Vol.2011-OS-116 No.11
2011/1/25

インターネットサービス基盤の構築に向けたパッシブ複製の自己安定化手法

大和崎　啓[†1]　長谷部　浩二[†1]
杉木章義[†1]　加藤和彦[†1]

インターネットサービスの可用性を向上させるための方法として複製手法が挙げられる．この手法ではしばしば合意アルゴリズムが用いられるが，アルゴリズムの性質により，最大同時故障台数などに関する制限がある．本論文では，多数の計算機の同時故障にも対処可能なインターネットサービス基盤の構築を目的とする．そのために，Dolev らによって提案された自己安定性を持つ合意アルゴリズムを基にしたパッシブ複製手法を提案する．

Self-Stabilizing Passive Replication for Internet Service Platforms

Kei Yamatozaki,[†1] Koji Hasebe,[†1] Akiyoshi Sugiki[†1] and Kazuhiko Kato[†1]

Replication is a well-known technique to improve the availability of Internet services. In implementing this technique, a consensus algorithm is often used to achieve consistency of replication. However, in order to guarantee the legal behavior of traditional consensus algorithms, it is required to satisfy a set of conditions, such as the maximum number of simultaneous node failures. The objective of this research is to develop an Internet service platform with high availability based on the passive replication technique. In particular, our proposed system aims to provide services despite simultaneous failure of the majority of nodes. To implement such a platform, we use the self-stabilizing consensus algorithm introduced by Dolev et al.

1. はじめに

近年，ブログや SNS，ネットオークションなどのようなインターネットサービスが広く普及している．こうしたインターネットサービスに求められる重要な性質の一つとして可用性があり，これを向上させるための方法として，複数の計算機にサービスを複製する手法が知られている．この手法を用いることによって，一部の計算機が故障しても他の計算機が故障した計算機を代替してサービスを継続することができる．

計算機を複製する手法には，主にアクティブ複製（active replication）[11] とパッシブ複製（passive replication）[1] が知られている．これらの複製手法を実現するには，すべての計算機のサービスの状態（サービス提供に必要なアプリケーションやデータなどの状態）を一致させる問題，すなわちサービスの一貫性の問題を解決する必要がある．こうしたサービスの一貫性は，分散システムにおいてすべての計算機が共通の値を出力する問題，すなわち合意問題を解決することによって実現することができる．この合意問題を解くための方法として，合意アルゴリズムが広く知られており，複製手法におけるサービスの状態を計算機間で一致させるためにしばしば用いられている．またこれに加えて，パッシブ複製ではプライマリ計算機と呼ばれる 1 台の計算機がサービスの提供を行うため，どの計算機がプライマリ計算機であるかという情報を計算機間で一致させる必要があるが，これも合意アルゴリズムを用いることで解決できる．

現在広く用いられている Paxos[8] などの合意アルゴリズムは，過半数以上の計算機から承認を得ることによって合意を形成する．そのため，過半数以上の計算機が常に正常でなければならず，このような制約条件が満たされない状態に陥ると，合意に至らずにアルゴリズムが停止してしまう．そのため，インターネットサービスを提供するシステムにこうした合意アルゴリズムを用いた場合，制約条件を満たせなくなる障害が発生すると，サービスの提供自体が停止してしまうという問題がある．

そこで本研究では，複製手法を基に，従来用いられてきた合意アルゴリズムを自己安定性（self-stabilization）[4),5] に基づく合意アルゴリズムに変更することで，自己安定的なインターネットサービス基盤の構築を行う．自己安定性とは，独立した計算機が相互に通信を行う分散環境において，障害によりシステムが正常な振る舞いから一時的に逸脱しても，自

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

動的に正常な振る舞いへと復帰する性質であり，Dijkstra によって初めて提唱された概念である．また，自己安定性の概念に基づく合意問題は Dolev らによって定義され，それを解決する合意アルゴリズムも Dolev らによって提案されている[6]．この自己安定性に基づく合意アルゴリズムは，Paxos などの従来の合意アルゴリズムと違い，合意の形成に過半数の計算機からの承認を必要としない．そのため，この合意アルゴリズムを用いることによって，本システムは，大規模な災害などの要因で多数の計算機が故障する障害のように，従来手法ではサービスが停止する障害にも対処することができる．

