Vol.2021-DPS-189 No.9
2021/12/20

長期間のネットワーク分断からの復旧時における データ間に関連が生じるサービスの統合手法

松舘 遼[1]　今井 信太郎[1]　北形 元[2]　新井 義和[1]　猪股 俊光[1]

概要：ネットワーク分断時におけるサービスの可用性の向上において，サービスを複製したサーバを分散配置する方法が有効であるが，各サーバでサービスを提供していくと，各サーバのデータベース間で不整合が発生する．これに対し，書き込み時において，読み取り時からデータが更新されていなかった場合のみ書き込みが成功する楽観的アプローチと呼ばれるデータ統合手法があるが，分断が長期化し多くの書き込みが発生する状況においては，最新データの破棄等の問題が発生するため，利用者が分断解消後にサービスを統合することは困難である．これに対し，操作を記録するコンポーネントをサービスに組み込むことによりこの問題の解決を図る手法が提案されているが，書き込みに対する返信がツリー構造になるような，データ間に関連があるようなサービスへの適用は困難である．本研究では，この問題に対して，メッセージを識別するための ID を振り直す機能を追加することを提案する．さらに，既存の機能も含めて評価実験を実施しその有効性を示す．

1. はじめに

日本は世界の国々と比較して多くの災害が発生する国であり，東日本大震災のような大規模災害も多く発生している．このような大規模災害が発生すると，通信設備の障害や通信量増加による輻輳等の様々なネットワーク障害が長期化することがある．例として，東日本大震災では，通信ケーブルの断線や基地局自体の倒壊，蓄電池や燃料の枯渇等により多くの通信設備障害が発生した．その結果，NTT ドコモ，KDDI，ソフトバンクモバイル，イー・モバイル及びウィルコムの 5 社それぞれが保有する携帯電話及び PHS 基地局で最大約 29,000 局が停波した．この膨大な被害やその後に発生した余震等の影響により，一部エリアを除いて大方復旧するまでに一か月半前後もの期間を要した [1]．これらのことから，ネットワークサービスを提供する際には長期間ネットワークが分断されてしまうような状況においても，サービスを継続させる可用性が求められる．

ネットワークが分断されてしまうような状況における可用性向上のための手法の一つとして，複数のサーバにサービスを複製し，それらを分散配置する方法がある．ネットワークの分断が発生してしまった場合でも，分散配置したサーバが同一のサービスを立ち上げることにより，利用者はいずれかのサーバに接続することができれば，サービスを継続して利用することができる．しかし，ネットワークが分断している状況では，分散配置されたサーバ間で通信することが困難であり，すべてのサーバでデータベースの内容を統一することが不可能となってしまう．

よって，ネットワークの分断が解消された際には，同一のエンティティに対して実行されたトランザクションから，プライマリサーバへ反映させる一つのトランザクションを選択する等の手法が必要となり，複数のサーバ及びデータベース間で不整合が発生しないようにデータを統合することが求められる．

本研究は，正常時には単一のサーバとデータベースがサービスを提供しつつ，バックアップを他のサーバおよびデータベースに定期的に作成しておき，ネットワーク分断時に本来のサーバから分断された側で他のサーバがサービスを提供し，復旧時に本来のサーバおよびデータベースのみがサービスを提供する状態に戻すような環境を対象とする．

データベース間の不整合を防ぐ手法として，楽観的並行性制御をはじめとした楽観的アプローチと言われる手法を用いる方法がある [2–7]．この手法は，データの書き込み時において，読み取り時からデータが更新されていなかった場合のみ書き込みが成功する手法である．このため，この手法では書き込みが頻繁に発生するような場合，競合箇

[1] 岩手県立大学 大学院ソフトウェア情報学研究科
Graduate School of Software and Information Science，Iwate Prefectural University
[2] 東北大学 電気通信研究所
Research Institute of Electrical Communication，Tohoku University

所の増加により，多くの書き込みに失敗し多数のトランザクションが破棄されてしまうため，利用者が書き込んだはずの最新のデータが破棄される等の問題が発生し，本研究が対象とするネットワークの分断が長期化するような場合には，単純に適用することは困難である．

この問題に対し，大坂らはユーザがサービスに対して行う操作を記録するコンポーネントをサービスに組み込むことにより解決を図るデータ管理手法を提案している [8]．この研究では，サービスの長期的な孤立運用後にデータを統合する際に生じるデータの不整合について考察し，それらの不整合を解消するような手法について提案している．しかし，書き込みに対する返信がツリー構造になるような，データ間に関連があるようなサービスについては考慮されていない．また，この手法が具体的に適用可能なサービスについてや，統合処理のサービスに与える影響については明らかにしていない．

以上のことから本研究では，ネットワークが長期間分断されてしまった状況から復旧する際におけるデータ間に関連が生じるサービスの統合手法について提案する．本提案手法では，各サーバでユーザの操作をログファイルに記録しておき，復旧後に本来のサーバがそのログファイルを収集して統合し，複数のサーバで行われた操作を障害発生前からサービスを提供しているサーバで再現する．その際に，データ間に関連が生じるサービスについては，ログファイルを統合する前にメッセージを識別するための ID を振り直す機能を追加することでデータ間の関連を維持したまま，サービスを統合することが可能となる．

