情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 2　99–112 (June 2010)

ネットワーク分断に対応した分散オブジェクトライブラリ

小長谷　秋雄[†1]　宮澤　和徳[†1]
品川　高廣[†1]　加藤　和彦[†1]

近年，インターネットサービスの可用性に対する要求が高まっている．可用性を維持するには，サービスを分散して提供する手法が有効だが，データの一貫性制御に関する問題について開発者側で対応する必要がある．本論文では，分散型のインターネットサービスを構築する際に有用な分散オブジェクトライブラリを提案する．本ライブラリでは，汎用的に利用可能なデータ構造と一貫性制御に関する処理を提供しており，サービス開発者は提供したいサービスに応じて適切なデータ構造を利用することで，分散型のサービスを容易に構築することができる．本論文では，提供すべきデータ構造と各データ構造に必要となる一貫性制御について分析を行い，設計および実装を行った．また，実際のサービスに本ライブラリを組み込んだシステムの実験を行い，分断時にもサービスを提供できることを確認した．

Distributed Object Library for Tolerating Network-partition

Akio Konagaya,[†1] Kazunori Miyazawa,[†1]
Takahiro Shinagawa[†1] and Kazuhiko Kato[†1]

Recently, the demand for the availability of internet services is increasing. In order to maintain the availability, the technique of distributing services is effective. However, there is a concern of the consistency of data used in services. In this paper, we propose a distributed object library to support constructions of distributed internet services. This library provides an integrity control mechanism of each data structure so that the developers can easily construct distributed internet services by using an appropriate data structure. We have designed the data structures and its nessesary integrity control of each data structure. Also, we have confirmed the feasibility of our proposal by building the library into the existing service.

1. はじめに

近年，インターネット経由で提供されるサービスの高機能化が著しい．たとえば，SaaS（Software as a Service）やクラウドコンピューティングで提供されるサービスの中には，電子メールやグループウェアなど，今日においては生活や場合によっては企業活動にとってなくてはならないサービスも提供されている．このようなサービスでは，サービスの可用性を維持することは最も重視すべき点の 1 つである．しかし，インターネットの問題点として，専用線と比べて信頼性が低く，ネットワーク障害などでサービスの提供を継続できなくなる可能性がある．

ネットワーク障害時にサービスの可用性を維持する手法の 1 つとして，複数のホストを地理的に分散して配置する手法がある．この手法では，クライアントからあるホストに一時的に到達できなくなっても，別のホストに到達できればサービスの提供を継続することができる．しかし複数のホストを地理的に分散配置すると，ホスト間のネットワーク遅延が増大したりネットワーク分断によって情報伝達自体が不可能になったりする可能性がある[2),10)]．このため，特に複数のホストに同時に書き込みがあった場合には，ホスト間でデータの一貫性をとることが難しくなる．これは CAP 定理[1),7)] として知られている問題であり，ネットワーク分断（Network Partition）が発生しているときには，一貫性（Consistency）と可用性（Availability）を同時に満たすことは困難であるとされている．

この問題を緩和するために，Eventual Consistency[14)] などの弱い一貫性制御を用いた手法が提案されている．たとえば，Bayou[11)] や Globe[13)]，小磯らのシステム[17)] では，ネットワーク分断中にも書き込みを可能にしつつ，サービスごとに更新データのマージ処理を記述することで一貫性を保つことができる．また，Gao ら[5)] は書籍サイト向けに特化することで，自動的にマージ処理ができる分散オブジェクトを提供している．しかしこれらの手法では，個々のマージ処理の記述はアプリケーションに依存したものとなるうえ，一般のプログラマがこのような処理を記述するのは容易ではない．一方，DDS[8)] や GlobeDB[12)] などでは，アプリケーションに依存しない形で複数ホストのデータ間で自動的に一貫性を保つことができるが，ホスト間のネットワークが分断していた場合には，複数ホストから 1 つの

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

データに対して同時に行われた書き込み処理を完了することができない．

本論文では，ネットワーク分断時にもサービスの提供を可能な限り維持し続けることができるシステムの構築を目的とした，インターネット・サービス向けの分散オブジェクトライブラリを提案する．本分散オブジェクトライブラリでは，個別のアプリケーションに依存することなくネットワーク分断への対応を実現するために，インターネット・サービスで比較的よく使われるデータ構造を抽象化することで，汎用性を保ちつつ個別のデータ構造のセマンティクスを用いた自動的なマージ処理を提供する．また，ネットワーク分断中でも書き込み処理を完了できるようにするために，ホストごとにデータに対して可能な操作の種類と範囲を制約としてあらかじめ設定しておくことで，制約の範囲内であれば他のホストと通信することなく書き込み処理を完了する権限を与える．

本分散オブジェクトライブラリには，以下のような利点がある．まず，サービスを実装するプログラマが適切なオブジェクトを選択して用いることで，マージ処理を自分で記述することなくネットワーク分断への対応が可能になる．また，提供するオブジェクトの種類もインターネット・サービスで比較的よく使われるものとすることで，プログラマが適切なオブジェクトを選びやすくしている．さらに，ネットワーク分断から回復した際にマージ処理を可能にするための要件の分析を行い，ネットワーク分断中であっても可能な限り書き込み操作を完了させることができる手法を実現している．これにより，ネットワーク分断時にも限られた範囲ではあるが，継続したサービスの提供を可能としている．

本論文では，RUBiS [15] というオークションサービスをベースにして，実際に提案した分散オブジェクトライブラリを用いたインターネットサービスの実装を行った．この実装を用いて実験を行った結果，ネットワーク分断中であっても限定的にサービスの提供が継続できていることを確認した．また，ネットワーク分断から回復した後のホスト間の通信量も，数分〜数時間の一時的な分断であれば問題ない量であることを確認した．

本論文の構成は以下のとおりである．まず 2 章で関連研究との違いについて述べる．続いて，3 章で本研究で対象とするネットワーク分断について述べる．次に 4 章でネットワーク分断に対応するために本研究で提案する一貫性制御手法の方針について述べ，5 章で方針に基づいた分散オブジェクトの設計について具体的に述べる．その後，6 章で提案手法を組み込んだライブラリの実装について述べた後，7 章で実際のインターネットサービスに対して本研究で提案するライブラリを適用した結果を示し，その評価を行う．最後に，本論文のまとめを 8 章に示す．

2. 関 連 研 究

CAP 定理で示される問題に対処する研究としては，(1) ネットワーク分断から回復した後のマージ処理の記述が必要になる手法と，(2) 分散オブジェクトにより一定の条件のもとで一貫性を自動的に回復できる手法の 2 種類に大きく分けられる．以下，それぞれについて本研究との違いを述べる．

2.1 マージ処理の記述

Bayou [11] では，ネットワーク分断から回復後のマージ処理を SQL に近い文法を用いて記述することができる．また小磯らのシステム [17] では，メールサービスのマージ処理を補助スクリプトで記述している．このように個別のサービスごとにマージ処理を記述する方式は，フレームワークとしては汎用性が高いが，実際のマージ処理はサービスの内容に著しく依存したものとなる．したがって，サービス開発者が適切なマージ処理を記述しなければならず，一般のプログラマにとって容易な作業ではない．

Dynamo [4] では，ネットワーク分断中に行われた更新操作を後から適用するために，vector clock を用いている．この方式では，更新操作が衝突しない限りは，更新時刻に基づいて最新の操作を自動的に適用することができる．しかし，同一のデータに対して複数のホストで同時に書き込みを行った場合には，1 つのデータに対して複数のバージョンが発生するため，それらのマージ処理はクライアント側で個別に定義・実装しなければならない．

TACT [16] では，continuous consistency により，更新可能な操作の回数や最終同期時刻からの経過時間などを制限することで，一貫性を維持しつつ他ホストとの同期のための通信量を調整できる．更新可能な操作をアプリケーションごとに制限するという手法は，本研究で提案する更新操作の制限と似ているが，TACT で用いられている制限は更新の衝突を避けるために行われているものではなく，更新が衝突した場合のマージ処理はアプリケーションごとにあらかじめ定義しなければならない．

