Sigma = ((I_i, T_i))_{i \in \{1,\ldots,n\}} = (I_1, T_i), \cdots, (I_n, T_n)$ をセグメント分解という．

続いて，Segmented invariant confluent の定義を示す．Segmented invariant confluent は大まかには，セグメント分解を構成する各セグメント内でオブジェクトが Invariant confluent になることを表しており，Invariant confluent ではないオブジェクトについても，調停の頻度を削減できる可能性があることを示している．Segmented invariant confluent は s_0, T, I に加えて Σ にも依存する．s_0, T, I, Σ に関して Segmented invariant confluent となることを (s_0, T, I, Σ)-confluent と表記する．

定義 6 ((s_0, T, I, Σ)-confluent)．オブジェクトを $O = (S, \sqcup)$, 初期状態を s_0，不変条件を I，トランザクション集合を T，セグメント分解を $\Sigma = (I_1, T_1), \ldots, (I_n, T_n)$ とする．O が s_0, I, T, Σ に関して Segmented invariant confluent であるとは次の 3 つの条件を満たすことをいう．

(S1) 初期状態 s_0 が不変条件を満たす．
(S2) セグメント分解 Σ を構成するすべてのセグメントの部分不変条件の和が I に等しい $(I = \bigcup_{i=1,\ldots,n} I_i)$．
(S3) 各セグメントにおいて，O は任意の初期状態に関して Invariant confluent である．つまり，すべての $(I_i, T_i) \in \Sigma$，すべての $s \in I_i$ に対して O が (s, T_i, I_i)-confluent を満たす．

本稿では，定義 6 の条件 (S1) はいかなるオブジェクトについても満たされていることを想定する．ここで，初期状態について，(s_0, T, I)-confluent の場合は，特定の初期状態 s_0 に関してのみ Invariant confluent であるかを考えていた．しかし，(s_0, T, I, Σ)-confluent の場合は各セグメント (I_i, T_i) で，任意の状態 $s \in I_i$ に関して Invariant confluent であるかを考えていることに注意する．これは，初期状態が必ず s_0 であることがわかっている通常の Invariant confluent の場合とは異なり，Segmented invariant confluent の場合ではセグメント間の遷移が起こるため，I_i においてどの状態が初期状態となるかが明らかには定まらないことによっていると考えられる．より具体的には，Segmented invariant confluent の場合，あるセグメント I_i から別のセグメント I_j へ遷移したときの状態が I_j での 初期状態に相当するが，この初期状態は一般に任意の状態を取り得る．よって，セグメント内の任意の状態 s を初期状態とした場合について (s, T_i, I_i)-confluent が成り立たなくてはいけない．

最後に，Segmented invariant confluent となるセグメント分解の例を示す．

例 3 (Segmented invariant confluent となる例)．例 2 でのオブジェクト，トランザクション集合，不変条件，初期状態を考える．このとき，O は到達可能な状態同士のマージにより不変条件を保たない場合があるため，(s_0, T, I)-confluent ではなかった．このオブジェクトに対するセグメント分解の例を以下に示した後，そのセグメント分解に関して O が Segmented invariant confluent となることを確認する．

セグメント分解 $\Sigma = (I_1, T_1), \ldots, (I_4, T_4)$ は次の通りであるとする．

- $(I_1, T_1) = (\{\,(x, y) \mid x < 0,\, y > 0\,\}, \{\,t_{x+1}, t_{y-1}\,\})$
- $(I_2, T_2) = (\{\,(x, y) \mid x \geqq 0,\, y \leqq 0\,\}, \{\,t_{x+1}, t_{y-1}\,\})$
- $(I_3, T_3) = (\{\,(x, y) \mid x = 0\}, \{\,t_{y-1}\,\})$
- $(I_4, T_4) = (\{\,(x, y) \mid y = 0\}, \{\,t_{x+1}\,\})$

O が (s_0, T, I, Σ)-confluent となることを確認する．Σ を構成するすべてのセグメントで Invariant closed となることがわかる．例えば，セグメント (I_1, T_1) に注目すると，I_1 に属するどんな状態同士のマージも I_1 に閉じている．同様の事実が I_2, I_3, I_4 についても成り立つ．よって，定理 1 から各セグメントでは任意の初期状態 $s \in I_i$ に対して，(s, T, I)-confluent であることがわかる．さらに，不変条件 I と $I_1, \ldots, I_4$ について，$I = \bigcup_{i=1,\ldots,4} I_i$ が成り立つ．以上から，O は (s_0, T, I, Σ)-confluent と判定できる．

2.3 システムモデル [7]

想定するシステムモデルについて述べる．このシステムモデルは [7] でのシステムモデルと同様である．各サーバでは更新可能な，オブジェクトのレプリカを管理する．各サーバでは，ユーザが発行したトランザクションをレプリカに対して実行するほか，定期的に，レプリカの状態を別のサーバへ送信し，別のサーバから送られたレプリカの状態を受け取りマージを行う．