2. 本研究の対象

2.1 本研究が対象とする環境

本研究は，大規模災害の発生等によってネットワークの分断が長期化し，その後に復旧するような状況を想定しており，障害発生前から稼働するサーバをメインサーバ，障害発生時から復旧時にかけて稼働するサーバをサブサーバとする．各サーバの構成としては，メインサーバ１台，サブサーバ N 台 (N>0) とし，各サーバは別々のネットワークセグメントに存在するものとする．また，障害発生前までは，メインサーバからサブサーバへレプリケーションを実行し，各サーバのデータベース間におけるデータの整合性を保っているものとする．利用者それぞれの情報に関する操作ログについては，利用者個人単位での時系列的なタイムスタンプの整合がとれているものとする．

対象とするサービスの特徴としては，サービスを提供するデータベースにおいてレコードを一意に決定することができるという一意性のあるフィールドをもつものを対象とする．例として，電話番号やメールアドレス，SNS 等のアカウントに紐づくユーザ ID 等のようなものが挙げられる．

また，サーバ間の通信が可能な状態と不可能な状態が短期間のうちに断続して発生せず，ネットワークの分断発生と復旧が明確な障害について対象とする．ビザンチン障害のような合意形成を必要とするような障害については，対象外とする．

2.2 本研究が対象とするサービスの特性

分散データベースにおいて整合性を保証するためにトランザクションが満たすべき特性として ACID 特性と BASE 特性がある [9].

ACID 特性は, 原子性（Atomicity）, 一貫性（Consistency）, 独立性（Isolation）永続性（Durability）の 4 つの特性のことで，これらすべてを満たすことで完全な整合性をとることが可能である．完全な整合性をとる必要があるサービスの例として，金融機関向けの基幹系システムなどが考えられる [10].

一方で BASE 特性は, ネットワークが分断されるような状況でも基本的にいつでも利用できる（Basically Available）, データの更新は複数のノードに対して徐々に更新内容が反映される（Soft-state）, 時間は多少かかっても結果的な整合がとれる（Eventual consistency）といった 3 つの特性を満たすことで，結果整合性をとることが可能である．結果整合性を必要とする例として，Web メールやツイッターをはじめとしたクラウドサービスなどが考えられる [11].

Brewer は，分散データベースのクエリは ACID 特性か BASE 特性のどちらかを満たすべきであると主張し，このことが広く受け入れられている [12]．したがって，サービスの分類として ACID 特性を満たすような完全な整合性が必要なものと BASE 特性を満たすような結果整合性を満たすものの大きく分けて 2 つに分類できると考えられる．

本研究は，ネットワークが長期間分断されるような状況を想定し，サーバ間の通信が困難な状況でもサービスを継続して利用することを前提としているため，対象とするサービスとしてはネットワークの分断に耐性のあることが必須である．また，本提案システムは，ネットワークの分断が解消されたサーバから，順次サービスを統合し最新の内容に更新しつつ，最終的には全サーバ及びデータベースで整合をとることができる．以上のことから，本研究が対象とするサービスの特性としては，ACID 特性を満たすサービスというよりは，BASE 特性を満たすサービスが対象となると考えられる．

3. 関連研究

3.1 楽観的並行性制御

データの不整合を防ぐ統合のための手法として，楽観的アプローチの一つである楽観的並行性制御がある [2]．サブサーバが自身で発生した変更内容をメインサーバに書き込む際には，サブサーバが自身とメインサーバで競合が発生していないか検証を行い，競合が発生していない場合の

みサブサーバが自身の変更内容をメインサーバに書き込み，そうでない場合は変更内容を破棄する．この手法では，各エンティティごとに付与されるトランザクション ID がメインサーバへの書き込みやメインサーバの更新時に加算されることを利用する．サブサーバの変更内容をメインサーバに書き込み可能かどうかの検証では，このトランザクション ID が一致しているかを確認し，一致していた場合のみ書き込み，それ以外はサブサーバの変更を破棄する．例として，安否確認サービスに適用した場合の挙動について図 1 と図 2 に示す．

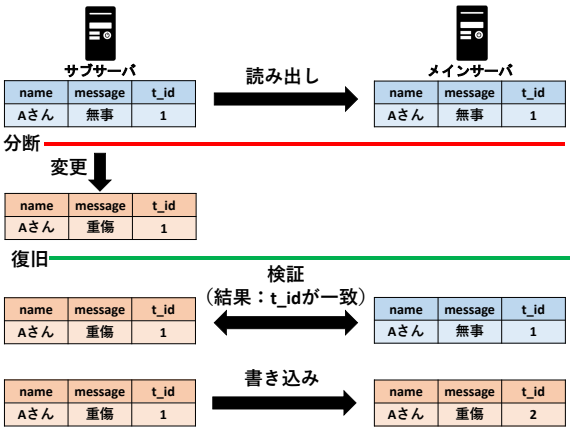

図 1　楽観的並行性制御（書き込み成功）

まず，図 1 では，データの競合が発生せず，サブサーバの変更内容がメインサーバに書き込まれた場合を示している．最初に，サブサーバがメインサーバのデータを読み出し，「A さんが無事である」という内容を取得する．その後ネットワークが分断し，サブサーバで「A さんが重傷である」という内容に変更があったとする．その後ネットワークが復旧し，サービス統合時にメインサーバへ書き込み可能であるかどうかを検証するが，図 1 では，ネットワークが分断してから復旧するまでに，メインサーバでは更新が発生していないため，サブサーバのトランザクション ID とメインサーバのトランザクション ID が一致する．したがって，サブサーバは変更内容をメインサーバに書き込む．

