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

小磯知之[†,☆] 阿部洋丈[††] 池嶋俊[†]
石川宗寿[†] リチャードポッター[††] 加藤和彦[†,††]

本論文では，大規模障害を含めた様々な障害を自律的に乗り越えて持続可能な（サステーナブルな）サービスを実現するための基盤ツールキットの設計について述べる．本方式は，多数の計算機を連合させ，それらにサービスの実行とサーバ機能の状態保存を自律的に分担させることでサービスのサステーナブル化を実現する．また，本提案方式は，仮想計算機を用いることで，既存のサービスを容易にサステーナブルにすることを可能にしている．

Design of an Infrastructure Toolkit for Sustainable Service

Tomoyuki Koiso,[†,☆] Hirotake Abe,[††] Syun Ikejima,[†]
Munetoshi Ishikawa,[††] Richard Potter[††] and Kazuhiko Kato[†,††]

In this paper, we describe our design of an infrastructure toolkit for realizing sustainable services, which can surmount various kinds of failures, including catastrophe. Our system consists of many federating computers. It enables the service to be sustained by making some of the computers run server functions, and the others share storing states of the service. And also our method is designed to be applicable for existing services by using virtual machines.

1. はじめに

インターネットを介して提供されるサービスは，様々な原因によってしばしばサービス停止状態に陥る．その原因として最近注目を集めているのが，災害などの原因によって生じる大規模障害（catastrophe）である．一般的なハードウェア障害はシステムの一部のコンポーネントの故障が原因であるのに対し，大規模障害では様々なコンポーネントが同時に故障し，その結果，サービスが完全に機能停止する．

大規模障害の原因としては，地震・火災・事故・テロなどによる物理的破壊があげられる．しかし，これらのほかにも，物理的な破壊に匹敵しうる障害がある．それはインターネット接続障害である．インターネット接続は，インターネットを介したサービスにとっていわば生命線であり，その障害はすなわちサービス全体の完全停止を意味する．インターネット接続障害の原因としては，ネットワーク機器の障害，工事ミスや動物によるケーブル断線，ネットワーク機器の設定ミスなどがある．

そのような大規模障害に対処するために，遠隔地にバックアップシステムをあらかじめ用意するというアプローチがとられている[3),18)]．このアプローチは，物理的に，もしくは，ネットワーク的に，もしくはその両方で十分に離れた場所にバックアップシステムを配置し，メインシステムにおける処理内容をネットワーク越しに伝搬する．そうすることで，メインシステムがサービス停止状態になった場合でも，バックアップシステムがすぐにサービスを再開可能な状態を保つことができる．

しかし，このようなアプローチは非常にコストがかかる．たとえば，同じシステムが動作するためには，メインシステムと同等の処理能力を持ったバックアップシステムを常時待機させておく必要がある．バックアップシステムには，計算機そのものに加えて，インターネット接続，土地建物，電源・空調設備，人員も含まれる．また，処理内容を迅速に伝搬するためには，インターネット接続とは別に，信頼性や品質の保証された専用ネットワークがしばしば用いられる．これらの理由から，バックアップシステム維持のための TCO

† 筑波大学
University of Tsukuba
†† 科学技術振興機構 CREST
JST CREST
☆ 現在，東日本電信電話株式会社
Presently with NTT East Corporation

（Total Cost of Ownership）は膨大になりうる．銀行の勘定系システムなど，潤沢な予算をかけることが可能なシステムには十分適用可能だが，そうでないシステムにとっては現実的なアプローチとはいえない．しかし，潤沢な予算をかけることができないシステムであっても，大規模障害に対処可能にしたい場合がある．

そこで我々は，少ないコストで大規模障害に対処するための方式の実現に向けて研究を行っている．我々の目論見は，高価な多重化システムを使うのではなく，非常に多数の安価な計算機を組み合わせることで同様の効果を実現するというものである．これまでにも，メインシステムとバックアップシステムの 1 対 1 ではなく，1 つのメインシステムに対して複数のバックアップを持つ方式[9] や，複数のサイトがお互いにお互いのバックアップを持ち合う方式[6] が提案されている．それらの方式は，比較的少数の（たとえば 3 台から 5 台程度），ある程度優れた性能を持つ計算機を利用することを想定している．それに対して我々の方式は，一般的なパーソナルコンピュータを非常に多数，たとえば数十台から数百台程度を連合させることによって実現することを目指している．これらの PC は，そのサービス専用である必要はなく，普段は別の目的に使われていてもよい．我々は，そのようにして実現されるサービスのことを**サステーナブルサービス**（Sustainable Service）と呼んでいる．