IceCube [9] では，あらかじめ定義した前提条件を満たすように更新操作の順序などに制約を設けることで，操作が衝突した場合には衝突がなるべく起こらないように操作の順序を自動的に並べ替えている．前提条件を定義し，これを満たすように操作を制限するという手法は，本研究で提案する手法と似ているが，IceCube で定義する前提条件や制約は更新操作順序の最適化のためのものであり，本研究で提案するような衝突を回避し，マージ操作を自動で行うために定義する制限とは目的が異なる．

2.2　分散オブジェクト

DDS[8] では，ハッシュ構造を用いた汎用的な分散オブジェクトを実現している．しかし，このシステムではクラスタ環境での運用を想定しており，クラスタ内部のネットワークは信頼性が高いことを仮定しているため，各ホスト間のネットワーク分断には対応できない．

Globe[13] や GlobeDB[12] では，広域のネットワーク上でインターネットサービスを提供するために有用な一貫性制御などの機能を組み込んだ汎用的な分散オブジェクトを提案している．しかし，Globe ではシステムの雛形が提案されているだけであり，実際にサービスとして提供する場合には一貫性制御の部分の実装を行う必要がある．また，GlobeDB では汎用的なデータベースのドライバにより自動的に一貫性制御が行えるが，primary-backup プロトコルを用いており，ネットワーク分断中には更新操作を完了することはできない．

Gao らの研究[5),6)] では，インターネットサービスの可用性を高めるために緩い一貫性制御を組み込んだ分散オブジェクトを提案している．しかし，書籍販売サイトに特化した設計になっているため，汎用的なサービスへの適用可能性は必ずしも明らかになっていない．

3.　ネットワーク分断

本章では，本論文で想定するネットワーク分断の定義と，本論文におけるネットワーク分断時への対応における目標について述べる．

3.1　想定するネットワーク分断

本論文で対象とするネットワーク分断とは，サービスを提供するホスト間のネットワークが，何らかの通信経路上のトラブルによって，一時的に著しく遅延が増大する，あるいは一時的に通信ができなくなる状態である．このとき，サービスを提供するホスト間の通信には障害が生じるが，クライアントからは地理的に分散した複数のホストのいずれかには到達できることを想定している．たとえば，日本と米国にホストが置かれていた場合に，日本と米国間の通信には一時的に障害が発生しているが，日本国内のクライアントは日本のホスト，米国内のクライアントは米国のホストにそれぞれ継続してアクセスできる状況が考えられる．

ネットワークが分断する期間としては，本研究では数秒から数分〜数十分を想定している．すなわち，その期間中サービスが完全に停止してしまうと，ユーザにとって著しく不利益が生じるが，恒久的にサービスが利用できなくなるわけではなく，いずれはネットワーク分断が回復することが期待される状況を想定している．また，必ずしもネットワーク分断の発生を検出できる必要はなく，ネットワーク分断と回復を短期間に繰り返すような不安定な状態になることも想定している．

なお，クライアントとホスト間のネットワーク通信は安定して行えることを想定しており，クライアントとホスト間の分断は想定していない．また，ストレージ障害などのホスト自身の障害は想定しておらず，ハードウェア多重化などの仕組みでホスト自身の安定性は保証されていると仮定する．

3.2　本論文におけるネットワーク分断への対応の目標

ネットワーク分断の問題は古くから知られているが[3)]，インターネットのような広域分散ネットワークの普及により現実味を帯びている．また，近年のクラウドコンピューティングにおいても，CAP 定理として示されているネットワーク分断と一貫性，可用性の問題が注目を集めている．

一般にインターネットでは，eventual consistency などのように弱い一貫性制御を許容することで，広域分散ネットワークにおける一貫性と可用性の問題に対処している．たとえば，DNS や Web の静的コンテンツなどでは，クライアントから一時的には古いデータが見えてしまうことを許容して一貫性の制約を緩めることにより，可用性の向上を図っている．しかし，DNS のようにデータを更新する権限を持つホストが一カ所である場合は問題が少ないが，多数のホストから同一のデータに対して同時に書き込むことがある場合には，依然として可用性を保つことは難しい．

本論文では，ネットワーク分断時における書き込み操作の衝突という問題に対して，ネットワーク分断が生じていないときとまったく同等の可用性を維持するのではなく，操作内容には状況によってある程度の制限が生じるものの，可能な限りネットワーク分断中でも一貫性のある形で書き込み操作を完了できる段階的機能低下（graceful degradation）を実現することを目標とする．これにより，ネットワーク分断中にサービスがまったく利用できなくなる状況を回避して全体としての可用性を向上させつつ，ネットワーク分断から回復したときにも特別な操作を行うことなく自動的にシステム全体の一貫性を回復して，シームレスにサービス利用を継続できるようにする．

4.　ネットワーク分断への対応

本章では，まずネットワーク分断中でも書き込み操作を完了できるようにするために必要な基本方針について述べる．次に，通常では衝突を回避できないような書き込みに対しても，データの意味的制約をふまえて操作の内容に制限を加えることで衝突を事前に回避する手法について概略を述べる．その後，数値とセットという 2 つの抽象的なデータ構造に対し

て，この手法を適用した場合の例について述べる．

4.1 基本方針

書き込みの衝突を回避するための方針としては，大きく分けて 3 つあげられる．これらの方針は既存の研究でもすでに示されている内容を含んでいるが，本分散オブジェクトライブラリの設計においても基本となる方針であるため，以下でそれぞれについて説明する．

データの細分化

データの単位が大きいと，複数のホストから書き込んだ際に衝突が発生する可能性が大きくなる．これは疑似共有（false share）と呼ばれている問題で[14]，本来は書き込みが衝突しないはずであるのに，書き込みの単位が大きいために衝突してしまうことがある．したがって，データの単位をなるべく小さくすることにより，書き込みの衝突が起きる可能性を小さくすることができる．ただしあまり単位が小さすぎるとデータ管理のオーバヘッドが大きくなるため，適度なバランスが必要となる．

優先順位付け

複数の書き込み操作があった場合に，それぞれの書き込みに優先順位を付けて，最も優先順位の高い操作のみを有効とすることで，書き込みの衝突を後から自動的に解決することができる．典型的には，vector clock や time stamp など時刻で順序付けすることで，最新の操作のみが有効となる手法である[14]．この場合，優先順位の低い操作は失われてしまうことになるため，それでもかまわないようなデータ構造の場合にのみ適用することができる．

マージ操作の定義

複数の書き込み操作のいずれも適用できるようにするために，書き込み操作のマージ操作を定義する手法である．たとえば数値であれば，“+1” と “+2” という 2 つの書き込みに対して，足し算という演算を定義することで “+3” にマージして両方の操作を適用することができる．マージ操作は，いずれの書き込みも有効になるという点で最も有効な手法であるが，具体的なマージ手順は対象となるデータ構造に依存するほか，必ずしもすべての操作に対して定義できるとは限らない．

4.2 意味的制約と操作制限

上記の基本方針をふまえたうえで，ネットワーク分断中でも書き込み操作を完了できる可能性をさらに高めるための手法について検討する．

あるデータ構造に対する操作をマージする場合，書き込み操作の種類によってマージ処理を定義できる場合とできない場合が生じる．たとえば数値の場合，加算や減算といった演算は，任意の順番で後から適用できるため，容易にマージ演算を定義できる．しかし値を代入する操作の場合，すべてを有効にすることはできず，いずれかの操作は失われることになる．したがって，データ構造のセマンティクスに基づいて，ネットワーク分断中には可能な操作の種類を制限してマージ操作を定義できるものだけに限定することにより，分断中でも書き込み操作を完了できる可能性を高めることができる．

また，データの性質によっても，マージ操作が定義可能な場合とそうでない場合が生じる．たとえば数値であれば，上限も下限もない無限の値をとりうる数値であれば，加算や減算という操作に対して制限を設けることなくマージ可能である．しかし，たとえば上限があった場合には，「加算」操作を無制限に認めるわけにはいかない．この場合「減算」だけを操作として認めることで，分断中の書き込み操作を完了できるようになる．