自己安定性の性質から，本研究で提案するシステムは一時的に正常な振る舞いから逸脱することを許すシステムとなる．したがって，常に正常な振る舞いに保たれていなければならないサービスでは，同一のサービスを提供する計算機間において一貫性の問題が発生する可能性がある．しかし，掲示板やブログサイトといった必ずしも厳密な一貫性を必要としないサービスに対しては，障害によっていかなる状態になったとしても，いずれは正常な振る舞いへと復帰することが保証されているため，高い可用性を持つシステムを構築することができる．特に，災害掲示板のように自然災害によって多くの計算機が失われても，サービスの継続が望まれるサービスにおいて有用なシステムである．

本論文の構成は以下の通りである．第 2 章では，本システムで用いる自己安定性に基づく合意アルゴリズムについて説明する．第 3 章では，システム設計の概要について説明する．第 4 章では本システムの実装について示し，第 5 章では，本システムの障害に対する動作実験の結果を示す．第 6 章で関連研究について述べ，最後に第 7 章で本論文をまとめ，今後の課題についても述べる．

2. 自己安定性に基づく合意アルゴリズム

複製手法をはじめとする種々の分散システムを実現する上で，合意問題を解くことはしばしば中心的な課題となる．そのため，これまで様々な合意アルゴリズムが提案されており，その代表的なものとして Paxos[8] が広く知られている．こうした合意問題に対して，Dolev らは自己安定的な方法によって合意問題を解決するアルゴリズムを提案している[6]．Dolev らの合意アルゴリズムは，合意形成を幾度も繰り返すことに特徴がある．この繰り返しの中の 1 回の合意を形成する期間は epoch と呼ばれ，この合意アルゴリズムでは epoch ごとに合意値が決定される．このように合意を幾度も繰り返すため，障害などの要因によって一時的に合意が形成できない状態に陥ったとしても，その要因が解消された後の合意によって，システム全体で再び合意を形成することができる．したがって，一時的に正常な状態から逸脱しても，いつか合意を形成することができる．

ここで，一般に非同期システム，すなわち命令の実行時間やメッセージの遅延，各計算機の持つ時計の変動に対して上限を持たないシステムでは，故障している計算機と処理が遅れている計算機を区別することができず，合意を形成できないことが証明されている[7]．そのため，合意アルゴリズムでは，障害検出器の存在を仮定し，処理のタイムアウトなどは障害検出器によって隠蔽される．Dolev らの論文では，提案している合意アルゴリズムが要求する自己安定性に基づく障害検出器の提案も行っており，この障害検出器は以下の性質を満たす．

- **Strong completeness**: ある時刻以降，正常なすべての計算機は，故障したすべての計算機を故障と推測し続ける．
- **Eventual weak accuracy**: ある時刻以降，故障と推測されない正常な計算機が存在する．

非同期システムにおいて自己安定的な合意を実現するために，Dolev らのアルゴリズムでは，おおよそ以下の方法が採られている．まず，各計算機は互いに独立に epoch 数と呼ばれる変数 (これを以下で e と表す) を持っている．この値は，直観的にはその計算機が最後に合意形成に参加できた epoch 数を表している．ある epoch での合意形成に参加できた計算機は，合意が完了する度に epoch 数を 1 増加させながら次の epoch での合意形成に進む．一方，合意に参加できなかった計算機は，後の epoch で合意に参加できたときに，他の計算機から過去の epoch の合意値をすべてコピーする．また，各 epoch における合意形成では，巡回コーディネータと呼ばれる手法が用いられている．これは，コーディネータと呼ばれる役割を，合意形成に参加する計算機間で巡回させ，コーディネータになった計算機が合意値を順次提案しながら最終的な合意に至るというものである．このようなコーディネータを巡回させる仕組みとして，各計算機は互いに独立に round 数と呼ばれる変数 (以下 r と表す) を持つ．また，epoch 数と round 数の他に，各計算機には 1 から n までの固有の ID 番号 (n はシステム全体の計算機数) が割り当てられているとする．さらにこれらの変数に加えて，ここでは各計算機の持つ状態を抽象化し，これを変数 s として表す．(なおここで言う「計算機の状態」とは，具体的にはその計算機がある時点でメモリやディスク，レジスタなどに保持しているデータの内容を意味し，この中には過去の各 epoch における合意値の履歴も含まれるものとする．)

以上で概観した合意形成の過程は，具体的には以下 (1)–(3) の手順からなる．(計算機間でのメッセージの交換については，**図 1** も参照のこと．)