セグメントが存在する下でのトランザクション実行時のふるまいを Algorithm 1 に示す．調停が必要になるのは，不変条件

Algorithm 1 実行時の動作

```
1: if t ∈ T_i then
2:     if t(s) ∈ I_i then commit(t)
3:     else if t(s) ∉ I then abort(t)
4:     else global coordination                ▷ another segment
5: else global coordination                    ▷ t is not safe
```

を満たすが現在のセグメントの部分不変条件 I_i を満たさない場合 (4 行目) か，実行するトランザクションが現在のセグメントのトランザクション集合 T_i に属さない場合である (5 行目)．それ以外の場合の場合は調停フリーな実行が可能で，実行後の状態が現在のセグメントの部分不変条件 I_i を満たす場合はトランザクションをコミット可能 (2 行目)，実行後の状態が不変条件 I を満たさない場合はトランザクションをアボートする (3 行目)．

3 提案手法

2 章では調停フリーなトランザクションの実行とセグメントについて説明したが，従来の研究 [7] ではオブジェクトが Segmented invariant confluent となるセグメント分解を求めるためのアルゴリズムは示されておらず，セグメント分解はアプリケーション設計者が手動で作成する必要があった．本章では，(s_0, T, I)-confluent ではないオブジェクトを対象として，オブジェクトが (s_0, T, I, Σ)-confluent となるようなセグメント分解 Σ を求めるための手法を提案する．提案手法は，不変条件を制限してセグメントを作成する技術と，トランザクション集合を制限してセグメントを作成する技術から構成され，それぞれにより作成されたセグメントを用いて セグメント分解を作成する．不変条件を制限する手法では，Invariant closed，ゆえに Invariant confluent となることが保証されたセグメントを作成し，トランザクション集合を制限する手法では Invariant confluent となることが期待されるセグメントを作成する．1 つめの不変条件を制限する手法では，セグメントの部分不変条件が Invariant closed となるように不変条件を制限してセグメントを構築する．その際，Invariant confluent の反例となる状態同士は異なるセグメントの部分不変条件に属しているべきであるという観察を用いる．2 つめのトランザクション集合を制限する手法では，Invariant confluent の反例となる状態に到達させうるトランザクションペアを特定し，実行可能なトランザクション集合から除外したセグメントを構築する．

本章では，3.1 節において不変条件を制限する手法について，3.2 節においてトランザクション集合を制限する手法について述べる．最後に 3.3 節で不変条件の制限とトランザクション集合の制限の 2 つの手法から構成される提案手法のアルゴリズムをまとめる．

3.1 不変条件の制限

オブジェクトが (s_0, T, I)-confluent ではないとき，定義 2 から $s_1 \sqcup s_2 \notin I$ を満たす到達可能な状態 $s_1, s_2 \in I$ が存在する．このような反例となる状態は SMT ソルバを用いた I-closed 検査により得られる [7]．不変条件を制限して Invariant closed となるセグメントの部分不変条件を作成するためには，このような反例 s_1, s_2 同士は異なるセグメントの部分不変条件に属していなくてはいけないことがわかる．

そこで，提案手法では反例 s_1, s_2 を基にして，それぞれから Invariant closed となることが期待される部分不変条件を有するセグメントを作成する．ただし，後に示すように作成した部分不変条件は Invariant closed とはならない場合がある．その場合，ユーザからの入力にもとづいて部分不変条件をさらに制限することで Invariant closed を満たすことを目指す．

以下，3.1.1 項で部分不変条件の作成方法について，3.1.2 項 でユーザが入力するさらなる制限と，その制限が満たすべき必要条件について述べる．最後に 3.1.3 項で不変条件を制限するアルゴリズムについてまとめる．

3.1.1 作成する部分不変条件

I-closed 検査で得られる，(s_0, T, I)-confluent の反例となる，状態のペアを s_1, s_2 とする．片方の s_i $(i = 1, 2)$ を含んで I_i-closed となる部分不変条件 I_i に必要な条件として次の 2 つが考えられる．

(C1) $I_i \subset \{\, s \mid s \in I,\, s \sqcup s_i \in I \,\}$
(C2) I_i-closed である

条件 (C1) が必要である理由は，I_i が s_i を含んで I_i-closed となる場合，任意の $s \in I_i$ は $s \sqcup s_i \in I_i$ ゆえに，$s \sqcup s_i \in I$ を満たすためである．そこで，提案手法では条件 (C1) を満たすために s_i から次のようにして部分不変条件 I_i を作成し，

$$I_i = \{\, s \mid s \in I,\, s \sqcup s_i \in I \,\} \tag{2}$$

I_i が条件 (C2) を満たすかを I_i-closed 検査により確認する．

この部分不変条件の構成方法から，反例同士は同じ部分不変条件に含まれないことがわかる．なぜなら，例えば s_1 に注目したとき，他方の反例 s_2 に対して $s_1 \sqcup s_2 \notin I$ が成り立つためである．また，I_i は必ず元の不変条件 I の部分集合となっているため，セグメントの部分不変条件になり得る．