操作の種類の制限に加えて，操作の範囲についてもデータのセマンティクスに基づいて制限をかけることで分断中に書き込み操作を完了できる可能性を高めることができる．たとえば数値の例であれば，下限が 0 を下回ってはならない数値であった場合，「減算」という演算に対しては，減算する値の大きさに対して制限をかける．すなわち，すべての減算をマージして適用しても 0 を下回らない範囲であれば，その演算はマージ可能になる．

このように，データ構造のセマンティクスによる制約条件のもとで，ネットワーク分断中に可能な操作の「種類」や「範囲」に制限を設けることで，分断中に行われた書き込み操作を完了できる操作だけを取り出すことが可能になり，可用性を向上させることができる．

4.3 基本データ構造

上記の手法に基づき，「数値」と「セット」という基本的なデータ構造に適用した例を示す．

4.3.1 数値構造

数値構造は，単一の数値を表すデータ構造である．これは ID など値自身に意味のあるものではなく，何らかの量を表すデータ構造として使われることを想定している．たとえば，カウンタや在庫数など，個々を区別しないデータの総数を管理するために用いるデータ構造である．数値構造に対する演算としては，「加算」「減算」「代入」といった操作が考えられる．それぞれの操作は，引数として演算の対象となる値をとる．「乗算」や「除算」といった演算は，マージ演算の定義が容易ではないこと，量を表すデータ構造に対してはあまり用いられないことから対象としない．

数値構造の値のとりうる範囲に制約がまったくない場合は，「加算」および「減算」の操作に対しては制限はない．単純にすべてのホストでの演算内容を集めてマージして適用すればよい．一方，値の範囲に制約がある場合は，演算の種類と量に制約が生じる．たとえば，値に上限がある場合には，「加算」に対して引数となる値を適用した結果が上限を超えない

という制限をかける．一方，値に下限がある場合には，「減算」に対して引数となる値を適用した結果が下限を超えないという制限をかける．

このとき，引数の値の制限は，すべてのホストのおける操作の合計が適用されることになる．しかしネットワーク分断中には，ホスト全体において制限を超えているかどうかを判断することはできない．そこで，各ホストに対して，あらかじめ操作可能な値の範囲を分配しておくという手法を用いる．たとえば，現在の値が 100 で，0 以上という制約があった場合，100 という値を分割して各ホストに割り振る．たとえば，ホストが 10 台あった場合，それぞれのホストに均等に 10 を割り振るという方法が考えられる．この場合，ネットワーク分断中であっても，各ホストごとに 10 を超えない範囲であれば，自由に「減算」を行ってその操作を確定することができる．各ホストに割り振る量については，必要に応じてポリシとして設定することができるが，具体的な割り振り方については本論文では対象外とし，つねに均等割りを用いることとする．なお，各ホストで「加算」があった場合は，その分を減算可能な範囲として追加することも可能である．

一方，「代入」操作については，すべてのホストにおける操作を有効にする演算は定義できないため，優先順位を付けていずれか 1 つのみを選択することになる．この場合，代入する値は現在の数値の値に依存したものにならないように注意する必要がある．これは，このデータ構造から取得した値は最新のものである保証がないためである．したがって，「代入」を行う際は基本的には”write-only” とするべきである．たとえば，初期化時に最初に値を代入する場合や 1 つのホストで書いた値を他のホストに伝搬させるときなどが想定される．

4.3.2　セット構造

セット構造は，複数の要素の集合を表したデータ構造であり，基本的には同じ内容の要素が重複することも許されている構造となっている．このデータ構造は，マップなどのより複雑なデータ構造のベースとなる構造である．セット構造に対して行われる操作としては，要素の「追加」「更新」「削除」が考えられる．

「追加」操作に関しては，追加する要素の内容に意味的な制約がない場合には，特に制限を加える必要はない．マージ処理は，すべてのホストで行われた追加処理をすべて適用すればよい．まったく同じ内容の要素を追加したとしても，重複が許されているため，それぞれ別々の要素として扱われることになる．一方，要素の内容に制約がある場合は，「追加」操作に制限が加えられる．たとえば，要素の内容が重複してはならないといった制約がある場合は，複数のホストで同じ内容の要素を追加しないようにしなければならない．そこで，各ホストに対して「追加」できる要素の内容の空間を分割して割り振る．たとえば，文字列であれば，文字列のチェックサムをホスト数で割った余りがホスト番号に一意するように権限を割り振るといった方式が考えられる．この場合，「追加」できる要素の内容にかなり制限が加わるが，まったく追加できない場合に比べると可用性が向上する．

一方「削除」については，「追加」操作がない場合には特に制約を加える必要はない．同じ要素を複数のホストで同時に削除したとしても，マージ処理は単に要素を削除すればよい．しかし，同じ内容の要素を複数のホストで同時に「追加」と「削除」を行った場合，操作を適用する順番が入れ替わると，削除したはずの要素が復活したり，追加したばかりの要素が削除されたりする可能性がある．そこで，操作の対象となる要素を厳密に特定するために，要素を「追加」する際には，すべてのホストでユニークな ID を割り当てる．「削除」する際はこの ID を指定することにより，同じ内容の要素を同時に「追加」したとしても，2 つの要素を正しく区別することができる．

要素の「更新」操作については，すべてのホストにおける操作を有効にする演算は定義できないため，優先順位を付けていずれか 1 つのみを選択することになる．数値の場合と同様，「更新」操作は現在の値に基づかない “write-only” とするべきである．

4.4　アプリケーションへの適用

これまでに提案した基本方針，および基本データ構造に基づき，現実のアプリケーションへ適用する場合の指針について述べる．実際に本提案に基づいてアプリケーションを設計する場合，操作に対する何らかの制限を許容する必要があるが，制限可能な操作はアプリケーションごとに異なる．本章では，分断に対応するために認めなければならない制限について明らかにし，その後具体的な設計指針について述べるためインターネットオークション，座席予約システム，電子マネー管理システムの 3 つのシステムを対象にして，衝突する可能性のある操作をあげ，分断時にもある程度の可用性を維持するために必要な制限について具体的に述べる．

4.4.1　制限の種類

アプリケーションを分断に対応させるために操作に対して認める必要がある制限について，本研究では以下の種類に分類している．

操作の即時性制限

操作の即時性制限とは，分断中であってもすべての操作を可能としたい場合に認める制限である．この制限を用いる場合，行われた操作が処理されるのは分断が回復した後であるということを認める必要があるが，分断中でもすべてのホストで操作が実行可能になるという特徴を持つ．具体的な一貫性制御としては，行われたすべての操作に優先順位を付け，優

先順位の最も高い操作のみを有効にするという手法を用いることで一貫性を維持することができる．したがって，分断中に行った操作の中で優先順位の低い操作はすべて無効となってしまうことになる．

操作の範囲制限

操作の範囲制限とは，分断中に行われた操作は確実に処理されてほしい場合に用いる制限である．この制限を用いる場合，分断中に行える操作が限定されてしまうということを認める必要があるが，行えた操作は確実に処理することができるという特徴を持つ．具体的な一貫性制御としては，分断中に操作可能な範囲をホストごとに分割してしまい，それぞれのホストで行われた操作のマージ操作をあらかじめ定義することで，お互いに衝突することがない状況を作り出すという手法を用いることで一貫性を維持することができる．ここで，操作可能な範囲の分割とは，たとえば 4.3.2 項で述べた追加可能な要素の空間の分割などに相当し，あるいは 4.3.1 項で述べたような操作量の均等割振りなどに相当する．したがって，本来行えたはずの操作が分断中は行える範囲や量が限定されてしまったために行えなくなる可能性があることになる．