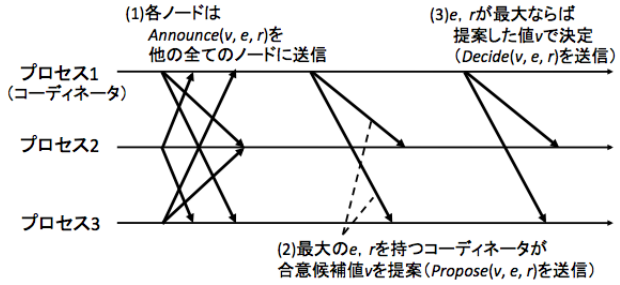

図 1 合意の手順
Fig. 1 Process of consensus

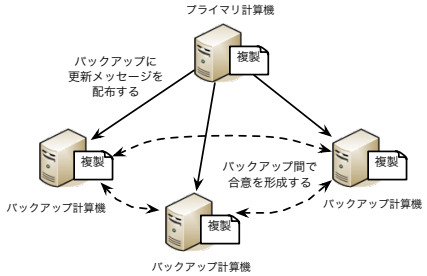

図 2 システムの概要
Fig. 2 Overview of the system architecture

(1) 各計算機は合意候補値 v を決め，この v と自身の epoch 数および round 数の 3 つ組からなるメッセージ $Announce(v, e, r)$ を他のすべての計算機に送信する．

(2) 自身の ID が $r \bmod n$ と等しい計算機はコーディネータとなる．コーディネータでない計算機は，この段階では他からのメッセージを待つだけで何も行わない．コーディネータとなった計算機は，現在自分の持つ epoch 数が (1) で得られた他のどの計算機の epoch 数よりも大きければ (等しい場合には，自分の持つ round 数の方が大きければ)，v，e，r および現在の状態 s の 4 つ組からなるメッセージ $Propose(v, s, e, r)$ を合意の提案値として他の計算機に送信する．そうでなければ，r を 1 増加させて (1) に戻る．(なお，コーディネータが既に他の計算機から提案値を受け取っている場合には，この提案値の送信は行われないものとする．)

(3) コーディネータは，(2) の終了以降に送られてきた $Announce(v, e, r)$ をもとに，依然として自身の持つ e が最大である場合には (等しい場合には，自身の持つ r の方が大きければ)，(2) で提案した合意候補値 v を epoch e での合意値として決定し，$Decide(v, s, e, r)$ を他の計算機に送信する．そうでなければ (すなわち (2) から (3) に移行する過程で他の計算機の epoch 数もしくは round 数が増加した場合には)r を 1 増加させて (1) に戻る．コーディネータおよび決定値 $Decide(v, s, e, r)$ を受け取った計算機は，自身の状態を s に書き換えるとともに，epoch 数と round 数をそれぞれ $e + 1$，0 にリセットして (1) に戻る．

以上の手順は非同期的にそれぞれの計算機によって独立に実行される．したがって，互いの epoch 数や round 数の把握は，この手順に基づいてそれぞれの計算機が別々のタイミングで送信するメッセージから行う．また，epoch 数や round 数の切り替わりも，以下の手順に従いながら個々の計算機で独立に行われる．

以上で説明した自己安定的な合意アルゴリズムについて，以下の 3 つの性質の成り立つことが Dolev らによって証明されている[6]．

- **Eventual termination**: 正常なすべての計算機は，いつか合意する．
- **Eventual validity**: 正常なすべての計算機が合意した値は，いつかいずれかの正常な計算機が提案した値となる．
- **Eventual agreement**: 正常なすべての計算機は，いつか同じ値で合意する．

Dolev らのアルゴリズムでは，Paxos のような過半数合意を基本とする合意アルゴリズムとは異なり，合意値の決定が過半数からの応答を待たずに行われる．そのため，過半数以上の計算機が故障しても，動作を継続することができる．ただしこのトレードオフとして，計算機間で通信遅延が発生すると，複数の計算機のグループでそれぞれ独立に合意形成が行われ，その結果，同一の epoch に対して異なる値で合意が形成されるということが起こり得る．(これに対して Paxos では，合意は常に過半数の計算機からの応答を待って行われるため，通信遅延によって複数の計算機のグループに分断されたとしても，合意が形成されるのは高々1 つのグループのみである．) しかしながら，Dolev らの合意アルゴリズムでは，合意形成が繰り返し行われるため，異なる値で合意が形成されるというような正常な振る舞いからの逸脱があっても，いつか必ず正常な振る舞いへと自動的に復帰することが保証される．