3.1.2 ユーザによる部分不変条件の制限

上のように作成した I_i は必ずしも I_i-closed とはならない．それを示す例を次の例 4 に示す．

例 4 (I_i-closed とならない例)．レプリカ数 2 の PN-counter [9] を考える．状態集合は $S = \{\, ((p_1, p_2), (n_1, n_2)) \mid p_1, p_2, n_1, n_2 \in \mathbb{N} \,\}$ で，マージ $\sqcup$ は要素ごとの最大値をとる操作である．初期状態は $((0,0),(0,0))$ であるとする．p_i, n_i はレプリカ i でのみトランザクションによる更新が可能であり，p_i はレプリカ i でインクリメントされた回数，n_i はレプリカ i でデクリメントされた回数を表す．カウンタの値は，すべてのレプリカのインクリメントの回数の和 $(p_1 + p_2)$ からデクリメントされた回数の和 $(n_1 + n_2)$ を引くことで求める $(p_1 + p_2 - n_1 - n_2)$．不変条件を $I = \{\, ((p_1, p_2), (n_1, n_2)) \mid p1 + p2 - n1 - n2 \geqq 0 \,\}$，トランザクション集合を $T = \{\, inc_1, inc_2, dec_1, dec_2 \,\}$ とする．

ただし，ここで inc_i は p_i を 1 インクリメントすることでカウンタをインクリメントする操作で，dec_i は n_i を 1 インクリメントすることでカウンタをデクリメントする操作である．

このオブジェクトは (s_0, T, I)-confluent ではない．反例は $s_1 = ((0,1),(1,0)), s_2 = ((1,2),(0,3))$ である．反例 $s_1 = ((0,1),(1,0))$ に対して I_1 を考えると，I_1-closed ではないことがわかる．なぜなら，$s'_1 = ((1,2),(3,0)) \in I_1, s'_2 = ((2,3),(2,3)) \in I_1$ に対して $s'_1 \sqcup s'_2 = ((2,3),(3,3))$ を考えると，$s'_1 \sqcup s'_2 \notin I$ かつ $s'_1 \sqcup s'_2 \sqcup s_1 = ((2,3),(3,3)) \notin I$ となるためである．

このような場合，部分不変条件をさらに制限することで Invariant closed となる部分不変条件を作成することが可能な場合がある．具体的には，I_i に対して加える制限を I_{add} としたとき，次のような $I_{i-refined}$ の $I_{i-refined}$-closed 性を考える．

$$I_{i-refined} = \{\, s \mid s \in I,\, s \sqcup s_i \in I,\, s \in I_{add} \,\} \qquad (3)$$

加える制限 I_{add} として最も望ましいものは，$I_{i-refined}$ が $I_{i-refined}$-closed となることを保証するようなものである．しかし，そのような制限は一般に明らかではない．提案手法では，I_{add} はユーザからの入力を想定する．ユーザはどのような制限でも入力可能であるが，$I_{i-refined}$-closed となるように I_i を制限するためには I_{add} が満たすべき条件が存在する．ユーザはその条件を満たすように I_{add} を考え入力する．以下，その条件について述べる．

I_i は s_i を含むが I_i-closed ではない．そのため，I_i-closed 検査により得られる反例 s'_1, s'_2 が存在する．$I_{i-refined}$ は s_i を含んで $I_{i-refined}$-closed となることを目標としているため，I_{add} は s'_1, s'_2 を共に含んでいてはいけない．そこで，ユーザが入力する制限 I_{add} は (R1) s_i を含み，s'_1, s'_2 のいずれか一方を含むか (R2) s_i を含み，s'_1, s'_2 の両方を含まないことが必要である．条件 (R1) は例えば，s_i と，s'_1 もしくは s'_2 が満たす共通の条件から I_{add} を決める場合が考えられる．条件 (R2) は例えば，$I_{1-refined}$ の作り方から I_2 に対して入力する I_{add} を決める場合が考えられる．

$I_{i-refined}$ となる I_{add} を見つけることができない場合は，$I_{add} = \emptyset$ としてセグメントを作成しない．次の例 5 で I_{add} を適切に選ぶことで，$I_{i-refined}$-closed な部分不変条件を作成可能であることを確認する．

例 5. 例 4 でのオブジェクトと反例 s_1, s_2 を考える．s_1 に対して作成した部分不変条件 I_1 は I_1-closed ではないことがわかった．反例は $s'_1 = ((1,2),(3,0)) \in I_1$, $s'_2 = ((2,3),(2,3)) \in I_1$ であった．I_{add} として $s_1 = ((0,1),(1,0))$ および $s'_1 = ((1,2),(3,0))$ が満たす条件である $n_2 = 0$ をとると，$I_{1-refined} = \{\, s \mid s \in I, s \sqcup s_i \in I, n_2 = 0 \,\}$ は $I_{1-refined}$-closed となる．