この 2 種類の制限は，データ構造のセマンティクスによって，どちらの制限もかけなくてよい場合と，少なくともどちらか一方の制限をかける必要がある場合に分類できる．どちらの制限もかけなくても一貫性が維持される場合は，たとえば数値の加算あるいは要素の追加操作しか定義されておらず，上限が決まっていないようなデータ構造などが考えられる．これは，加算や追加操作の場合，上限が決まっていなければ，ネットワーク分断のいかんにかかわらず操作を行った瞬間にデータに対して即時反映しても衝突は起こらないからである．これに対して，少なくともどちらか一方の制限をかけなければ一貫性が維持されない場合は，たとえば下限のある数値構造における減算操作あるいは重複の許されないセット構造への追加操作などが考えられる．このような操作に対しては，基本的に即時性制限や範囲制限のどちらを用いても一貫性を維持することはできるが，データ構造のセマンティクスを考慮してなるべく適切な制限を選択する必要がある．すなわち，分断中でもすべての操作が行えるようにしたいのか，それとも分断中に行われた操作は確実に処理されることを保証したいのかのどちらを優先すべきかということを，アプリケーション内の操作の性質に基づいて適切に選択することによって，ネットワーク分断の際にもアプリケーションの可用性の低下をできる限り小さく抑えることができる．

4.4.2　アプリケーション例

ここでは 4.4.1 項で述べた制限をもとに，現実のアプリケーションを分断に対応するための具体的な手法について述べる．

インターネットオークション

インターネットオークションにおいて，分断中に操作が衝突する可能性のある操作としては，新規ユーザの追加，即落札価格による落札，および同価格による入札の 3 つが考えられる．

新規ユーザの追加操作においては，システム全体で一意となるユーザ名を登録する必要があるため，分断中に別のホストで同じユーザ名を登録した場合に衝突が発生する．この操作における適切な制限としては，ユーザ側から見て，行った登録操作が後ほど無効になるよりも，あらかじめ登録可能なユーザ名が制限されていた方が，システムのセマンティクス的により自然であると考えられるため，操作の範囲制限が適切であると考えられる．具体的にはユーザ名を要素の重複がないセット構造として管理し，追加操作の制限として追加可能なユーザ名の範囲をホストごとに決めてしまう．このようにすることで，分断中に登録できるユーザ名をホストごとに制限することができ，ユーザ名の衝突を回避することができるため，登録操作が後ほど無効になるという事態は起こりえない．ただしこの場合は，本来登録可能なユーザ名が分断中は登録不可能となる制限を許容する必要がある．

即落札価格による落札操作については，落札可能な商品の個数には限りがあるため，分断中に各ホストで行われた即落札操作によって在庫数を超える即落札が行われた場合に衝突が発生する．この即落札操作については，基本的に行われた操作が後で無効となる事態は避けた方がセマンティクス的に望ましいため，操作の範囲制限が適切であると考えられる．ただし，在庫数が複数ある場合と 1 つしかない場合とで一貫性制御を分けて考える必要がある．在庫数が複数ある場合は，減算操作の制限として在庫数をもとに減算可能な数をホストごとに均等に割り振ることで，分断中に落札可能な商品個数をホストごとに制限することができる．具体的には，在庫数を数値構造によって管理し，減算操作の制限として在庫数を元に減算可能な数をホストごとに均等に割り振ることで，在庫数を超えない即落札操作が可能となる．しかし，在庫数が 1 個であった場合には，減算可能な数をホストごとに割り振ることができない．そのため，操作の範囲制限を採用した場合には即落札可能な商品をホストごとに分割するという手法を用いざるをえず，本来即落札可能な商品に対する操作が行えないという事態を引き起こす．これはオークションとしての可用性を著しく損なっているため，在庫数が 1 個であった場合には特別に操作の即時性制限を採用して，操作が行われた時刻によって優先順位を付け，最も早く行われた操作のみを有効にするという手法を用いることにする．具体的には，在庫数を数値構造によって管理し，減算操作の制限として

最も早く行われた減算操作のみを有効にすることで，1 個の商品に対して複数のユーザが同時に即落札するという事態を避けることができる．この場合，在庫数が複数ある場合には分断中に本来落札可能な個数よりも少ない個数でしか即落札操作を行えなくなり，在庫数が 1 個の場合には分断中に即落札操作を行ったとしてもタイミングによっては操作が無効になる可能性があるという制限を許容する必要がある．

同価格による入札操作においては，通常同価格による入札操作は認められていないが，分断中には他のホストの現在価格を知ることができないため，同じ価格による入札操作が行われる可能性があり衝突が発生することになる．この操作については，即落札操作のときと同様，通常入札可能な商品が分断中は入札できなくなるという事態はできる限り避けるべきであるため，操作の即時性制限を適用することが望ましいと考えられる．したがって，入札が行われた時刻によって順序付けを行い，先に行われた操作のみを有効にすることで一貫性を維持する手法を用いる．具体的には，入札価格を数値構造として管理し，入札操作を代入操作と見なして，最も早く代入された操作のみを有効にすることで実現できる．この場合，行われた入札操作が無効になる可能性があるということを許容する必要があるが，本来オークションシステムにおいて，入札という操作は入札された時間および価格によって優先順位が付けられ，最も優先順位が高い，すなわち最も早く誰よりも高額で行われた入札が有効になるという性質を持つことから，比較的オークションのセマンティクスに沿った制限であるといえる．

座席予約システム

列車や飛行機，会場などの座席予約システムにおいて，同座席の複数予約など，座席の予約という操作そのものが分断中に衝突する可能性のある操作になる．この操作は，座席の予約という性質上，基本的には後で予約が無効となってしまったという状況は許容できないため，操作の範囲制限の適用が望ましいと考えられる．このとき，座席位置を絶対的に指定した予約を可能とする場合と，ある範囲，たとえば会場の座席予約でいえば S 席や A 席といった座席の大まかな位置を指定した予約を可能とする場合とで，適切な範囲制限の種類が異なる．座席位置を絶対的に指定した予約を行いたい場合は，予約可能な座席の範囲をホストごとに分割することによって，1 つの座席に複数の予約が入るという状況を避けることができる．具体的には，各座席をセット構造の一要素として管理し，変更操作を行えるホストの範囲を分割することによって実現できる．また，座席の大まかな位置を指定した予約を行いたい場合には，予約可能な座席の数をホストごとに均等に割り振ることによって，座席全体の数を超えた予約が入るという状況を避けることができる．具体的には，大まかな位置の座席の個数を数値構造によって管理し，予約を減算操作としてホストごとに減算可能な量を制限することによって実現できる．ただし，操作の範囲制限を採用したことで，本来は空席であっても分断中はホストによって予約可能な座席が決まっているために予約できないという制限を許容する必要がある．

電子マネー管理システム

電子マネーの管理システムにおいては，銀行などの口座から自身の電子マネー口座への入金操作や，インターネットショッピングの支払い操作など，出入金などの一連の操作をトランザクションという不可分な単位で扱う必要がある．このシステムに対して提案手法を適用する場合，これまでに述べたアプリケーションとは違い，そもそも本提案手法の前提条件として弱い一貫性を認める必要があるためにトランザクション内の状態が見えてしまう可能性がある．ただし，最終的には入金および出金操作のどちらも有効になるということを保証できれば，結果として不整合が起こることはない．したがって，行われた操作が後に無効となるという状況は絶対に避けなければならないため，操作の範囲制限を採用するのが望ましいと考えられる．これをふまえて，もし，一般的なトランザクション処理を行いたい場合は，出金，入金操作の権限をいったん 1 つのホストに集めてしまい，このホスト内でトランザクション処理を行った後に権限を解放することで，トランザクション内部の状態を不可視にすることができる．具体的には，たとえばユーザごとの出金および入金を司る権限をそれぞれセット構造として管理し，トランザクション処理を行いたいホストに対して対応した出金および入金操作の権限を集めることで実現できる．また，分断中にトランザクション処理を行いたい場合は，分断中に使用可能な権限をあらかじめホストごとに限定する，すなわち操作可能な範囲を狭めることで，分断中でもトランザクション処理に必要な権限の集中を実現することができる．この場合，ユーザごとにトランザクション処理が可能なホストが異なるため，ユーザがアクセスしているホストによって出入金操作が可能な場合とそうでない場合が発生する可能性があるという制限を許容する必要がある．

5. 分散オブジェクトライブラリ

本論文で提案する分散オブジェクトライブラリは，クラス階層構造となっている．上位クラスは，抽象的なデータ構造に対して想定される操作や制約条件を定義できる抽象部分となっており，下位クラスは，上位クラスをもとにして実際にサービスを開発する際に有用な具象部分となっている．以下では，それぞれについて説明する．

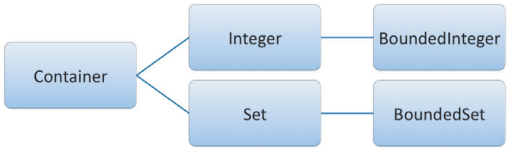

図 1　抽象クラス
Fig. 1　Abstract classes.