サステーナブルサービスの目的は，サーバ計算機自体を継続的に動作させることではなく，広域分散環境に置かれたコンピュータが自律的に協調動作することで，サービス自体を持続させることである．つまり，サーバとなるコンピュータ自体を多重化などして頑強にする手法や，多重化された専用通信回線を用いるなどして耐故障性を持たせる手法とは異なる．また，我々の手法と既存の手法は共存可能であり，単体で効果を得ることも，また，組み合わせてさらに高い効果を得ることも可能である．

本研究の目的は，既存のサーバプログラムを利用する形でサステーナブルサービスを実現することである．そこで本論文では，サステーナブルサービス実現のための一方式として，サステーナブルサービスツールキットを提案する．サステーナブルサービスツールキットとは，既存のサーバプログラムをサステーナブルサービス化するための基盤となる機能群である．我々の提案システムの特徴は，サーバの持つデータを広域分散ストレージで保存するという既存方式とは異なり，仮想計算機技術を用いることで，サービスを提供する機能そのものを持続可能にするという点である．仮想計算機モニタには，仮想計算機の実行状態を永続化する機能（**スナップショット機能**）を持つものがある．そのような機能を利用することで，サーバプログラムをまったく改変することなく実行状態を保存すること，さらに，その実行状態を用いることで，システムに参加しているどの計算機上でもサーバの機能を復元することが可能になる．それにより，障害発生時におけるクライアントからサーバへの到達可能性を向上させることができる．また，本提案方式には，仮想計算機を用いることで，異なる計算機上でサーバプログラムが動作する場合に生じうる，環境の違いによるトラブルの発生を抑えることができるという利点もある．

我々の方式では，ネットワークの分断に際して，同時に複数のサーバが動作する場合がある．また，障害発生時には，一時的に一部のデータへのアクセスが不能になる場合がある．しかし，一般に，既存のシステムはそのような事態を想定しては作られていない．そのため，本ツールキットを用いてすべてのサービスをサステーナブル化することができるわけではない．また，本ツールキットによってサステーナブル化することが可能とされる場合でも，障害の発生を隠蔽するために，補助プログラムの作成，もしくは，サーバプログラムに対する改変が必要となる場合がある．本論文では，サステーナブル化するために必要な条件，および，サステーナブル化するために必要な変更についても議論する．

以降，本論文は次のように構成されている．2 章ではサステーナブルサービスについて述べる．3 章ではサステーナブルサービスツールキットの設計について述べる．4 章ではその設計に基づいたプロトタイプシステムの実装について述べる．5 章ではそのプロトタイプシステムを用いた性能測定の結果について示す．6 章ではまとめと今後の課題について述べる．

2. サステーナブルサービス

サステーナブルサービスの目的は，大規模障害を含めた様々な障害を乗り越えてサーバ機能が生き延び，サービスの提供を持続させることを可能とすることである．従来は，サービスを持続させるためにサーバ機能を実行する計算機を生き延びさせることが行われてきた．サステーナブルサービスのアイディアは，生き延びさせるべきなのはサーバ機能であり，サーバ計算機ではないということである．そのため，多重化などによって信頼性を高めた計算機を仮定せず，通常の計算機を多数利用することで冗長性を高める．ただし，このアプローチは，計算機の信頼性を高める従来の

アプローチと対立するものではなく，サステーナブルサービスを実現している計算機群の信頼性が高ければ，それはサービス全体の安定性の向上に貢献する．

以下に，サステーナブルサービスの動作の概観を示す．

サステーナブルサービスは，インターネットに接続された複数の計算機によって構成される．サービスプログラムは，この計算機群上のいずれかの上で動作する．システムは，このサービスプログラムを実行している仮想機械を外部に公開し，クライアントからのサービス要求に対応する．以降では，サービスを実行しているシステムの計算機をサーバと呼び，その他の計算機のことをサーバ候補と呼ぶ．サーバとサーバ候補群はオーバレイネットワークを構成している．

計算機群は，サービスプログラムの状態をこのネットワーク上で共有する（**実行状態の共有**）．サーバは，障害が発生した後で新しいサーバがサービスの提供を継続できるように，サービスプログラムの状態を永続化してサーバ候補群へと転送する．サーバ候補群は，障害によってサービスが停止する場合に備えてその状態を保存する．