3.1.3 不変条件を制限するアルゴリズム

不変条件を制限する手法のアルゴリズムを Algorithm 2 に示す．アルゴリズムは I-closed の反例 s_1, s_2 を入力として，

Algorithm 2 不変条件の制限

Input: s_1, s_2, I
Output: I_1, I_2

```
1:  function GenClosedInvariant(s, I)
2:      I' ← { s' | s' ∈ I, s' ⊔ s ∈ I }
3:      I_add ← { s' | s' ∈ I }
4:      while I_add ≠ ∅ do
5:          if CheckIfIclosed(I' ∩ I_add) then
6:              return I' ∩ I_add
7:          I_add ← GetRestrictionFromUser()
8:      return ∅
9:
10: function RestrictInvariant(s_1, s_2, I)
11:     I_1 ← GenClosedInvairant(s_1, I)
12:     I_2 ← GenClosedInvairant(s_2, I)
13:     return I_1, I_2
```

Invariant closed であることが保証される I の部分不変条件 I_1, I_2 を返す．アルゴリズムは，まず RestrictInvariant を呼び出す (10 行目)．RestrictInvariant では，s_1, s_2 のそれぞれに対して GenClosedInvariant を呼び出し Invariant closed となる部分不変条件 I_1, I_2 を作成する．得られた I_1, I_2 をアルゴリズムの最終的な出力として返す (11–13 行目)．関数 GenClosedInvariant は状態 s と不変条件を入力として受け取り，s を含んで Invariant closed となる部分不変条件を返す．GenClosedInvariant では，まず式 (2) に基づき s から I' を作成する (2 行目)．次に，ユーザからの制限である I_{add} を不変条件で初期化しておく (3 行目)．続いて，ユーザからの制限が $\emptyset$ ではない限り，$I' \cap I_{add}$ が $(I' \cap I_{add})$-closed であるかを検査し，ユーザからの入力を受けることを繰り返す (4–7 行目)．このとき，$(I' \cap I_{add})$-closed になれば $I' \cap I_{add}$ を返す (5–6 行目)．$I' \cap I_{add}$ が Invariant closed にならず，ユーザからの制限が $\emptyset$ になれば，そのまま $\emptyset$ を返す (8 行目)．

3.2 トランザクション集合の制限

前節では，Invariant confluent ではないオブジェクトに対して不変条件を制限することで，到達可能な状態を削減し Invariant confluent なセグメントを作成する手法について述べた．本節では，トランザクション集合を制限することで，到達可能な状態を削減し，Invariant confluent になることが期待されるセグメントを作成する手法について述べる．提案手法はトランザクションペアが不変条件に関して Conflict となる場合に注目する．Conflict となるトランザクションペアを含むセグメントは必ず Invariant confluent にならないことを活用し，トランザクションを除去していく．

以下，まず Conflict の定義を示し，Conflict となるトランザクションペアを含むセグメントは必ずある初期状態について Invariant confluent にならないことについて述べる．そのあとトランザクション集合を制限するアルゴリズムについて述べる．

Conflict は I と $\sqcup$ に依存し，$(I, \sqcup)$-conflict と書く．

定義 7 ($(I, \sqcup)$-conflict)．トランザクションペア t_1, t_2 が $(I, \sqcup)$-

conflict であるとは，ある $s \in I$ が存在して，$t_1(s), t_2(s) \in I$ かつ $t_1(s) \sqcup t_2(s) \notin I$ を満たすことをいう．

次の定理 2 に示すように，$(I, \sqcup)$-conflict となるトランザクションペアは，Invariant confluent の反例が存在することを保証する．

定理 2. $(I, \sqcup)$-conflict となるトランザクションペアをトランザクション集合 T に含むとき，ある初期状態 $s \in I$ に関してオブジェクトは (s, T, I)-confluent ではない．

証明. t_1, t_2 は $(I, \sqcup)$-conflict であるので，定義から，ある状態 $s \in I$ が存在して，$t_1(s), t_2(s) \in I$ かつ $t_1(s) \sqcup t_2(s) \notin I$ を満たす．この s を初期状態とすると，$t_1(s), t_2(s) \in I$ は (s, T, I)-confluent の反例となる．よって，オブジェクトは初期状態 s に関して (s, T, I)-confluent ではない． □