3. システムの概要

本システムは，自己安定性を持つ合意アルゴリズムを用いてパッシブ複製[1] を行うもの

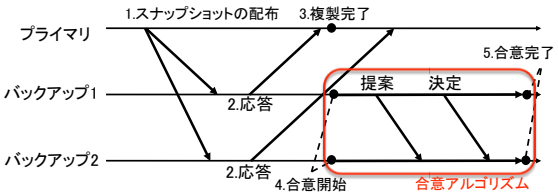

図 3 スナップショットの複製手順
Fig. 3 Process of snapshot replication

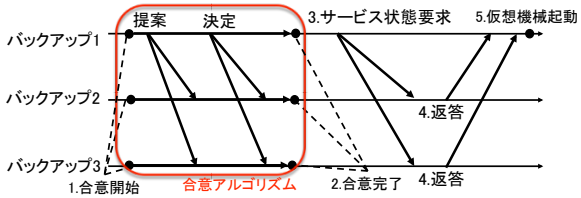

図 4 プライマリ計算機の選出手順
Fig. 4 Process of primary node election

である．**図 2** で示すように，パッシブ複製では，システム内の計算機は通常 1 台のプライマリ計算機と残りのバックアップ計算機の 2 つに分けられる．プライマリ計算機は，クライアントからのリクエストを処理し，その結果書き換わったサービスの状態を定期的にバックアップ計算機に配布する．ここで，書き換わったサービスの状態のことを以下で「更新メッセージ」と呼び，各更新メッセージはバージョン番号で管理されるものとする．プライマリ計算機が故障した場合，バックアップ計算機の中から 1 台の計算機がプライマリ計算機の代替として，それまでに送られてきた更新メッセージからサービスを復元し，再開する．

パッシブ複製を実現するには，プライマリ計算機とバックアップ計算機の状態を一致させることと，プライマリ計算機が故障した際の新たなプライマリ計算機を選出することの 2 つが必要となる．そこで，このような複製手法では，合意アルゴリズムを用いて更新メッセージが配布されるたびに，更新メッセージについて合意を取りサービス状態を一致させる．また，プライマリ計算機が故障した場合は，プライマリ計算機候補について合意を取り，正しく 1 台の計算機が選出される．本システムでは，従来の複製手法でしばしば用いられてきた過半数合意を基にした合意アルゴリズムの代わりに，先述した Dolev らの合意アルゴリズムを用いている．そのため，過半数以上の計算機の故障にも対処することができるパッシブ複製システムとなる．

まず，更新メッセージの配布による複製手法の手順を以下の (1) − (5) に示し，**図 3** に図示する．

(1) プライマリ計算機は各バックアップ計算機に更新メッセージを配布する．
(2) 更新メッセージを受信したバックアップ計算機はプライマリ計算機に ACK 応答を返す．
(3) プライマリ計算機は，少なくとも 1 台のバックアップ計算機から ACK 応答を受け取ると，バックアップ計算機が完了したと判断し，次の更新メッセージ配布の準備を行う．
(4) 更新メッセージを受け取ったバックアップ計算機は，それを合意候補値とし，Dolev らの合意アルゴリズムに従い合意を行う．
(5) 合意が完了するとサービスの複製を更新し，(1) に戻る．

次に，プライマリ計算機が故障した際の新たなプライマリ計算機選出の手順について示す．プライマリ計算機の選出では，プライマリ計算機が故障したか否かに関わらず合意を定期的に繰り返す．そのため，複数台のプライマリ計算機が起動してしまった場合にも，それ以降の合意で 1 台に戻すことができる．プライマリ計算機が故障した場合の代替からサービスの再開までを含めたプライマリ計算機選出の手順を以下の (1) − (5) に示し，**図 4** に図示する．

(1) 各計算機は現在のプライマリ計算機が正常であれば，現在のプライマリ計算機を，故障している場合は自身を合意候補値として合意を開始する．
(2) 合意完了後，プライマリ計算機が変わらない場合は (1) に戻る．
(3) 新たなプライマリ計算機が選出された場合，プライマリ計算機は各バックアップ計算機が持つ更新メッセージのバージョンを問い合わせる．
(4) 問い合わせを受けた各バックアップ計算機は，合意が完了している更新メッセージを含めたサービスの複製をプライマリ計算機に返す．
(5) プライマリ計算機は受け取ったサービスの複製の中でより新しいものからサービスを再開する．その後 (1) に戻る．

以上の手順の注意点として，Dolev らの合意アルゴリズムの性質上，一時的に最新の更新メッセージでの合意が形成できている計算機とそうでない計算機が混在してしまうことがある．また，プライマリ計算機が故障した場合，各バックアップ計算機が自身を合意候補値として合意を行うため，新プライマリ計算機は最新の更新メッセージで合意できていると