5.1 抽 象 部 分

本ライブラリにおける抽象部分のクラス構造を**図 1** に示す．抽象部分のクラスは開発者側が直接使用することは想定しておらず，あくまでデータ構造を分類する指標として定義している．

Container クラス

Container クラスは，すべてのクラスの基底となるクラスであり，すべての下位クラスで必要となる機能を持っている．たとえば，すべてのホストの情報を同期した最終更新時刻を保持する．これにより，完全に同期がとれている情報と同期が確認できていない情報とを区別することができ，用途に応じて使い分けることができる．

Integer クラス

単一の数値を扱うクラスである．このクラスには，以下の 4 つのメソッドがある．

- `add`: 値の加算
- `sub`: 値の減算
- `set`: 値の設定
- `get`: 値の取得

Integer クラスでは，上限や下限がないため，操作に対して制限はない．分断中に行われた操作が衝突した場合の解消手段としては，以下のマージ操作を選択できる．

- `sum`: すべてのホストで行われた操作を足し合わせる．
- `max`: 指定した値が最も大きな操作だけを有効にする．
- `min`: 指定した値が最も小さな操作だけを有効にする．

`add` と `sub` および `get` メソッドのみ使用する場合は，`sum` 操作を選択できる．この場合，値の設定はオブジェクトの作成時にのみ行うことができる．`set` メソッドを使用する場合は，設定する値もしくは操作を行ったときの時刻に基づいて，その値が最大（`max`）もしくは最小（`min`）のものを有効とする．なお，この場合の時刻は NTP などで緩やかに同期された時刻であり，因果関係を保証するものではないことに注意が必要である．

BoundedInteger クラス

Integer クラスと同じく単一の数値を扱うクラスであるが，制約条件として上限や下限を設定することができる．

- `setUpperBound`: 上限の設定
- `setLowerBound`: 下限の設定

上限を設定すると，`add` 操作に制限がかかる．すなわち，`add` できる値の範囲は，すべてのホストの合計値が，上限値と現在の値との差を超える操作はできなくなる．下限を設定すると，`sub` 操作に対して，同様の制限がかかる．

ホストごとに実際に操作できる値の範囲は，別途用意するポリシモジュールによって決定する．現在のポリシモジュールは，値の範囲をホストの数で等分するポリシのみをサポートしている．ホスト間で操作できる値の範囲に偏りが生じないようにするために，ネットワークが分断していない間は必要に応じて再分配を行う．

操作に制限を設けることにより，数値の範囲に上限や下限などの制約がある場合でも，ネットワーク分断中に各ホストごとに許された範囲内で add や sub 操作を完了させることが可能になる．

Set クラス

複数の要素が集まった集合を扱うためのクラスである．このクラスでは，以下の 4 つのメソッドが使用可能である

- `create`: 要素の追加
- `delete`: 要素の削除
- `modify`: 要素の内容の変更
- `enumerate`: 要素の内容の取得

Set クラスでは，いずれの操作も制限はない．分断中に行われた操作が衝突した場合の解消手段としては，以下のマージ処理から選択できる．

- `sum`: すべてのホストで行われた操作を足し合わせる．
- `max`: 指定した値が最も大きな操作だけを有効にする．
- `min`: 指定した値が最も小さな操作だけを有効にする．

`create` と `delete` および `enumerate` メソッドのみ使用する場合は，`sum` 操作を選択できる．この場合，同じ内容の要素を複数のホストで `create` すると，重複して要素が作成される．`modify` メソッドを使用する場合は，設定する値もしくは操作を行った時の時刻に基づ

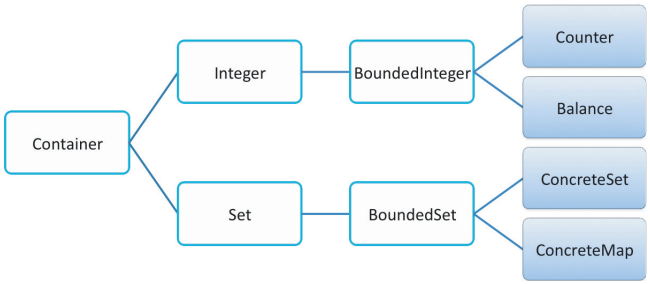

図 2　具象クラス（色付き部分）
Fig. 2　Concrete classes (colored).

いて，その値が最大（max）もしくは最小（min）のものを有効とする．

BoundedSet クラス

Set クラスと同じくデータの集合を扱うクラスであるが，データの重複がないという制限を設定できる．

- `forbitDuplicateItems()`: 重複の禁止

重複を禁止するという制約を設けると，`create` 操作に制限がかかる．すなわち `create` する要素の内容がすでに存在している可能性がある場合，その操作は失敗する．

ホストごとに `create` できる要素の内容の範囲は，BoundedInteger と同様にポリシモジュールによって決定する．現在のポリシモジュールは，要素の内容を文字列と仮定し，文字列の先頭文字によってホストごとに `create` する権限を振り分けている．

5.2　具 象 部 分

本ライブラリにおける具象部分のクラス構造を**図 2** に示す．具象部分では，抽象部分のクラスをもとにして，インターネットサービスにおいて有用であると考えられるデータ構造に特化してクラスを実装している．サービス開発者は，サービスに適合するように具象クラスを選択し使用することにより，ネットワーク分断に対応したサービスを構築することができる．

Counter クラス　単調増加する数値を扱うためのクラスである．BoundedInteger を継承しており，下限として 0 が，操作として `add` のみが定義されている．マージ操作としては `sum` を用いる．用途としては，Web ページのアクセスカウンタなどが考えられる．

Balance クラス　残数を表す数値を扱うためのクラスである．BoundedInteger を継承しており，下限を任意の値に定義できるほか，`add` に加えて `sub` を提供している．したがって，`sub` 操作では，現在の値に応じて操作の範囲に制限が加わる．マージ操作は `sum` を用いる．用途としては，商品の在庫数や口座の残高，ポイントの残高などが考えられる．

ConcreteSet クラス　一般的な集合を扱うためのクラスである．Set を継承しているが，要素の値の重複は許可されていない．したがって，`create` および `modify` 操作に制限が生じる．このクラスは直接使うのではなく，継承した下位クラスを使うことを想定している．

ConcreteMap クラス　一般的な Map 構造を扱うためのクラスである．BoundedSet クラスを継承しており，キーと値のペアをデータとして格納した集合である．キーの重複は許可されていないため，`create` 操作に制限が生じる．実際に使用する際は，ConcreteMap クラスから派生したクラスの中で適合するものを選択する．

6. 実　　装

6.1　システム構成

本論文の分散オブジェクトライブラリでサービスを構成した場合のシステムの全体像を**図 3** に示す．本システムでは，クライアントからの操作を受け付けるサービス提供レイヤ，分散オブジェクトライブラリ，ストレージによって構成されている．これらはホスト内に格納されており，地理的に離れた場所に複数台分散させて設置する．

サービス提供レイヤは，個別のインターネットサービスを提供するためのソフトウェアで，たとえば Web サービスであれば，HTTP サーバとアプリケーションサーバなどを用いて構成する．サービス提供レイヤでは，リクエストを処理する為に分散オブジェクトに格納された情報を用いて必要となる情報をクライアントに提供する．分散オブジェクトの実装は隠蔽されており，サービス開発者は内部の実装についてすべて知っている必要はない．