サーバ候補群は，サービスプログラムが正常にサービスを提供できているかを監視している（**サーバの監視**）．この監視で異常が検出された場合，サーバ候補群から新しくサーバとして振る舞うサーバ候補（次期サーバ）が選出される（**次期サーバの選出**）．次期サーバは，オーバレイネットワーク上で共有されているサービスプログラムの状態を収集（**実行状態の収集**）し，それらを用いて障害が起こる直前のサービスプログラムの状態を復元して，サービスの提供を継続する．

サービスプログラムは，負荷分散やスループットの向上を目的として複数のサーバ上で実行することが可能である（**サーバの複製**）．また，サービスの要求の変化などに応じて，その規模を縮小することも可能である（**サーバの合流**）．

3. サステーナブルサービスツールキットの設計

サステーナブルサービスを実現するためのアプローチとしては，既存のアプリケーションを利用するアプローチと，サステーナブル性を有するサービスを新たに設計するアプローチが考えられる．サステーナブルサービスツールキットは前者のアプローチのためのものであり，既存のアプリケーションを少ない変更でサステーナブル化することを目的としている．本論文では前者のアプローチに基づく設計について扱うが，後者のアプローチの場合でも設計の大半（仮想機械制御の設計およびサーバ合流の設計以外）については共通である．以下ではまず，提案システムの概観を示し動作の概略を述べる．次に，本ツールキットを用いてサステーナブル化できるサーバプログラムの範囲について，また，サステーナブル化可能なプログラムの分類とその実現方法について議論する．その後，サステーナブルサービスツールキットとして持つべき各機能の設計について述べる．

3.1 提案システムの概観

提案システムに参加する計算機を**ノード**と呼ぶ．サーバの役割をしていないノード（以降，**サーバ候補**と呼ぶ）はサーバの役割をしているノードを監視し，配布された実行状態を保存する．既存のアプリケーションプログラムは，基本的に単一のサーバ上で実行される．サーバに異常が発生した場合に共有されている実行状態をもとに，サーバを選択した 1 つのサーバ候補の仮想機械上に復元する．サーバ自体に起こったものではない障害によって，サーバが他の場所で復元されていた場合，障害が回復したときにサーバが複数存在する事態が起こる．このとき，サービスによっては複数のサーバが存在することを許さないことがある．提案システムはこのようなサービスに対して，複数のサーバを 1 つに合流させる．

上述の一連の動作を継続することで，提案システムはサステーナブルサービスを提供する．提案システムが提供する機能は，仮想機械の制御・実行状態の共有・障害検知・サーバ選出・サーバ復元・サーバ合流の 6 つに大別できる．これらの機能を具体的に設計し，実装を行うことで提案システムを実現する．上記の機能を有するノードの動作は**図 1** のような状態機械として表すことができ，この状態機械を分散環境上で各ノードが実行することでサービスにサステーナブル性を付加できる．

3.2 提案ツールキットの適用可能範囲

サステーナブルシステムでは，ネットワークの分断

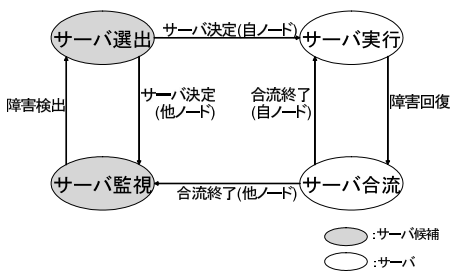


図 1 サステーナブルサービスツールキット状態遷移図
Fig. 1 State diagram used in Sustainable Service Toolkit.

のためにサーバが複数に分裂し，それぞれが並行してクライアントにサービスを提供する場合がある．また，サーバに障害が生じた場合や，サーバがネットワークから切り離された場合などには，サーバ上で行われた操作が失われる可能性がある．しかし，一般には，既存のサーバソフトウェアはそのような自体を想定して設計されていない．そのため，まったく手を加えないでサステーナブル化できる既存のサーバソフトウェアは多くない．あるサーバソフトウェアのサステーナブル化を検討する場合，それを実現ことが可能か，また，もし可能だとしても，どのような制約を受けるのかということを事前に検討することが重要である．