は限らない．そこで，より新しいサービスの状態からサービスを再開する方法として (4), (5) の手続きを行っている．このとき，(5) の更新メッセージの受信ではタイムアウトを設け，時間内に受信した更新メッセージの中からより新しい更新メッセージを選ぶ．しかし，タイムアウト以降により新しい更新メッセージが見つかることもあるため，サービスを再開させた後も，より新しい更新メッセージを受信すると，その更新メッセージを基にしたサービスに切り替える．

4. 実　　装

本システムはプログラミング言語として Scala[9] を用いて実装している．Scala では，メッセージパッシング方式の並列プログラムを実現するための Actor と呼ばれるライブラリを提供している．本システムでは，Actor を利用して各モジュールを実装しており，メッセージパッシング方式による分散システムを実現している．各モジュールは独立した Actor 上で動作しており，相互にメッセージの送受信を行い動作する．本システムは障害検出器と合意アルゴリズムおよび，それらを利用したプライマリ計算機の選出機構とパッシブ複製機構で構成している．

本システム上で動作するサービスについては，プライマリ計算機上で仮想機械を動作させ，サービスの提供を行う．そのため，OS やライブラリといったサービスに必要な環境すべてをカプセル化することができ，特定の環境に依存しない複製が可能となる．また，パッシブ複製の更新メッセージは仮想機械のスナップショットを用いることで実現することができる．

また，システムの一部は著者らが研究開発を行っている Kumoi[13] を利用して実装している．Kumoi とは，クラウド環境における並列分散スクリプティングフレームワークである．Kumoi では，クラウドシステムの研究を行う上で便利なライブラリを多数提供している．本システムでは，Kumoi が提供している仮想機械のインターフェースを利用しており，Kumoi の仮想機械インターフェースには，仮想機械用の API である libvirt を用いられている．そのため，本システムはサービスの動作環境として，Xen や KVM と言った様々な仮想機械を利用することができる．

5. 実　　験

本システムの障害発生時の振る舞いを検証するため，実装したシステムを使って動作実験を行った．実験で用いる計算機は，Dual Xeon 3.60GHz の CPU，2GB のメモリを持ち，ホスト OS に CentOS5.5，ゲスト OS に CentOS5.4 を用いている．仮想機械には Xen3.0.3 を用い，プログラムを動作させる Java 仮想機械のバージョンは JVM 1.6 である．システムは 5 台の計算機を 1000Base-T のネットワークで繋いで構成しており，メッセージパッシング方式によってシステム内のすべての計算機と互いに通信できる．また，障害検出器は各計算機が 1 秒ごとにハートビートを送り，10 回検出できない場合に障害と判断する．

以上のシステム環境を用いて 6 つの動作実験を行った．

(1) **単一故障**：プライマリ計算機を停止させる．また，新たなプライマリ計算機の選出後に同様にプライマリ計算機を停止させ，それぞれの停止後に 1 台の計算機がプライマリ計算機として選出されることを確認する．

(2) **過半数故障**：3 台の計算機を同時に停止させる．このように，過半数以上の計算機が停止しても，1 台の計算機がプライマリ計算機として選出されることを確認する．

(3) **ネットワーク分断**：一時的に通信障害を起こし，システム内の計算機を 2 つのグループに分ける．その後，通信を回復させる．このような通信遮断によるネットワーク分断が発生すると，自己安定な合意アルゴリズムの性質上，グループごとに別々に合意を行ってしまう．しかし，このような場合でも，ネットワーク分断が解消された後，再び全体で合意が行われ，状態が一致することを確認する．

(4) **プライマリ計算機選出直後の故障**：プライマリ計算機を停止させ，新たなプライマリ計算機が選出される直後にその計算機を停止させても，1 台の計算機が新たにプライマリ計算機として選出されることを確認する．

(5) **最新のスナップショットの喪失**：最新のスナップショットを持つ計算機が 1 台のみの状態で，プライマリ計算機と最新のスナップショットを持つ計算機を停止させ，古いスナップショットから仮想機械が起動することを確認する．

(6) **古いスナップショットからの起動**：最新のスナップショットが全体に配布されていない状態でプライマリ計算機を停止させる．このとき，新たなプライマリ計算機が一時的に最新のスナップショットを受け取れない状態にするため，新たなプライマリ計算機が各計算機からスナップショットを収集するための期間を 5 秒間とし，最新のスナップショットを持つ計算機では，スナップショットを返す前に 8 秒間のスリープを行うようにした．第 3 章で述べたとおり，Dolev らの合意アルゴリズムでは，一時的に各計算機の所持しているスナップショットのバージョンが一致していないことがある．このような状態でプライマリ計算機が停止すると，新しいプライマリ計算機に古いスナップショットが送られてしまう．このとき，最新のスナップショットを所持し