分散オブジェクトライブラリは，分散オブジェクトおよび管理マネージャで構成されている．分散オブジェクトは，サービス提供レイヤおよび管理マネージャから非同期にアクセスされ，その内部状態が変更される．また，管理マネージャは，ストレージへの永続化，他ホストとの更新情報伝達機能を備えており，これらは各オブジェクトのセマンティクスによって設定されたポリシに沿って実行される．

この分散オブジェクトは，サービス開発者が提供したいシステムに適合したオブジェクトを選択できるように，オブジェクトの表すデータ構造の種類に応じて分類されている．また，データ構造に適した一貫性制御をサービス開発者側で実装するコストを削減するため，

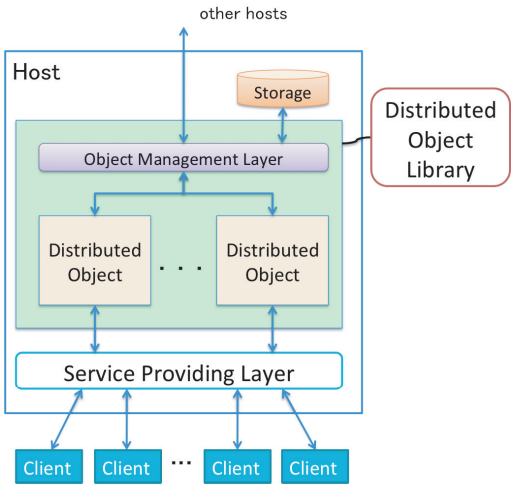

図 3　システム構成
Fig. 3　System architecture.

あらかじめオブジェクト内部に一貫性制御のための処理とそれを司るポリシが定義されており，サービス開発者は一貫性に関するポリシを設定するだけでホスト間のオブジェクトの同期処理が可能となる．

6.1.1　オブジェクト管理部分

オブジェクト管理部分では，主に「他ホストとの更新情報伝達」と「ストレージへの永続化」の 2 つの機能を備えている．これらの機能をいつ使用するかはオブジェクト内のポリシで決められており，定義されたポリシに従って情報の送受信や永続化が行われる．

他ホストへ更新情報を送るために，本システムではデータ構造に対して行われた操作をログとして保存している．このログに登録された各情報には，システム全体でユニークとなる ID と登録日時が定義されており，衝突回避操作を行う際に他の情報と区別することができる．このログを他ホストへ送信することにより，システム全体の同期を行う．この更新情報を送信するタイミングは，分散オブジェクトに定義されたポリシで決められており，サービス開発者はこのポリシを設定することで同期の頻度を調整することができる．

ストレージへの永続化は，通常データ構造へ何らかの操作が行われたときと，他ホストへ更新情報を送信するときに行われる．具体的な永続化手法としては，本研究ではオブジェクトのシリアライズを用いてファイルあるいはデータベースへ保存する手法を用いている．

6.2　例 外 処 理

本ライブラリでは，ネットワークが分断状態になった場合，例外を発行することでサービス提供レイヤにネットワーク分断を通知する機構を備えている．サービス開発者側は，ライブラリから発行された例外をもとにネットワーク分断を認識し，サービスの動作を変更させることができる．

本システムにおいて，分断中に 4.4.1 項で述べた衝突する可能性のある操作を行ったときに，例外が発行される可能性がある．具体的には，一貫性を維持するために操作可能な量や範囲を制限しているときに，決められた量や範囲を超える操作を行った場合に例外が発生する．ただし，操作の優先順位を付ける手法を用いていた場合には，分断から回復しなければ操作に優先順位を付けることができないため，制限のいかんにかかわらず例外は発行されないが，分断が回復した後に操作に付けられた優先順位が低ければ無効となってしまうという点に注意が必要である．ここで，現在の実装では，優先順位によって操作が無効とされた場合に，サービス提供レイヤへ通知を行う手段はないが，無効となる操作が発生した場合にあらかじめ登録しておいたイベントハンドラを呼ぶという手法を用いることで，サービス提供レイヤへ操作の無効を通知することはできると考えられる．

例外が発生すると，そのとき行われていた操作はただちに停止し，サービス提供レイヤに処理が戻る．このとき，サービス提供レイヤでネットワーク分断に対応した処置を講ずることができる．具体的には，分断対応のための操作制限によって処理が停止した旨，分断が回復した後には再度操作が可能になる可能性がある旨などをサービス利用者側に通知する必要があると考えられる．ただしこのとき，ライブラリ内のデータの一貫性維持に関して，サービス開発者側が何らかの操作を行う必要はない．これは，そもそもこの例外がデータの一貫性を崩さないように定義された制限によって発生した例外であるためである．したがって，サービス開発者が最初の設計の際に，ライブラリを利用してサービスのセマンティクスに合った制限を定義していれば，データの一貫性はライブラリ内で自動的に維持される．

6.3　パフォーマンスへの影響

本ライブラリを用いる場合，集中型のシステムと比較すると，各ホストでデータを最新の状態に保つために，更新を行う際につねに一貫性を維持するための処理を行わなければならないため，この部分が性能的なボトルネックとなる．具体的な処理としては，本ライブラリでは他のホストから送信されてきた操作のログを集めて，優先順位を付ける，あるいはログのマージを行うことによって一貫性を維持している．したがって，このログのサイズが大

きいほど優先順位付けやマージ処理にかかる時間も長くなり，データの更新に時間がかかってしまうことになる．ログのサイズが大きくなる状況としては，たとえば長期間ネットワークが分断された後に回復し，各ホストから分断中に行われた操作のログがいっせいに送信されてくるといった状況が想定される．ただし本ライブラリでは，基本的にネットワークが分断される時間は最長で数十分程度であることを想定しているため，もともとライブラリ内で扱っているオブジェクトの量が大きくなければログとして伝達しなければならないデータサイズも，パフォーマンスにそれほど大きな影響は与えないと考えられる．

7. 評　　価

本章では，提案する分散オブジェクトシステムの有効性を検証するために，実際のインターネットサービスを本システム上に移植した結果について述べる．本論文では，オープンソースのインターネットオークション・サービスである RUBiS を用いた．RUBiS [15] は，eBay というオークションサイトをより簡略化したものをもとにして設計されており，オークションサービスの性能評価を目的としたアプリケーションである．簡略化はされているものの，一般的なオークションサービスで用いられている機能はほぼ搭載されている．

7.1 サービスへの対応

RUBiS では，内部で扱う情報を**表 1** で示す 7 つに分類している．

表 1 の 7 つの情報をそれぞれ分散オブジェクトとするために，本分散オブジェクトライブラリで提供しているクラスとの対応関係を調べた．実際の対応関係を**表 2** に示す．

ここでは表 2 のうち，複雑な対応関係を定義する必要のあるユーザ情報と商品情報について，詳細に説明する．

ユーザ情報　ユーザ情報では，ユーザ ID を主キーとして区別しており，1 度作成された情報をユーザ自身が変更することはできない．ただし，ユーザの出品者としての評価が数値で記録されており，落札者からの評価によって出品者評価が変化する．

したがって，BoundedSet を継承した ConcreteMap クラスを用いる．このクラスに許可する操作としては `create` 操作のみを定義し，変更される可能性のある出品者評価の数値は，上限下限のない Balance クラスを用いる．

商品情報　商品情報では出品されている商品の情報が格納されており，1 度登録された情報が出品者によって変更されることはない．ただし，入札や即落札によって現在価格や出品個数が変化する．

したがって，BoundedSet を継承した ConcreteSet クラスを用いる．このクラスでは `create` 操作のみを定義する．変更される可能性のある入札数および現在価格は Counter クラスを，出品個数については下限として 0 を定義した Balance クラスを用いる．入札数で用いる Counter クラスでは加算操作のみを定義し，上限下限は設けない．したがって更新の衝突は起こらず，各ホストで行われた入札操作を集めて，その個数を加算すればよい．現在価格で用いる Counter クラスでは set 操作のみを定義し，操作が衝突した場合には値の大きい方，すなわち金額の大きい方を有効とする．出品個数で用いる Balance クラスでは sub 操作のみを定義し，0 より小さくなるような操作が行われた場合はタイムスタンプが遅い方の操作を無効とすることで衝突を回避する．

7.2 実　　験

本節では，ネットワーク分断時にも継続的にサービスを提供できるかどうか確認するための実験について述べる．実験には 3 台のサーバマシンと 3 台のクライアントマシンを用い

表 1　RUBiS で取り扱う情報
Table 1　Information used by RUBiS.