今回我々は，既存のサステーナブル化可能性の観点から，サーバソフトウェアの分類を試みた．ただし，サステーナブル化可能なサーバプログラムについて考える場合，起こりうるすべての障害について想定することは困難であった．そこで今回は，議論を簡単にするために，次のような仮定を置いた．

(1) ネットワークは，一時的に分断していてもよいが，障害発生後 Δf までには回復する．
(2) サーバが持つべきデータが一時的に消失してもよいが，障害発生後 Δf までに回復する．

ここで，Δf とは，サステーナブルシステムに参加している各計算機の管理者間で合意された，システムの復旧期限である．各管理者は，この時間内に障害が回復されるように努めることが要請される．たとえば，サーバになっている計算機にインターネット接続障害が起きた場合は，その計算機の管理者は Δf 以内にその接続を復旧させるか，代替の接続手段を講じることが要請される．また，サーバになっている計算機に障害が起きた場合は，その計算機の管理者は Δf 以内に計算機を修理し，障害発生直前の状態を復元することが要請される．計算機を障害発生直前の状態に戻すためには，テープなどのバックアップ装置，もしくは，ネットワークで接続された他の計算機などにデータを保存しておく必要がある☆．

既存のサービスがサステーナブル化できるかを判断するためには，アプリケーションの性質と，そのサービスに対する要求の性質を考慮する必要がある．上記2つの性質をあわせてサービスの性質とする．したがって同じアプリケーションを用いたとしても，サービスの性質は変わる可能性がある．

まず，サービスの持つ状態に変更が起きないサービスは，単純なコピーの存在を許すために，サステーナブル化可能である．この性質を持つサービスを Type0 とする．それ以外のサービスついては，サステーナブル化できるかを判断するために，性質を以下の 2 つの基準で判断する．

(1) 一部分のデータのみで処理を実行可能（partial-computable）か処理のためにはすべてのデータが揃っていることが必要（complete-data-required）か．
(2) サービスの実行に最大 Δf の遅延が許されるか（delay-acceptable），もしくは許されないか（delay-forbidden）．

この 2 つの基準をもとに，サービスは以下の表に基づき 3 つのサステーナブル化可能なカテゴリと，1 つのサステーナブル化不可能なカテゴリに分かれる（**表 1**）．

表 1 サステーナブル化可能性から見たサービスの分類
Table 1 Classification of services by feasibility of making sustainable.

	dealy-acceptable	delay-forbidden
partial-computable	Type1	Type2
complete-data-required	Type3	Cannot

以降，各 Type ごとにサービスの特徴と，サステーナブル化するために必要となる追加機能，および，障害発生時に起こりうる機能低下について述べる．

Type0

このサービスは状態を持たないため，サーバのコピーが存在しても影響がない．Type0 に分類されるサービスは，サステーナブル化のために必要な追加機能はない．

Type1

部分的なデータのみで処理することができ，かつサービスの遅延も許されるサービスはこのカテゴリに分類される．必要な追加機能は，サーバの合流が起こる際にそれぞれが持つデータをマージする機能である．マージをする方法は，上書き，変更点のマージ，独立したデータとして別個に保存，などが考えられる．障害発生時には，一部の処理の実行が遅延される．

Type2

部分的なデータのみで処理することはできるが，遅延は許されず，即時に応答することが求められるサービスはこのカテゴリに分類される．必要な追加機能は，前述のマージ機能と，遅延の発生が避けられない状況下ではサービスの提供を拒否する機能である．

Type3

サービスにおけるデータの処理を行うためにはすべ

☆ 復元する必要のあるデータをバックアップする目的で，後述の広域分散ストレージを利用できる可能性があるが，その検討は今後の課題である．

てのデータが揃っていることが要求されるが，サービスの遅延が許されるサービスはこのカテゴリに分類される．必要な追加機能は，すべてのデータを揃えられる場合のみマージを行う機能が必要である．また，場合によっては，すべてのデータを揃えられるまでクライアントからの要求の処理を遅延させる機能も必要になる．

Cannot

障害が起きたときにでも遅延は許されず，かつ，必要なデータが揃っていることが要求されるサービスはこのカテゴリに分類される．このカテゴリに含まれるサービスはサステーナブル化することができない．

3.3 既存サービスのサステーナブル化に関する考察

あるサービスのサステーナブル化を検討する場合，障害時にそのサービスがどのような挙動を示すことが要求されているかということを明らかにすることが重要である．なぜならば，それによってサステーナブル化できるかどうか，また，サステーナブル化するのに必要となる追加機能，および，サステーナブル化することによる機能低下が判断できるからである．以降，複数の異なる要求が存在し得る既存サービスを題材に，2 種類の異なる要求を例にあげ，どのようにサステーナブル化を検討することができるかを考察する．