この事実から，作成するセグメントには $(I, \sqcup)$-conflict となるトランザクションペアを含んではいけないことがわかる．そこで，提案手法では $(I, \sqcup)$-conflict となるトランザクションペアが存在しなくなるように，トランザクション集合からトランザクションを除去する．具体的には，トランザクションペア $t_1, t_2 \in T$ が $(I, \sqcup)$-conflict であるとき，t_1 と t_2 のどちらかをトランザクション集合から除去する．この際，t_1 と t_2 のどちらを除去しても良いが，例えば実行頻度等からトランザクションに重要度が存在する場合には，重要度の低いトランザクションを除去する．

Algorithm 3 にトランザクション集合を制限する手法のアルゴリズムを示す．アルゴリズムはオブジェクトのトランザクション集合 T，不変条件 I を入力として，制限されたトランザクション集合 T' を返す．まず，T' を T で初期化し，T から除去するトランザクションを要素とする集合 $T_{restrict}$ を空集合で初期化する (2–3 行目)．$\{t_1, t_2\}$ の形をしたすべてのトランザクション集合について，$(I, \sqcup)$-conflict かを検査する．$(I, \sqcup)$-conflict であれば，関数 ChooseTransaction により t_1, t_2 のうち除去するトランザクションを選択し，t_r とする．そして，選択されたトランザクション t_r を変数 $T_{restrict}$ に加える (4–7 行目)．すべてのペアについて検査を行ったら，T から $T_{restrict}$ を除き，制限されたトランザクション集合 T' を返す (8–9 行目)．

Algorithm 3 トランザクション集合の制限

Input: T, I
Output: T'

```
1: function RestrictTxSet(T, I)
2:     T' ← T                                   ▷ initialize T'
3:     T_restrict ← ∅          ▷ T' = T \ T_restrict is final result
4:     for {t1, t2} ∈ 𝔓(T) do
5:         if CheckIfConflict(t1, t2, I) then
6:             t_r = ChooseTransaction(t1, t2)
7:             T_restrict ← T_restrict ∪ {t_r}
8:     T' ← T \ T_restrict
9:     return T'
```

3.3 提案手法のアルゴリズム

提案手法は 3.1 節，3.2 節での手法により作成されたセグメントを用いてオブジェクトが Segmented invariant confluent となるようなセグメント分解を求める．この際の課題は，作成するセグメント分解が Segmented invariant confluent の定義 (定義 6) を満たすようにすることである．本稿では，定義 6 の条件 (S1) は満たされていることを前提としている．以下，提案手法において条件 (S2) および (S3) を保証するための方法を説明する．

条件 (S2) を満たすためには，セグメント分解に使用するセグメントの部分不変条件の和 $\bigcup I_i$ が不変条件 I に等しいことを保証する必要がある．そこで提案手法では，$I \setminus \bigcup I_i$ が空でなければ，その条件 $(I \setminus \bigcup I_i)$ を部分不変条件として持つ Invariant confluent なセグメント $(I \setminus \bigcup I_i, \emptyset)$ を新たに作成し $I = \bigcup I_i$ が成り立つことを保証する．

条件 (S3) を満たすためには，セグメント分解に使用するすべてのセグメントが Invariant confluent であることを保証する必要がある．不変条件を制限して作成されるセグメントは常に Invariant confluent になることが保証されるが (定理 1)，トランザクション集合を制限して作成されるセグメントは必ずしも Invariant confluent にならない．そこで，提案手法ではトランザクション集合を制限して作成されたセグメントについては，セグメントが Invariant confluent になるかを事前に検査する．この検査には，対話的に Invariant confluent であるかを検証する既存の技術を用いることができる [7]．条件 (S2) について，追加のセグメント $(I \setminus \bigcup I_i, \emptyset)$ を作成する必要があるのは，トランザクション集合を制限して作成されたセグメントが Invariant confluent ではない場合に限られることに注意する．なぜなら，トランザクション集合を制限して作成されたセグメントの部分不変条件は常に不変条件に一致しているためである．

Algorithm 4 にセグメント作成のためのアルゴリズムを示す．アルゴリズムは MakeSegmentation を呼び出しセグメント分解を求める (1 行目)．MakeSegmentation ではまず，RestrictInvariant (Algorithm 2) を呼び出して Invariant closed となる部分不変条件 I_1 および I_2 を得る (2 行目)．次に RestrictTxSet (Algorithm 3) を呼び出して制限されたトランザクション集合 T' を得る (3 行目)．続いて，トランザクション集合を制限して得られたセグメント (I, T') が Invariant confluent であるかを検査し，Invariant confluent であれば $\Sigma = (I_1, T), (I_2, T), (I, T')$ として，セグメント分解を作成する．作成した Σ に対して RemoveEmptySegment を呼び出し $I_i = \emptyset$ となるセグメントを除去した後，返却する (4–6 行目)．(I, T') が Invariant confluent ではない場合，I_1 と I_2 の和が I に一致するかを検査する．一致しない場合は $(I_1, T), (I_2, T)$ に空のセグメント $((I \setminus (I_1 \cup I_2)), \emptyset)$ を加えたものを Σ とし，Σ に対して RemoveEmptySegment を適用した後，返却する (7–9 行目)．I_1 と I_2 の和が I に一致する場合，$\Sigma = (I_1, T), (I_2, T)$ とし，Σ に対して RemoveEmpty-