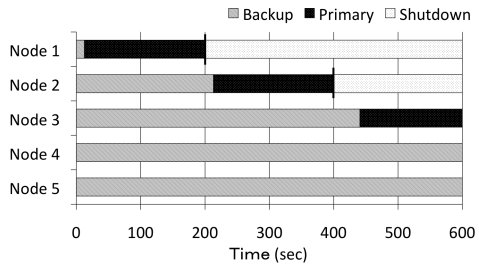

図 5 単一故障
Fig. 5 Serial failures

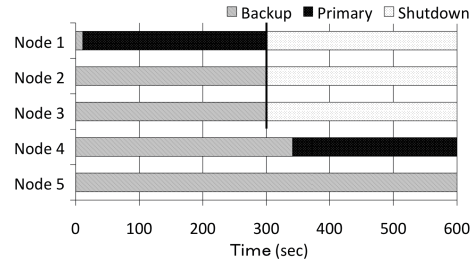

図 6 過半数以上の故障
Fig. 6 Failure of the majority of machines

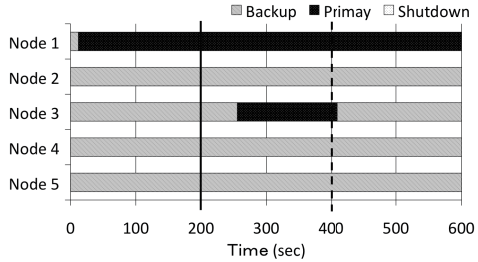

図 7 ネットワーク分断
Fig. 7 Network partitioning

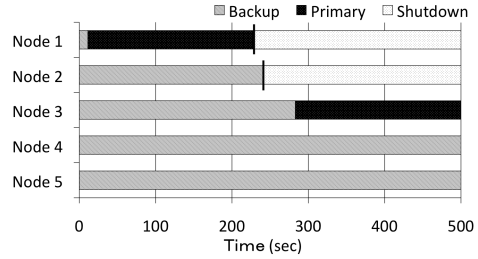

図 8 プライマリ計算機選出直後の故障
Fig. 8 Failure during election

ている計算機との通信が遅れると，古いスナップショットから仮想機械を起動させてしまうが，最新のスナップショットの状態でサービスを切り替えることを確認する．

以上の実験結果を以下に示す．

(1) **単一故障**：**図 5** に結果を示す．まず，プライマリ計算機である Node 1 を 200 秒経過時に停止させた．このとき，Node 1 はスナップショットの配布を完了していないため，新たなプライマリとして選ばれた Node 2 は 213 秒経過時に仮想機械をベースイメージから起動させる．次に，Node 2 を 400 秒経過時に停止させた．こちらは，スナップショットの配布完了後に停止したため，Node 3 は最新のスナップショットを受け取った後，440 秒経過時に仮想機械を起動している．

(2) **過半数故障**：5 台の計算機の中からプライマリ計算機を含む 3 台の計算機を一斉に停止させた結果を**図 6** に示す．300 秒経過時に動作している計算機が過半数以下になるが，341 秒経過時に Node 4 が新たなプライマリ計算機として選出され，仮想機械を起動している．

(3) **ネットワーク分断**：200 秒経過時に Node 1 と 2，Node 3-5 の間で通信を不能にした．**図 7** で示したとおり，Node3-5 では，プライマリ計算機が失われるため，255 秒経過時に Node 3 がプライマリ計算機として仮想機械を起動している．その後，400 秒経過時に通信が回復すると，Node3-5 からも Node 1 のプライマリ計算機が正常であると確認できるため，410 秒経過時に Node 3 は仮想機械を停止している．

(4) **プライマリ計算機選出直後の故障**：プライマリ計算機が停止してから 12 秒後に Node 2 が新たなプライマリ計算機として選出されるが，直後に Node 2 を停止させた．**図 8** から，Node 2 の停止から 42 秒後に Node 3 が新たなプライマリ計算機として選出され，仮想機械を起動していることが確認できる．