内部情報	説明
ユーザ情報	ユーザの名前，ユーザ ID，パスワード，居住地域，出品者としての評価などの情報
商品情報	商品名，出品個数，現在価格，即落札価格，出品者，落札日時などの情報
入札情報	入札者，入札商品，入札価格，入札個数，入札日時などの情報
即落札情報	即落札者，即落札商品，即落札個数，即落札日時などの情報
コメント情報	出品者に対するコメントと出品者への評価に関する情報
カテゴリ情報	出品商品のジャンル情報
地域情報	ユーザの居住地域情報

表 2　RUBiS と分散オブジェクトの関係
Table 2　The relationship between RUBiS and distributed object.

情報	使用オブジェクト（情報）	方向の制限	量の制限
ユーザ情報	ConcreteMap（全体）	追加操作のみ	ユーザ名の頭文字による制限
	Balance（評価）	加減算操作のみ	なし
商品情報	ConcreteSet（全体）	追加操作のみ	なし
	Counter（入札数）	加算操作のみ	なし
	Counter（現在価格）	代入操作のみ	初期価格以上
	Balance（在庫数）	減算操作のみ	0 以上
入札情報	ConcreteSet（全体）	追加操作のみ	なし
即落札情報	ConcreteSet（全体）	追加操作のみ	なし
コメント情報	ConcreteSet（全体）	追加操作のみ	なし
カテゴリ情報	ConcreteSet（全体）	操作禁止	なし
地域情報	ConcreteSet（全体）	操作禁止	なし

表 3　マシン環境
Table 3　Machine environment.

CPU	Xeon 3.6 GHz
Memory	2 GB
NIC	Intel Corp. 82545 GB Gigabit Ethernet
OS	Linux 2.6.18（CentOS 5.3）

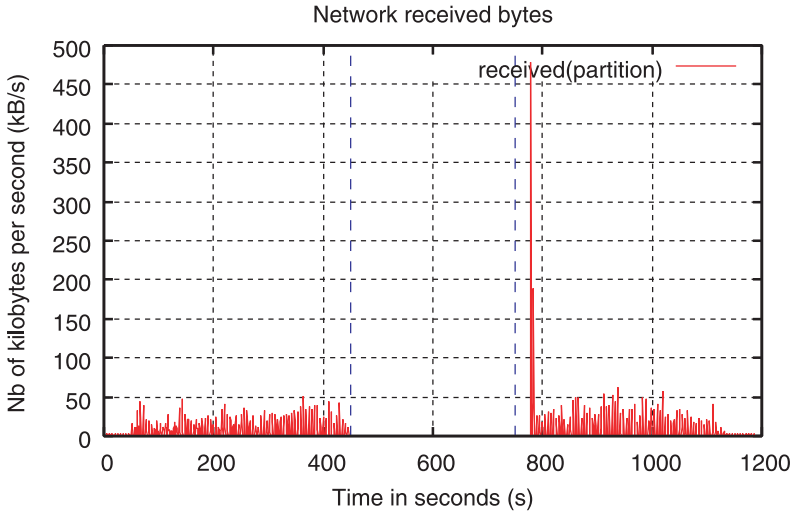

図 4　ネットワーク受信量（サーバ側）
Fig. 4　Network received bytes (Server side).

て，サーバマシンには前述した RUBiS を動作させた．クライアントは 240 ユーザの接続をエミュレートして，3 台のサーバに対して 15 分間接続を行った．このとき，15 分間のうちの 5 分間は，1 対のサーバおよびクライアントマシンを他の 2 対のマシンからネットワーク的に切断し，分断中もクライアントからの接続を処理できるかどうか調べた．

実験に用いたマシンのスペックは**表 3** のとおりである．サーバ間の同期タイミングは，5 秒おきに通信を行うように設定した．各クライアントマシンからそれぞれ別々のサーバに対して，RUBiS に搭載されているクライアントエミュレータを用いて接続を行った．

ネットワーク通信量を測定した実験結果について，**図 4** および**図 5** で示す．図 4 は，サーバ側のうちの 1 台の同期メッセージの受信量を示しており，図 5 はこのサーバへ接続して

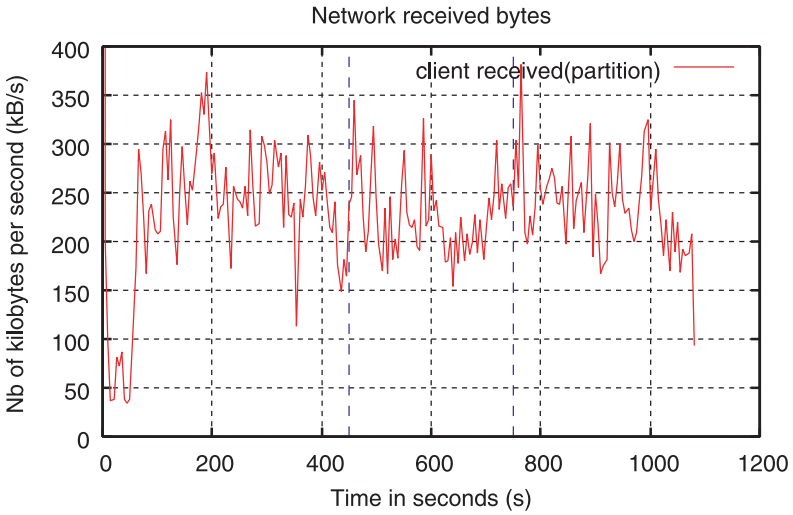

図 5　ネットワーク受信量（クライアント側）
Fig. 5　Network received bytes (Client side).

いるクライアントのネットワーク受信量を示している．図中で破線で囲まれた部分は，ネットワークが分断されている期間を示している．

図 4 では，ネットワークが分断した瞬間に受信量が大きく低下している．これは分断のためにサーバ間のメッセージ送信が不可能になったためである．一方図 5 では，多少ばらつきはあるが，ネットワーク分断の前後でネットワークの通信量に大きな変化はない．これらの結果は，サーバ間のネットワークが分断されていても，クライアントからのリクエストを処理できていることを示している．

ただし，分断中はクライアントからの操作が制限されているため，本来行えるはずの操作が許可されずにエラーとなる操作が発生している．今回の実験では，ユーザの追加操作において，分断がない場合の実験では 173 リクエストに対してエラー回数は 0 であったが，分断がある場合の実験では 172 リクエストに対して 32 回のエラーが発生している．

なお，クライアント側のネットワーク受信量のばらつきの原因としては，クライアントのリクエスト内容によって，サーバ側でのリクエスト処理にかかる時間が異なるためであると考えられる．すなわち，処理時間が長いリクエストがクライアントから多数送られた場合，その間レスポンスが遅れるため，クライアント側の受信量が低下してしまうと考えら

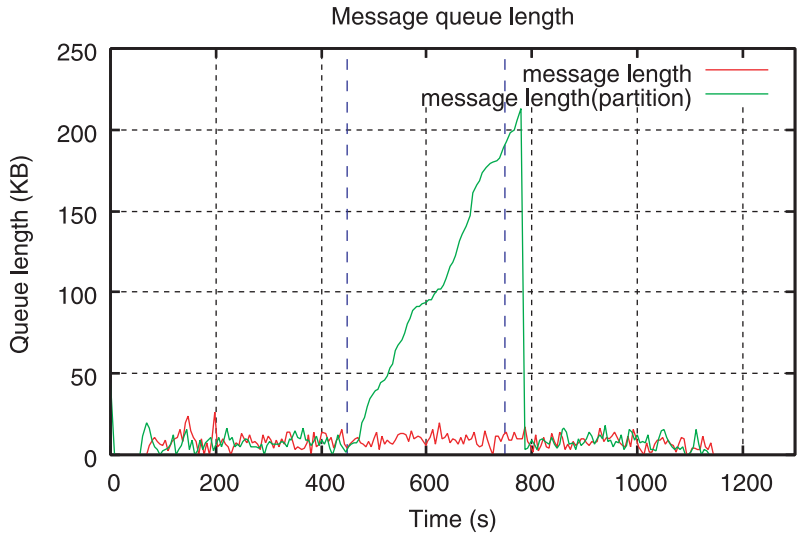

図 6　メッセージサイズ
Fig. 6　Message size.