Algorithm 4 セグメント分解の作成

Input: $s_1, s_2, T, I,$

Output: Σ

1: **function** MakeSegmentation(s_1, s_2, T, I)
2: $I_1, I_2 \leftarrow$ RestrictInvariant(s_1, s_2, I)
3: $T' \leftarrow$ RestrictTxSet(T, I)
4: **if** IsInvConfluent(I, T') **then** ▷ Invariant is covered
5: $\Sigma \leftarrow (I_1, T), (I_2, T), (I, T')$
6: **return** RemoveEmptySegment(Σ)
7: **if** $I_1 \cup I_2 \subsetneqq I$ **then** ▷ Invariant is not covered
8: $\Sigma \leftarrow (I_1, T), (I_2, T), ((I \setminus (I_1 \cup I_2)), \emptyset)$
9: **return** RemoveEmptySegment(Σ)
10: $\Sigma \leftarrow (I_1, T), (I_2, T)$
11: **return** RemoveEmptySegment(Σ)

Segment を適用した後，返却する (10–11 行目)．

4 評　　価

本章では提案手法を複数の具体例 [7] に適用した結果を示し，確かに Segmented invariant confluent となるセグメント分解が求められていることを確認する．Invariant closed 検査および $(I, \sqcup)$-conflict 検査には SMT ソルバ Z3 [8] を用いた．本稿では，例 8 での A_X, R_X, A_Y, R_Y，例 9 での B は $A = \{1, \ldots, 19\}$ の部分集合として検証を行った．

例 6. 例 2 のオブジェクトとトランザクション集合，不変条件を考える．このオブジェクトは (s_0, T, I)-confluent ではなく，反例は $s_1 = (3, -2)$, $s_2 = (-2, 7)$ であった．このオブジェクトに対して提案アルゴリズムを適用すると，まず不変条件を制限する手法により，s_1, s_2 のそれぞれから

$$\begin{aligned} I_1 &= \{\, (x,y) \mid (x,y) \in I,\ (x,y) \sqcup (3,-2) \in I \,\} \\ &= \{\, (x,y) \mid y = 0 \,\} \cup \{\, (x,y) \mid x \geqq 0,\ y \leqq 0 \,\} \\ I_2 &= \{\, (x,y) \mid (x,y) \in I,\ (x,y) \sqcup (-2,7) \in I \,\} \\ &= \{\, (x,y) \mid x = 0 \,\} \cup \{\, (x,y) \mid x \leqq 0,\ y \geqq 0 \,\} \end{aligned}$$

が作成される．I_1 と I_2 はそれぞれ，I_1-closed，I_2-closed となっている．トランザクション集合を制限する手法では，T のどのトランザクションペアをとっても $(I, \sqcup)$-conflict とはならない．そのため，トランザクション集合から除去されるトランザクションは存在せず，トランザクション集合を制限してできるセグメントは (I, T) となる．O はこのセグメントに関して Invariant confluent とはならない．よって，セグメント分解は不変条件を制限することで得られたセグメントのみから作成する．不変条件を制限することにより得られたセグメント (I_1, T_1), (I_2, T_2) について，その部分不変条件の和

$$\begin{aligned} I_1 \cup I_2 =& \{\, (x,y) \mid y = 0 \,\} \cup \{\, (x,y) \mid x \geqq 0,\ y \leqq 0 \,\} \cup \\ & \{\, (x,y) \mid x = 0 \,\} \cup \{\, (x,y) \mid x \leqq 0,\ y \geqq 0 \,\} \\ =& \{\, (x,y) \mid xy \leqq 0 \,\} = I \end{aligned}$$

は不変条件に一致するため，$\Sigma = (I_1, T), (I_2, T)$ が最終的なセグメント分解として得られる．$s_0 = (-42, 42) \in I$ であるから，O は (s_0, T, I, Σ)-confluent となる．