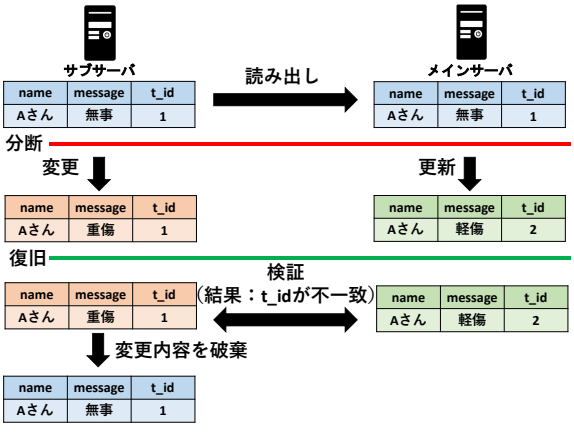

図 2　楽観的並行性制御（書き込み失敗）

次に，図 2 では，データの競合が発生し，サブサーバの変更内容をメインサーバに書き込むことができなかった場合を示している．図 1 との違いは，ネットワーク分断後にメインサーバで「A さんが軽傷である」という内容への更新がある点である．このとき，メインサーバでは更新が発生したためトランザクション ID が加算され．その後，サブサーバで「A さんが重傷である」という内容に変更したとする．この段階でネットワークが復旧し，検証が行われると，サブサーバとメインサーバでトランザクション ID が不一致となるため，サブサーバは変更内容を破棄する．

このように，楽観的並行性制御をはじめとする楽観的アプローチは，書き込み対象のエンティティにおいて，トランザクション ID が加算された時点で，加算以前に読み出したデータに対する変更内容はすべて破棄されてしまう．したがって，大規模災害発生時のように長期間ネットワークが分断されてしまう状況では多くのトランザクションが実行されることによるロングトランザクションの発生や競合箇所の増加が生じ，多数のトランザクションが破棄されてしまう可能性が非常に高まるため，本来必要であったデータを破棄してしまう可能性が高くなる．したがって，最新のデータを破棄してしまう等の意味的な不整合が発生する可能性が高くなるため，データの競合箇所を修正しながらも，多くのトランザクションの破棄を防ぐことが必要である．

3.2　操作を記録したコンポーネントを組み込むデータ管理手法

大坂らは，長期間のネットワーク分断時に意味的な不整合を解消するために操作を記録したコンポーネントをサービスに組み込むデータ管理手法を提案した [8]．この手法では，各サーバで実行された操作をデータベースに記録し，ネットワーク障害が発生しなかったと仮定した際に想定される操作を再現する．この研究では，ネットワークの分断状態が長期間続いた場合に起こりうる不整合について考察し，それらの不整合を解消する手法について提案している．また，考察した不整合が発生する状況を実際に再現し，提案手法を適用した試作システムを用いた実験を通じて，有効性を示している．しかし，書き込みに対する返信がツリー構造になるような，データ間に関連があるようなサービスについては考察されておらず，この手法ではそのようなサービスには対応できない．また，この手法が具体的に適用可能なサービスや実際にユーザのアクセスを再現した実験については実施しておらず，実際に統合に要する処理時間等については明らかになっていない．

4.　提案手法

4.1　提案システムの概要

本研究では，大坂らの研究 [8] で提案されている提案機

構を既存機能とし，既存機能に加えて本研究の提案する機能を追加する．ただし，本研究では，操作の記録方法として，データベースに記録するのではなくログファイルに操作を記録する．まず，障害発生前からメインサーバがサービスを提供し，メインサーバがサブサーバへレプリケーションを行う．この状況でネットワークが分断されると，サブサーバは，自動的にこの障害を検知し，メインサーバと同一のサービスを起動して，サービスを提供する．その後，ネットワークが復旧した際には，サブサーバが復旧を検知し，サービスを停止する．サブサーバがサービス停止後，サブサーバで記録していたデータベースへの操作ログファイルをメインサーバへ送信する．メインサーバでは，サービスを停止後サブサーバからログファイルを取得し，サブサーバから取得したログファイルとメインサーバ自身のログファイルから，4.4 節で述べる処理を施した上で統合し，統合ログファイルを作成する．メインサーバが統合ログファイルを使用して，自身のデータベースにおいて時系列に操作を実行し，データベースへの統合処理を行い，完了後にサービスを再開する．