(5) **最新のスナップショットの喪失**：バージョン 1 のスナップショットを全体に配布した

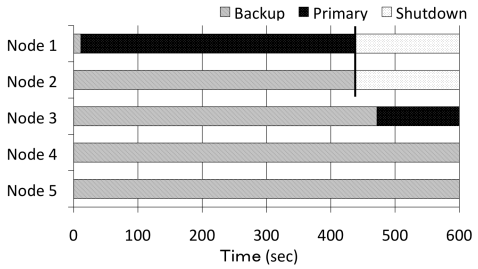

図 9 最新のスナップショットの喪失
Fig. 9 Loss of the latest snapshot

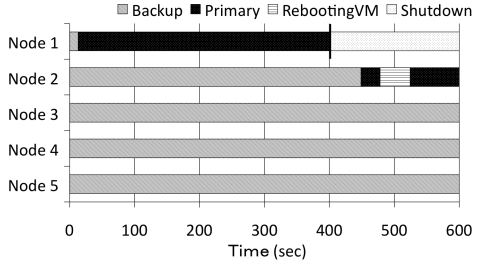

図 10 古いスナップショットからの起動
Fig. 10 Booting a backup with an old version of the snapshot

後，バージョン 2 のスナップショットが 1 台にのみ配布された状態で，プライマリ計算機とバージョン 2 のスナップショットを持つ計算機を停止させた．**図 9** から，35 秒後に新たなプライマリ計算機として Node 3 がバージョン 1 のスナップショットで仮想機械を起動した．

(6) **古いスナップショットからの起動**：システム全体でバージョン 1 のスナップショットによる合意が取れた後，バージョン 2 のスナップショットが Node 4 と 5 にのみ配布された状態で，プライマリ計算機を停止させた結果を**図 10** に示す．400 秒経過時にプライマリ計算機を停止させると，448 秒経過時に新たなプライマリ計算機となった Node 2 がバージョン 1 のスナップショットから仮想機械を起動した．しかし，他のバックアップ計算機が最新のスナップショットを所持しているため，478 秒経過時にバージョン 1 のスナップショットから起動した仮想機械を停止させ，524 秒経過時には最新のスナップショットから仮想機械を起動させている．

以上で示した 6 つの実験からわかることは，まず実験 1 および 2 から，プライマリ計算機が故障してしまい正常な計算機がシステム全体の過半数以下になっても，合意により新たなプライマリ計算機が選出され，サービスが継続できることがわかる．また，実験 4 から，新たなプライマリ選出途中でプライマリ計算機の候補が故障すると，サービスの再開が遅れてしまうが，正しく再開することができることがわかる．

次に実験 3 では，ネットワーク分断時にも両拠点においてサービスの継続が可能であることがわかる．しかし，ネットワーク分断が起こると，各プライマリ計算機が提供するサービスの状態に一貫性が保てなくなる．また，プライマリ計算機が 1 台に戻る際に，各プライマリ計算機間のサービスに行われた更新を矛盾なく融合するため，サービスのマージ処理を行う必要もある．現在これらの対処法として，過半数以上の計算機と通信ができなくなった際に，提供するサービスに制限を加える手法や，更新のタイムスタンプをとり，それを基にマージする手法を考えている．マージ手法の実装については今後の課題とする．

最後に，実験 5 および 6 では古いスナップショットから仮想機械が起動している．実験 5 では，最新のスナップショットが完全に失われるため，プライマリ計算機は最新を除いたスナップショットの中で，より新しいスナップショットから仮想機械を起動している．また，実験 6 では，最新のスナップショットが存在するものの，プライマリ計算機が最新のスナップショットを持たないため，古いスナップショットから仮想機械を起動させている．その場合にも，最新のスナップショットが後から見つかると，最新の状態にサービスを切り替えることができる．しかし，ただ切り替えるだけでは，古いスナップショットを用いてサービスを起動してから切り替わるまでに行われた更新が失われてしまう．そのため，ネットワーク分断からの回復時と同様に，サービスの状態をマージする必要があり，その実現方法についても今後の課題である．

6. 関 連 研 究

インターネットサービスの耐障害性を向上させる基盤ソフトウェアとして，サステーナブルシステム[14] がある．サステーナブルシステムでは，仮想機械を利用したパッシブ複製を行うシステムであり，本システムが目指すものとも一致している．サステーナブルシステムでは，合意アルゴリズムに Paxos を利用しており，過半数以上の計算機が正常であるという仮定を置く代わりに，仮定が満たされている限り常に正常に動作するシステムである．それに対し本システムは，一時的に正常な振る舞いから逸脱することを許すシステムであるが，自動的に正常な動作へと復帰するサービス基盤である．