例 7. 例 4 のオブジェクトとトランザクション集合，不変条件を考える．このオブジェクトは (s_0, T, I)-confluent ではなく，反例は $s_1 = ((0,1),(1,0)), s_2 = ((1,2),(0,3))$ であった．不変条件を制限する手法を適用すると，s_1 に対して，$I_1 = \{\, s \in \mathbb{N}^2 \times \mathbb{N}^2 \mid s \in I,\ s \sqcup s_1 \in I \,\}$ がまず作成されるが，I_1 は例 4 で確認したように I_1-closed ではない (反例は $s_1' = ((1,2),(3,0)) \in I_1$, $s_2' = ((2,3),(2,3)) \in I_1$)．そこで，ユーザが I_{add} として s_1 と s_1' がともに満たす条件である $n_2 = 0$ を入力する (例 5) と $I_{1-refined} = \{\, s \in \mathbb{N}^2 \times \mathbb{N}^2 \mid s \in I,\ s \sqcup s_1 \in I,\ n_2 = 0 \,\}$ は $I_{1-refined}$-closed を満たす．s_2 に対して，初めに作成される部分不変条件 $I_2 = \{\, s \in \mathbb{N}^2 \times \mathbb{N}^2 \mid s \in I,\ s \sqcup s_2 \in I \,\}$ は I_2-closed ではない (反例は $s_1'' = ((5,0),(1,4))$, $s_2'' = ((6,1),(0,7))$)．I_2 に対して，ユーザが I_{add} として s_2 と s_2'' がともに満たす条件として $n_1 = 0$ を入力すると，$I_{1-refined} = \{\, s \in \mathbb{N}^2 \times \mathbb{N}^2 \mid s \in I,\ s \sqcup s_1 \in I,\ n_1 = 0 \,\}$ は $I_{2-refined}$-closed を満たす．

トランザクション集合を制限する手法では，$(I, \sqcup)$-conflict となるトランザクションペアとして dec_1, dec_2 が検出される．よって，制限されたトランザクション集合 $T' = T \setminus \{\, dec_2 \,\} = \{\, inc_1, inc_2, dec_1 \,\}$ をトランザクション集合とするセグメント (I, T') が得られる．このセグメント (I, T') は，任意の初期状態 $s \in I$ に対して (s, T', I, Σ)-confluent であると判定できる．よって，セグメント分解として $\Sigma = (I_{1-refined}, T), (I_{2-refined}, T), (I, T')$ が得られる．$s_0 = ((0,),(0,0)) \in I$ であるから，O は (s_0, T, I, Σ)-confluent となる．

例 8. 外部キー制約を表現する例を考える．オブジェクトの状態は 2P-set [9] のペア $((A_X, R_X), (A_Y, R_Y))$ で表現され，マージは各要素ごとの集合和で行う．このオブジェクトは $X(= A_X \setminus R_X)$ が $Y(= A_Y \setminus R_Y)$ の要素を参照する状況を表現している．本稿では，章の冒頭で述べたように，$A_X, R_X, A_Y, R_Y \subset A$ として検証を行った．初期状態は $((\emptyset, \emptyset), (\emptyset, \emptyset))$ であるとする．トランザクション集合は $T = \{\, Insert_X, Delete_X\ Insert_Y, Delete_Y \,\}$ であり，$Insert_X$ は A_X に要素を追加する操作，$Delete_X$ は R_X に要素を追加する操作である．不変条件は $A_X \setminus R_X \subset A_Y \setminus R_Y$ である．このオブジェクトは (s_0, T, I)-confluent ではない．反例は，$s_1 = ((\emptyset, \emptyset), (A, A)), s_2 = ((A, \emptyset), (A, \emptyset))$ である．

このオブジェクトに対して提案手法を適用すると，まず，不変条件を制限する手法により s_1, s_2 のそれぞれから I_1, I_2 が作成されるが，これはともに I_1-closed および I_2-closed を満たす．I_1 は $X = \emptyset$ となる集合のみから構成されるセグメントであり，I_2 は $R_Y \subset R_X$ となるセグメントであると考えられる．例えば，$((\{\,1\,\}, \emptyset), (\{\,1,2\,\}, \emptyset)) \in I_2$ に対して，Y から 2 を削除する操作を適用した状態は不変条件自体は満たすが，I_2 を満たさなく，この操作は調停が必要となる．トランザクションを制限する手法では，$(I, \sqcup)$-conflict であると検出されるト

ランザクションペアは $Insert_X, Delete_Y$ である．よって，制限されたトランザクション集合 $T' = T \setminus \{ Delete_Y \}$ をトランザクション集合としたセグメント (I, T') が作成される．このセグメントでは任意の初期状態について Invariant confluent となる．

よって，セグメント分解 $\Sigma = (I_1, T), (I_2, T), (I, T')$ が求められる．$s_0 = ((\emptyset, \emptyset), (\emptyset, \emptyset))$ は不変条件を満たすことから，O は (s_0, T, I, Σ)-confluent となる．

例 9. オークションアプリケーションの例を考える．$S = \{ (B, w) \mid B \in \mathfrak{P}(\mathbb{N} \setminus \{0\}), w \in \mathbb{N} \}$ とし，マージは B については集合和，w については最大値をとる操作とする．本稿では，本章の冒頭で述べたように $B \subset A$ として検証を行った．初期状態は $(\emptyset, \perp)$ であるとする．ただし，$\perp$ はどんな自然数よりも小さい値である．トランザクション集合は $T = \{ t_b \mid b \in \mathbb{N} \} \cup \{ close \}$ である．t_b は b を B に追加する操作で，$close$ は w に $max(B)$ を設定する操作である．不変条件は $I = \{ (B, w) \mid w \neq \perp \Rightarrow w = max(B) \}$ である．このオブジェクトは (s_0, T, I)-confluent ではない．反例は，$s_1 = (\{1\}, 1)$, $s_2 = (A, -1)$ である．