4.2 複数のログファイル統合と実行

統合処理はメインサーバで実行する．図 3 にメインサーバにおける統合処理のフローチャートを示す．まず，障害復旧後に障害時にサービスを提供していたサブサーバからログファイルを取得し，データ間に関連があるサービスであれば，4.4 節で述べるメッセージを識別するための ID を振り直す処理を実行する．次に，メインサーバ自身が記録していたログファイルとサブサーバから取得したログファイルをログファイル上のタイムスタンプを手がかりに統合し，時系列にソートする．本研究では，2 節で述べたようにユーザそれぞれの情報に関する操作ログについては，ユーザ個人単位での時系列的なタイムスタンプの整合がとれているものとするため，タイムスタンプを使用してソートする．時系列にソート後，ソートした操作のログを全操作の履歴として書き出し，統合ログファイルを作成する．統合ログファイルを作成後，一意性のあるフィールドが変更されるサービスの場合は，4.5.1 節で述べる各ユーザにおける一意性のあるフィールドの変更履歴を作成し，その変更履歴をもとに統合ログファイルとメインサーバのデータベースに対して一意性のあるフィールドを最新の状態にする．メインサーバは，統合ログファイルの情報をもとに自身のデータベースへ操作を実行する．これにより，各サーバで実行されたデータベースに対する操作を時系列に実行することが可能となり，最新データ破棄等の意味的な不整合を解消することが可能となる．

4.3 統合ログファイルから統合処理を実行する際の問題点

統合ログファイル上の情報からデータベースへの統合処理を実行する際に，統合ログファイルをそのまま実行してしまうと以下のような問題点が発生する．

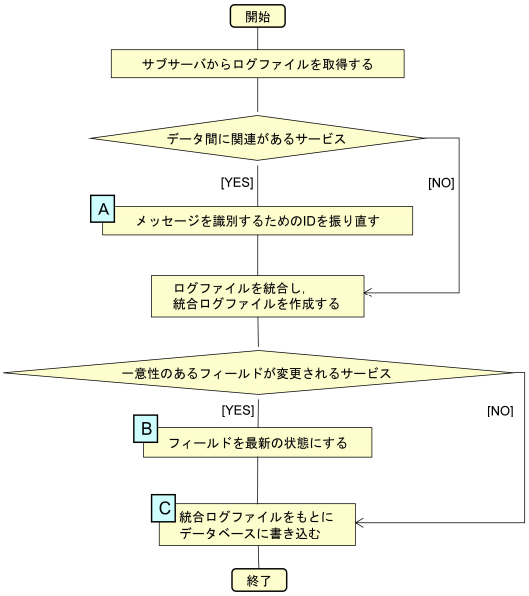

図 3 メインサーバにおける統合処理のフローチャート

1. データ間の関連損失
2. 一意性のあるフィールドの変更
3. 追加処理の重複
4. 存在しない対象の更新処理

2〜4 については既存機能で対応可能であるが，1 については既存機能では対応不可能である．本研究では，既存機能で対応不可能な 1 についての対応手法について提案する．

4.4 データ間の関連損失への対応

SNS 等のサービスでは，自身のメッセージだけではなく，誰に送信したものかが重要となる．例えば，ツイッター等のサービスでは，ユーザ自身がツイートを投稿するだけでなく，あるツイートにリプライするような場合が考えられる．図 4 にデータ間に関連があるサービスの例を示す．図 4 の左側の棒は返信先と返信元を表しており，ひし形側が返信元，矢印側が返信先を示している．矢印がないものについては，返信先がなく単体で投稿されたメッセージを表している．図 4 からわかるようにデータ間に関連があるサービスは，登録されたデータに対して複数の分岐が生じ，分岐したデータから再び分岐が生じるような特徴を持つ．複数のサーバでこのサービスを提供すると，図 4 のような分岐が複数のサーバで発生するため，返信元と返信先の関連を維持するような統合をしなければ，本来とは別の返信先へメッセージを投稿したことになってしまう．

これを踏まえて，図 5 にデータ間の関連を考慮せずに統合した際の結果について示す．図 5 では，ネットワークが分断してから復旧するまでに，各サーバで別々のデータが登録されてしまうため，メッセージを識別するための ID である投稿 ID4〜6 が重複してしまう．ログ情報をもとにメ

図 4　データ間に関連があるサービスの例

インサーバがサブサーバのデータをそのまま登録してしまうと，図 5 の赤い四角枠で囲まれた部分のようにサブサーバにおける投稿 ID4～6 のデータが，メインサーバに登録された投稿の次の投稿 ID である 7～9 に順次割り振られてしまう．しかし，返信先 ID はそのままの状態で登録されるため，本来の返信先とは別の返信先へメッセージを投稿したことになり，データ間の関連が損なわれてしまう．例えば，図 5 の赤い丸枠で囲まれた部分のように投稿 ID8,9 の返信先は，本来は統合後の投稿 ID7 への返信であるのに対し，別の返信先である投稿 ID4 への返信に変わってしまう．