れる．このリクエスト処理時間の違いについては，通常の RUBiS であればリクエストの内容によらずほぼ一定であるため，本ライブラリを使用したことによって生じたものである．ただし，本研究においてはネットワーク分断中でも継続的にサービスを提供できるということを目的としているため，レスポンスタイムの向上などパフォーマンスに関して考慮した実装を行うことで改善できると考えている．

図 6 に，上記のサーバにおいて他ホストに対して送信したメッセージサイズの時間的変化を示す．破線で囲まれた部分は，ネットワーク分断が起きている期間である．ネットワーク分断中は，メッセージの送信が平均して 1 KB/s 程度で行われているが，そのメッセージは受信されないため，徐々にメッセージがキューに溜まっている様子が分かる．

分断回復後は，分断中に行われた操作を他のホストに向けて送信するため，同期メッセージの受信量が大幅に増加している．この分断中のメッセージキューのサイズは，本実験では 5 分間で 300 KB 程度であった．これは現在の一般的なネットワーク環境においてはメッセージ送信に支障を来すほどの増加量ではない．仮に 50 分分断した場合でも 3 MB 程度と予想されるため，数分〜数十分程度の分断であれば，メッセージキューのサイズの増加はそれほど問題にならないと考えられる．

8. まとめと今後の予定

本論文では，インターネット経由でサービスを提供する際に，サーバのホスト間でネットワーク分断が発生した場合でも，提供しているサービスの停止を回避することを目的として，ネットワーク分断から回復した際に一貫性回復を自動的に行うための分散オブジェクトライブラリを提案した．多くのインターネットサービスで用いられる基本的なデータ構造を提供することで，サービス開発者が適切な分散オブジェクトを選択することにより，ネットワーク分断を考慮したサービスを容易に構築することができる．また，各分散オブジェクトでは，データ構造のセマンティクスを考慮した書き込み操作のマージ処理を定義し，操作の範囲や量にあらかじめ制限を加えることによって，分断中でも各ホストで一部の書き込み操作を完了できるようにした．また，本研究で提案したライブラリの設計をもとに実際のシステムへ組み込み，その動作を評価することで，本論文で提案した一貫性に対する解決策がある程度有効であることを示した．

今後の課題としては，より多くのインターネットサービスに適用できるようにするために，さらなるデータ構造の分類やその一貫性制御の分析を進め，より汎用的に用いることができるライブラリとして構築する点があげられる．また，本論文では RUBiS というオークションサービスに対して提案したライブラリを適用したが，その他の種類のサービスにも同様に適用できるかを，実際に本ライブラリを組み込んで有効性を確認することがあげられる．

参　考　文　献

1) Brewer, E.A.: Towards robust distributed systems, *PODC '00* (Invited talk) (2000).
2) Dahlin, M., Chandra, B.B.V., Gao, L. and Nayate, A.: End-to-end WAN service availability, *IEEE/ACM Trans.Netw.*, Vol.11, No.2, pp.300–313 (2003).
3) Davidson, S.B., Garcia-Molina, H. and Skeen, D.: Consistency in a partitioned network: A survey, *ACM Comput. Surv.*, Vol.17, No.3, pp.341–370 (1985).
4) DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: Amazon's highly available key-value store, *Proc. 21st Symposium on Operating Systems Principles*, ACM, pp.205–220 (2007).
5) Gao, L., Dahlin, M., Nayate, A., Zheng, J. and Iyengar, A.: Improving Availability and Performance with Application-Specific Data Replication, *IEEE Trans. Knowl.*

Data Eng., Vol.17, No.1, pp.106–120 (2005).
6) Gao, L., Dahlin, M., Nayate, A., Zheng, J. and Iyengar, A.: Application specific data replication for edge services, *Proc. 12th International Conference on World Wide Web*, pp.449–460 (2003).
7) Gilbert, S. and Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, *SIGACT News*, Vol.33, No.2, pp.51–59 (2002).
8) Gribble, S.D., Brewer, E.A., Hellerstein, J.M. and Culler, D.: Scalable, Distributed Data Structures for Internet Service Construction, *Proc. OSDI 2000*, pp.319–332 (2000).
9) Kermarrec, A.-M., Rowstron, A., Shapiro, M. and Druschel, P.: The IceCube approach to the reconciliation of divergent replicas, *Proc. 20th ACM Symposium on Principles of Distributed Computing*, pp.210–218 (2001).
10) Paxson, V.: End-to-end routing behavior in the Internet, *Proc. SIGCOMM '96*, pp.25–38 (1996).
11) Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M. and Demers, A.J.: Flexible Update Propagation for Weakly Consistent Replication, *Proc. 16th ACM Symposium on Operating Systems Principles* (*SOSP '97*), pp.288–301 (1997).
12) Sivasubramanian, S., Alonso, G., Pierre, G. and van Steen, M.: GlobeDB: Autonomic Data Replication for Web Applications, *Proc. 14th International Conference on World Wide Web* (*WWW '05*), pp.33–42 (2005).
13) van Steen, M., Homburg, P. and Tanenbaum, A.S.: Globe: A Wide-Area Distributed System, *IEEE Concurrency*, Vol.7, No.1, pp.70–78 (1999).
14) Tanenbaum, A.S. and van Steen, M.: *Distributed Systems: Principles and Paradigms, 2nd Edition*, chapter7, Prentice-Hall, Inc. (2006).
15) Team, R.: RUBiS. http://rubis.ow2.org/index.html
16) Yu, H. and Vahdat, A.: Design and evaluation of a conit-based continuous consistency model for replicated services, *ACM Trans. Comput. Syst.*, Vol.20, No.3, pp.239–282 (2002).
17) 小磯知之，阿部洋丈，池嶋　俊，石川宗寿，ポッターリチャード，加藤和彦：サステーナブルサービスための基盤ツールキットの設計，情報処理学会論文誌コンピューティングシステム（ACS），Vol.48, No.SIG3, pp.13–26 (2007).

(平成 21 年 10 月 2 日受付)
(平成 22 年 2 月 16 日採録)

小長谷秋雄（学生会員）
1985 年生．2010 年筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程修了，修士（工学）．分散システムに興味を持つ．

宮澤　和徳
1986 年生．筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程所属．分散システムに興味を持つ．

品川　高廣（正会員）
1974 年生．2003 年東京大学大学院理学系研究科情報科学専攻博士課程修了，博士（理学）取得，東京農工大学助手就任．2007 年より筑波大学大学院システム情報工学研究科講師．オペレーティングシステムや仮想マシンモニタ等のシステムソフトウェアに興味を持つ．平成 11 年度情報処理学会論文賞，平成 14 年度山下記念研究賞受賞．

加藤　和彦（正会員）
1962 年生．1985 年筑波大学第三学群情報学類卒業．1989 年東京大学大学院理学系研究科博士課程中退．1992 年博士（理学）（東京大学）．1989 年東京大学理学部情報科学科助手，1993 年筑波大学電子・情報工学系講師，1996 年同助教授，2004 年より筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻教授．オペレーティングシステム，分散システム，仮想計算環境，セキュリティに興味を持つ．1990 年情報処理学会学術奨励賞，1992 年同研究賞，2005 年同論文賞，2004 年日本ソフトウェア科学会論文賞各受賞．