このオブジェクトに対して提案手法を適用すると，まず，不変条件を制限する手法により s_1, s_2 のそれぞれから I_1, I_2 が作成されるが，ともに I_1-closed および I_2-closed を満たす．I_1 は $w \neq \perp$ となる状態全体，および，状態 $(B, w) = (\{1\}, \perp)$ からなる．I_2 は $close$ が実行されていない状態全体 $(w = \perp)$ および $w = max(A)$ で不変条件を満たす状態全体からなる．I_2 は $w = \perp$ となる状態全体，および $w = max(A)$ で不変条件を満たす状態全体からなる．おおよそ，I_1 はオークションが終了している状態を表していると解釈ができ，I_2 はオークションが進行中の状態を表していると解釈できる．トランザクションを制限する手法では，$(I, \sqcup)$-conflict であると検出されるトランザクションペアは $t_b, close$ である．ここで，実行頻度が低いことが想定される $close$ を除外する．よって，制限されたトランザクション集合 $T' = T \setminus \{ close \}$ をトランザクション集合としたセグメント (I, T') が作成される．このセグメントでは任意の初期状態 $s \in I$ について (s, T', I)-confluent となる．

よって，セグメント分解 $\Sigma = (I_1, T), (I_2, T), (I, T')$ が求められる．$s_0 = (\emptyset, \perp)$ は不変条件を満たすことから，O は (s_0, T, I, Σ)-confluent となる．

5 関 連 研 究

[10] では，調停フリーな実行に基づくキーバリューストア Anna を設計，実装しており，調停が必要となるシステムと比較して優れた性能を示すことが報告されている．[1] では不変条件を守るための RedBlue consistency という一貫性が提案されている．RedBlue consistency ではトランザクションから生成される shadow operation という操作を扱い，shadow operation が不変条件を崩しうる場合，その操作には調停が必要と判定する．[1] では shadow operation は人手で生成されていたが，[2] では特定のデータ型を対象として，自動的に shadow operation を生成する手法を提案している．[3] では，[1] での手法では本来同期する必要がない操作同士まで同期するように判定されてしまうという課題に対して，より細粒度に同期すべき shadow operation を決定するための手法を提案している．[4] では [3] と同様，同期すべき操作を最弱事前条件を用いた静的解析により決定する手法を提案している．[5] では不変条件を満たすシステムを Token system としてモデル化し，システムがトランザクションの実行で不変条件を保つことができるかを検証する方法を提案している．さらに，その検証の自動化を SMT ソルバを用いて実現している．CRDT [9] は結果整合性を提供する，レプリカに対する調停フリーな更新が可能なデータ構造であるが，CRDT そのものは不変条件を扱わない．

6 ま と め

本稿では，Invariant confluent ではないオブジェクトに対して，Invariant confluent なセグメントからなるセグメント分解を作成する手法を提案した．提案手法は不変条件を制限する手法とトランザクション集合を制限する手法から構成される．提案手法は不変条件を制限することで Invariant closed なセグメントを作成し，トランザクション集合を制限することで Invariant confluent となることが期待されるセグメントを作成する．提案手法を複数の具体例に適用することで，Invariant confluent なセグメントからなるセグメント分解を作成可能であることを確認した．

文 献

[1] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making Geo-Replicated Systems Fast as Possible, Consistent When Necessary. In *OSDI*, 2012.

[2] C. Li, J. Leitão, A. Clement, N. Preguiça, R. Rodrigues, and V. Vafeiadis. Automating the Choice of Consistency Levels in Replicated Systems. In *USENIX ATC*, 2014.

[3] C. Li, N. Preguiça, and R. Rodrigues. Fine-Grained Consistency for Geo-Replicated Systems. In *USENIX ATC*, 2018.

[4] V. Balegas, S. Duarte, C. Ferreira, R. Rodrigues, N. Preguiça, M. Najafzadeh, and M. Shapiro. Putting Consistency Back into Eventual Consistency. In *EuroSys*, 2015.

[5] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro. 'Cause I'm Strong Enough: Reasoning about Consistency Choices in Distributed Systems. In *POPL*, 2016.

[6] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination Avoidance in Database Systems. In *VLDB*, 2014.

[7] M. Whittaker and J. M. Hellerstein. Interactive Checks for Coordination Avoidance. In *VLDB*, 2018.

[8] L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, 2008.

[9] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Research Report RR-7506, INRIA, 2011.

[10] C. Wu, J. Faleiro, Y. Lin, and J. Hellerstein. Anna: A KVS for Any Scale. In In *ICDE*, 2018.