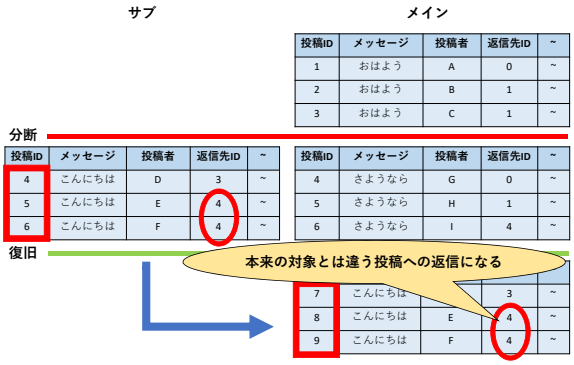

図 5　データ間の関連を維持できない統合

次に図 6 では，データ間の関連を考慮して統合した際の結果を示している．図 6 はネットワークが分断してから復旧するまで図 5 と同じ状況であるが，統合の際に投稿 ID の変更に伴って返信先 ID も同様に変更している．具体的に説明すると，図 6 の赤い四角枠の部分のようにサブサーバにおける投稿 ID4～6 のデータが統合後に投稿 ID7～9 に順次割り振られ，図 6 の赤い丸枠の部分ではそれに伴って投稿 ID8,9 の返信先 ID である 4 が 7 に変更されている．この処理を加えることで，本来の返信先に返信したことになり，データ間の関連を維持することが可能となる．この処理は，図 3 の A にあたる部分で実行される．

4.5　既存機能で対応可能な問題点と対応

4.5.1　一意性のあるフィールドの変更への対応

本研究が対象とするようなネットワークが長期間分断するような状況では，各サーバそれぞれが異なるネットワーク内でサービスを提供する．よって，一意性のあるフィールドに変更が発生した場合には，同一のユーザの操作であったとしても，複数のユーザ ID や電話番号等が存在することとなり，正しくユーザを識別することが困難となる．したがって，一意性のあるフィールドに変更が発生した場合には，その変更履歴を作成し，統合ログファイル上及びデータベース内を最新のユーザ情報に更新する必要がある．この処理は，図 3 の B にあたる部分で実行される．

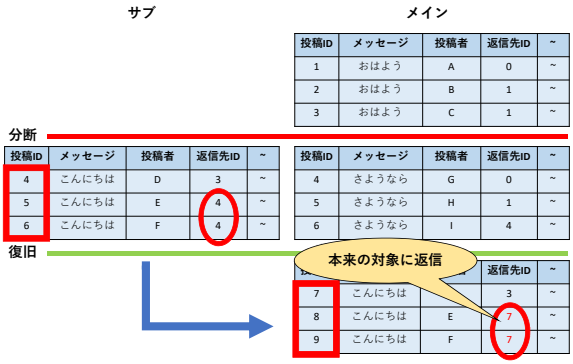

図 6　データ間の関連を維持できる統合

4.5.2　追加処理の重複への対応

まず，追加処理が重複する状況として，ログファイル上において，あるエンティティの追加処理が複数記録されている，もしくはすでに存在するエンティティのデータに対して追加処理が記録されているような場合が考えられる．これらのような場合は，すでに存在するデータに対する追加処理が発生してしまうため，データが衝突してしまう．このような追加処理の重複が発生する例を，安否確認サービスを対象とした場合の図 7 の左側にある統合ログファイルに示す．この例では，統合ログファイル上では最初に「A さんが無事である」というデータを追加したあとで，「A さんが軽傷である」という情報を追加しようとしており，二つ目の追加処理である「A さんが軽傷である」というデータがエラーとなってしまう．したがって，メインサーバ自身のデータベース内でログファイル上における追加対象のエンティティが存在した場合には，図 7 の右側にあるデータベース書き込み時の実行のように追加処理を更新処理に変更して操作を実行する必要がある．この処理は，図 3 の C にあたる部分で実行される．

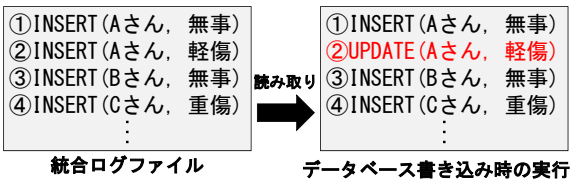

図 7　追加処理の重複への対応

4.5.3　存在しない対象の更新処理への対応

存在しない対象の更新処理が発生する状況として，ある

サーバにおける削除操作のあとに，異なるサーバで更新操作が実行されるような場合が考えられる．例として，安否確認サービスを対象とした場合における図 8 の左側にある統合ログファイルのような場合が考えられる．これは，障害発生前にメインサーバがサブサーバに「A さんが軽傷である」というデータをレプリケーションし，ネットワーク分断後に A さんが避難所等を渡り歩いたことで，メインサーバとサブサーバで実行した操作が異なったようなことが考えられる．この場合，まず A さんは，メインサーバにおいて「A さんが軽傷である」というデータを削除している．その後，サブサーバにおいて「A さんが重傷である」という情報に更新している．こうした場合，統合ログファイル上では，「A さんが軽傷である」データに対して削除操作を実行したため，次の「A さんが重傷である」という内容に更新する際に，更新対象が削除されてしまっていることになる．したがって，存在しない対象に対して更新処理を実行するとエラーになってしまうため，図 8 の右側にあるデータベース書き込み時の実行のように更新処理を追加処理に変更して実行する必要がある．この処理は，図 3 の C にあたる部分で実行される．