同じく仮想機械を利用したパッシブ複製を行う複製機構として Remus[2] がある．Remus は Xen の live migration 機能を改変したスナップショット機能により，1 秒間に数十回という高い頻度で複製を行っている．しかし，複製手法に合意アルゴリズムが用いられているか明らかではない．一方，アクティブ複製による複製手法の研究として VM-FIT[10] がある．アクティブ複製はパッシブ複製と比較すると，ネットワーク転送量が小さいという利点があるが，決定的であるサービスでの利用に限られる．決定的であるとは，状態と入力によって次に遷移すべき状態が一意に定まることを言う．

アクティブ複製やパッシブ複製以外の複製手法として，準アクティブ複製（semi-active replication）[12] や準パッシブ複製（semi-passive replication）[3] などがある．これらの手法は，アクティブ複製とパッシブ複製を組み合わせ，それぞれの欠点を補った手法である．本システムではパッシブ複製を用いているが，より実用的なシステムを実現するための方法として，これらの複製手法の応用についても今後の検討課題としたい．

7. まとめと今後の課題

本研究では，自己安定性に基づくインターネットサービス基盤を提案した．本システムは，パッシブ複製による複製手法を自己安定性を持つ合意アルゴリズムを用いて実現している．そのため，一時的な最大同時故障台数に関する制限がなくなり，多数の計算機が同時に故障する障害にも対処することが可能になる．このようなシステムを Scala により実装し，仮想機械と組み合わせた動作実験を行った．

今後は課題として，まず実際のサービスをシステム上で動作させ，その評価実験を行うことが挙げられる．次に，より実用性の高いシステムを実現するための設計を行っていく．具体的には，プライマリ計算機が 2 台から 1 台に戻る際のサービスのマージを行う．また一方で，より堅牢な自己安定のアプローチとして，強安定（superstabilization）がある．強安定とは，自己安定アルゴリズムが障害から復帰する過程において，満たすべき安全性に関する制約を加えたものである．そのため，小規模な障害に対して，より早い復帰が可能または異常な状態に陥らないシステムの実現が期待される．

謝辞 本研究の一部は，総務省戦略的情報通信研究開発推進制度 (SCOPE)ICT イノベーション促進型研究開発「ディペンダブルな自律連合型クラウドコンピューティング基盤の研究開発」の支援を受けて行われた．

参 考 文 献

1) N. Budhiraja, K. Marsullo, F. B. Schneider, S. Toueg. The Primary-Backup Approach. *Distributed Systems (2nd ed)*, pp.199–216, 1993.
2) B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. *USENIX NSDI'08*, pp.161–174, April 2008.
3) X.Defago and A. Schiper. Semi-passive replication and Lazy Consensus. *Journal of Parallel and Distributed Systems*, Vol.64, No.12, pp.1380–1398, December 2004.
4) E. W. Dijkstra. Self-stabilizing System in Spite of Distributed Control. *Communications of the ACM*, Vol.17, No.11, pp.643–644, November 2006.
5) S. Dolev. *SelfStabilization*. The MIT Press, 2000.
6) S. Dolev, R. I. Kat, and E. M. Schiller. When Consensus Meets Self-Stabilization. *10th International Conference On Principles Of Distributed Systems*, Vol. 4305, Springer LNCS, pp.45–63, November 2006.
7) M. J. Fischer, N. A. Lynch and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, Vol.32, No.2, pp.374–382, April 1985.
8) L. Lamport. The Part-time Parliament. *ACM TOCS*, Vol.16, No.2, pp.133–169, May 1998.
9) M. Odersky. Scala, http://www.scala-lang.org/
10) H. P. Reiser and R. Kapitza. Hypervisor-Based Efficient Proactive Recovery. *IEEE SRDS'07*, pp.88–92, Octorber 2007.
11) F. B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, Vol.22, No.4, pp.299–319, December 1990.
12) D. Stodden. Semi-active Workload Replication and Live Migration with Paravirtual Machines. *Xen Summit*, Spring 2007.
13) A. Sugiki, K. Kato, Y. Ishii, H. Taniguchi, N. Hirooka. Kumoi: A High-Level Scripting Environment for Collective Virtual machines. *ICPADS'10*, to appear.
14) 杉木章義，大和崎啓，加藤和彦．広域分散環境のための仮想機械を利用したサービス協調複製基盤．情報処理学会論文誌コンピューティングシステム，Vol.2, No.1, pp.1–11, March 2009.
15) 大和崎啓，長谷部浩二，杉木章義，加藤和彦．耐障害性向上のための自己安定性に基づくインターネットサービス基盤の構築．ソフトウェア科学会コンピュータソフトウェア，近刊．