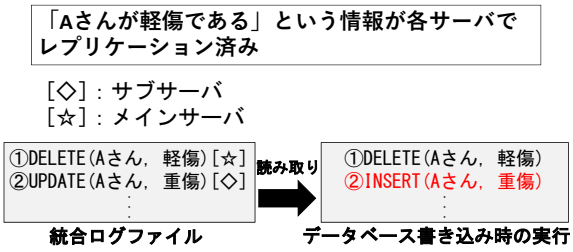

図 8　存在しない対象の更新処理への対応

5.　評価実験

本実験では，安否確認サービスおよび文献 [13] を参考に作成した簡易版ツイッターにユーザを一意に決定可能なユーザ ID を変更できる機能を追加したサービスを対象として提案手法を評価する．

5.1　実験環境

実験環境を図 9 に示す．本実験は，メインサーバ 1 台とサブサーバ 1 台で実施する．各サーバには，Raspberry PI 3 B+を使用し，ルータと LAN ケーブルには 1000BASE-T に対応したものを使用する．ユーザの操作を実行する方法として，ユーザ操作再現用の PC を 1 台用意し，メインサーバとサブサーバにアクセスする．また，ネットワークの分断と復旧を再現する方法として，ルータ 2 とその親のルータをつないでいる LAN ケーブルを抜き差しする．

5.2　ユーザの操作の定義

本実験では，ユーザの操作を以下のように定義する．

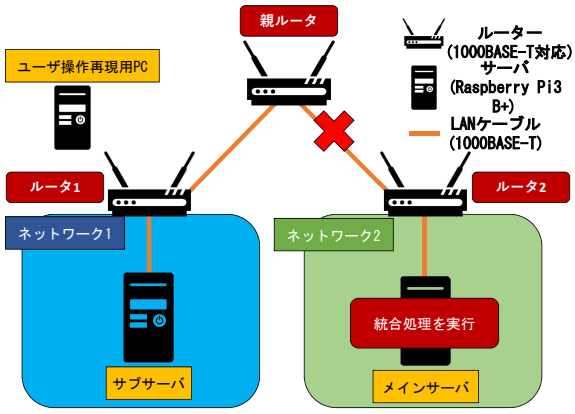

図 9　実験環境

5.2.1　安否確認サービス

安否情報の登録

各サーバに対して安否情報を登録する．新規で安否情報を登録するときや，安否情報を削除したあとに再度登録するような場合に実行する．サーバに安否情報が存在している場合には実行しない．

安否情報の更新

各サーバに対して，各ユーザが自身のユーザ ID に対応した安否情報を検索し，安否情報を更新する．サーバに安否情報が登録されていない場合は実行しない．

安否情報の削除

各サーバに対して，各ユーザが自身のユーザ ID に対応した安否情報を検索し，登録されている安否情報を削除する．サーバに安否情報が登録されていない場合は実行しない．今回は，削除フラグを使用してデータの削除を再現した．

5.2.2　簡易版ツイッター

ツイートの投稿

ユーザが，ツイートを投稿する．この操作は，ツイッターにおけるリプライではなくリプライ先のない投稿である．

ツイートへのリプライ

ユーザが，各サーバ上のツイートに対しリプライする．

ツイートの削除

ユーザが，自身がツイートしたツイートを検索し，ツイートを削除する．

ユーザ ID の変更

ユーザが，一意性のある自身のユーザ ID を変更する．

5.3　ユーザのサーバへのアクセス

本実験では，ユーザが最大 1 操作まで実行できる単位として「フェーズ」を定義し，ユーザはフェーズ単位で各サーバにアクセスし操作を実行することとした．ユーザは，そのフェーズで実行可能な操作からランダムで実行する操作を選択する．

例えば，A というユーザが存在するとし，ユーザ A はフェーズ 1 でメインサーバに情報を追加し，フェーズ 2 でメインサーバに対して情報を更新する．その後，避難所等を移動したと仮定し，フェーズ 3 でサブサーバに対して情報を追加して，フェーズ 4，フェーズ 5 ではサブサーバに対して情報の更新を実行する．また，フェーズとフェーズの間隔は数分程度空けることとし，ユーザが各サーバへ移動し操作を実行する行動を再現している．

5.4 評価方法

5.4.1 エンティティが意味的に正しい状態となっている割合

1 つ目の評価軸は，統合処理実行後のメインサーバのエンティティが意味的に正しい状態となっている割合とする．例えば，安否確認サービスを例にすると，あるユーザが「無事である」という内容から「軽傷である」という内容に更新した場合に，最新の内容である「軽傷である」という内容になっているかどうかということである．安否確認サービスについての評価では，このエンティティが意味的に正しい状態となっている割合ついて提案手法を組み込んだシステムと楽観的アプローチを組み込んだシステムそれぞれで割合を算出し比較する．簡易版ツイッターについては，提案手法を組み込んだシステムについて割合を算出し評価する．

5.4.2 データの統合にかかる処理時間

2 つ目の評価軸は，データの統合にかかる処理時間とする．提案手法では，メインサーバがサブサーバからログファイルを取得し，メインサーバ自身のログファイルとサブサーバのログファイルを統合し，データベースに書き込むまでを処理時間とする．楽観的アプローチでは，ネットワーク分断時から復旧までに変更があったユーザ分データの検証と書き込みを実行し，処理にかかった時間を計測する．この評価軸についても，安否確認サービスでは提案手法と楽観的アプローチそれぞれを組み込んだシステムで比較し，簡易版ツイッターでは，提案手法を組み込んだシステムについて評価する．

5.5 実験結果

実験結果について 5.4 節で示した 2 つの評価軸をもとに評価する．各実験では，ユーザ数を 1000 人から 10000 人の範囲で 1000 人ずつ増加させ，各ユーザ数ごとに 10 回ずつ実験を実行した．

5.5.1 安否確認サービスを対象とした提案手法と楽観的アプローチの比較

図 10 にエンティティが意味的に正しい状態となっている割合についての実験結果を示す．縦軸はメインサーバのエンティティが意味的に正しい状態となっている割合，横軸はユーザ数を示している．提案手法ではエンティティが意味的に正しい状態となっている割合がいずれのユーザ数でも 100%であったのに対し，楽観的アプローチでは，70%前後となった．

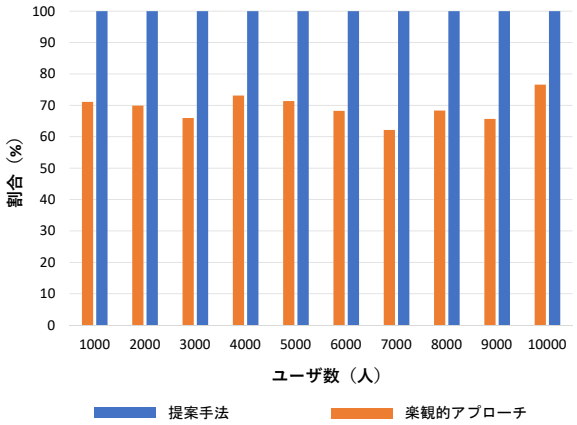

図 10 エンティティが意味的に正しい状態となっている割合（安否確認サービス）

続いて，図 11 に統合にかかる処理時間についての実験結果を示す．縦軸はデータの統合にかかる処理時間（分），横軸はユーザ数を示している．実験結果から，いずれのユーザ数に対しても提案手法より楽観的アプローチの方が処理時間が短いことが分かった．

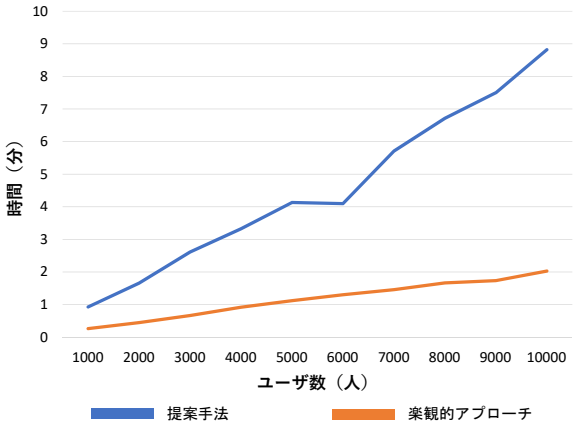

図 11 データの統合にかかる処理時間（安否確認サービス）

5.5.2 簡易版ツイッターを対象とした提案手法の評価

エンティティが意味的に正しい状態となっている割合については，安否確認サービスに適用した際と同様に 100%であった．よって，ツイート間におけるリプライの関連やユーザ ID の変更についても，ログ情報から意味的な不整合を解消しながらデータを統合できたことが分かった．

また，図 12 にデータの統合にかかる処理時間についての実験結果を示す．この結果から，簡易版ツイッターについては，ユーザ数が 10000 人の場合に統合にかかる処理時間が 9 分前後となることがわかった．

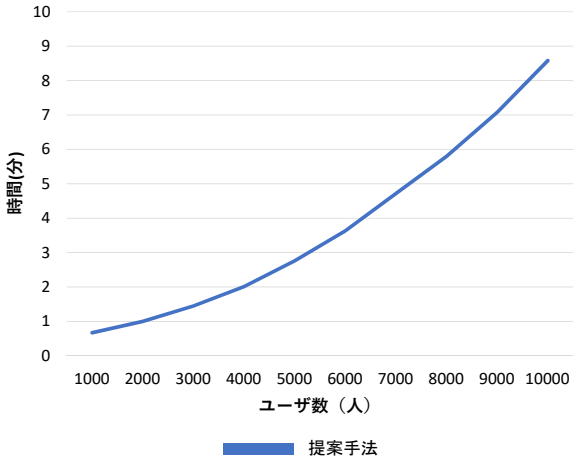

図 12 データの統合にかかる処理時間（簡易版ツイッター）

5.6 考察

最初に，安否確認サービスを対象とした提案手法と楽観的アプローチの比較結果について考察する．まず，エンティティが意味的に正しい状態となっている割合の結果では，提案手法では全サーバで行われた全操作をメインサーバで再現しているが，楽観的アプローチではサブサーバで変更があったユーザ及びレコードについて書き込みの有無を判断する検証及び書き込みを実行している．よって，楽観的アプローチでは 3.1 節で示した楽観的並行性制御の更新失敗例のような事例が少なからず発生し，最新のデータに更新できなかったと考えられる．それに対し，提案手法では全サーバで行われた全操作を時系列で実行することにより最新のデータに更新できたと考えられる．

次に，データの統合にかかる処理時間について，楽観的アプローチではサブサーバにおいてデータの変更があったエンティティのみに対し，メインサーバへの書き込み可能かどうかの検証や書き込みの実行をしているが，提案手法では全サーバにおける全操作を再現し実行している．よって，データベースへの書き込み数が楽観的アプローチよりも提案手法の方が多かったことにより，提案手法のほうが処理時間が長くなったと考えられる．

簡易版ツイッターを対象とした提案手法の実験結果では，エンティティが意味的に正しい状態となっている割合について，安否確認サービスで実験した際と同様データベースへの全操作を実行しているため，データベース内がすべて最新の状態になり，ツイート間の関連を保持したまま統合できたことが分かった．また，統合にかかる処理時間については，今回の実験における最大のユーザ数である 10000 人を対象にした場合でも 9 分前後で統合処理を完了できることが分かった．したがって，本研究の目的であるデータ間に関連があるサービスにおいて，データ間の関連を維持したままサービスを統合することが可能となったと言える．

6. おわりに

本研究では，ネットワークが長期間分断されてしまった状況から復旧する際のデータ間に関連が生じるサービスのデータ統合手法について提案し，既存機能の評価も含めてネットワークの分断の長期化と復旧を再現した評価実験を実施した．その結果，安否確認サービスに適用した際には，提案手法は楽観的アプローチと比較してデータの統合に時間がかかるが，意味的な不整合を解消しデータを統合することが可能であることが分かった．また，簡易版ツイッターに適用した際にも，データの統合に時間はかかるものの意味的な不整合を解消できることが分かった．今後の課題としてはデータの統合にかかる処理時間を減少するように手法の改善を進めながら，他のサービスにも適用して評価していきたい．

参考文献

[1] 総務省，“平成 23 年版 情報通信白書 第 1 部第 1 節 (1)”, https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h23/html/nc111100.html (参照 2021/11/22).
[2] H. T. Kung, John T. Robinson, “On optimistic methods for concurrency control”, ACM Transactions on Database Systems, Vol. 6, No. 2, pp. 213-226, 1981.
[3] S. B. Davidson, “Optimism and consistency in partitioned distributed database systems”, ACM Transactions on Database Systems (TODS), Vol. 9, No. 3, pp. 456–481, 1984.
[4] M. Herlihy, “Optimistic Concurrency Control for Abstract Data Types”, ACM SIGOPS Operating Systems Review, Vol. 21, No. 2, pp. 33-44, 1987.
[5] N. Schiper, R. Schmidt, and F. Pedone, “Optimistic algorithms for partial database replication”, Proceedings of the 10th international conference on Principles of Dis tributed Systems, pp. 81–93, 2006.
[6] Z. Xing, L. Gruenwald, S. Song, “An optimistic concurrency control algorithm for mobile ad-hoc network databases”, IDEAS ’10: Proceedings of the Fourteenth International Database Engineering & Applications Symposium, pp. 199-204, 2010.
[7] D. Sciascia, F. Pedone, and F. Junqueira, “Scalable deferred update replication”, Proceedings of the Interna tional Conference on Dependable Systems and Networks, pp. 1–12, 2012.
[8] 大坂優輝，北形元，長谷川剛，“長期的な孤立運用後にサービスを統合可能なデータ管理手法”，研究報告コンピュータセキュリティ (CSEC)，vol. 71，pp. 1-8，2021.
[9] 白鳥則郎，三石大，吉廣卓哉 他，データベース —ビッグデータ時代の基礎—，第 26 巻，未来へつなぐデジタルシリーズ，共立出版，2018.
[10] 石田政海，“基幹系向けミドルウェアのオブジェクトモデル”，Unisys 技報 : Unisys technology review. 19(3)(63)，日本ユニシス，1999.
[11] 萩原正義，“クラウドコンピューティング : 7. クラウドアプリケーションの分析と開発手法”，情報処理，Vol. 50, No. 11，pp. 1092-1098，2009.
[12] E. A. Brewer, “Towards Robust Distributed Systems”, In Proc. of PODC2000(Invited Talk)，2000.
[13] たぐちまこと，よくわかる PHP の教科書 [PHP7 対応版], pp. 236-240，マイナビ出版，2018.