静的なウェブサーバ

静的なウェブサーバとは，主に静的なコンテンツの配信を目的としたウェブサーバを指す．ここでは 2 種類の静的なウェブサーバの例を考える．1 つ目の例は，可能な限り多くのクライアントにウェブページを閲覧してほしい場合である．その場合は，Type0 のサービスにすることが求められる．ただし，Type0 を選択した場合は，障害時にウェブページを更新することができなくなる．2 つ目の例は，可能な限り最新の更新内容をクライアントに閲覧してほしいが，それができない場合は古い内容でもよいので閲覧してほしいという場合である．そのような場合は Type2 に分類される．

メッセージ配送サーバ

1 つ目の例は，障害発生時にはメッセージの配送が遅延してもよいので，できるだけ多くのクライアントが利用できるようにしたいという場合である．その場合は Type1 が選択される．2 つめの例は，障害時においても，即時にメッセージを配送してほしいという場合である．その場合は，Type2 が選択される．その結果，即時性が保証できない場合にはサービス提供が拒否される可能性が出てくる．

ファイルサーバ

1 つ目の例は，障害発生時でもできるだけサーバを利用したいという場合である．その場合は Type1 が選択される．その場合は，データの更新に遅延が発生し，すべてのデータが揃っていることは保証されなくなる．2 つ目の例は，障害発生時においてもつねに最新のファイルがすべて揃っていることが要求される場合である．本提案方式の枠組みにおいてはそのように保証することはできないので，この場合はサステーナブル化不可能となる．つまり，cannot に分類せざるをえない．

ID 登録サーバ

1 つ目の例は，障害時でも即時に ID 登録を完了させたいという場合である．ID を登録するためには，他の誰かが同じ ID を取得していないことを確認する必要がある．すべてのデータを即時に集めることは不可能であるため，この場合は cannot に分類される．2 つ目の例は，障害発生時には ID 登録申請可否の通知が遅延してもよいという場合である．その場合は，すべてのデータが揃っている必要があるが，遅延が許容されているので，実現可能である．つまり，この場合は Type3 に分類される．

3.4 仮想機械

アプリケーションプログラムを変更せずに実行状態を取得するために，実行状態を取得できる仮想機械を用いる．しかし，仮想機械の中にはメモリやファイルシステムなど，計算機の実行状態をすべて保存することで，実行状態を取得するものがある．この方式は実行状態のデータサイズが巨大（数百 MB〜数 GB）になってしまうことがあるが，アプリケーションプログラムを変更することなく実行状態を取得することができるという大きな利点がある．

この利点を生かすことで，提案システムと既存のアプリケーションとの親和性を高めることができる．しかし，このように巨大な実行状態を分散環境で共有することは，ネットワーク負荷や，共有するために必要な記憶領域などの観点から現実的ではない．そこで，提案システムでは実行状態の差分を生成し，差分情報を結合することで元の実行状態が復元できるよう工夫する．差分情報を共有することで，少ない情報量で実行状態を共有することができるので，先に述べた問題を軽減できる．

3.5 ノードの設計

提案システムに参加するノードは，それぞれ通信可能なノードの IP アドレスとポート番号の組のリスト（以降，ノードテーブル），自ノードの持つ実行状態のファイル名のリスト（以降，ファイルリスト），通信不能になったノードの IP アドレスとポート番号のリス

表 2 仮想機械制御のトリガ
Table 2 Triggers of actions controlling virtual machine.

コマンド	トリガ
仮想機械の起動	サーバ開始時
実行状態の取得	タイマー/サービスからの要求
実行状態の再生	サーバ復元時
仮想機械の終了	合流終了時

トを持つ．また，現在のサーバの IP アドレスとポート番号を保持している．サーバは仮想機械の制御，実行状態の配布を行う．サーバ候補はサーバの監視を行う．その他のファイル転送などの基本的な手続きはサーバとサーバ候補が共通して扱うプログラム群として実装する．

3.6 仮想機械の制御

仮想機械に対して提案システムは，実行状態の取得と再生，仮想機械の起動と終了の要求を行う．仮想機械によっては，これらを外部から実行するために API が用意されており，コマンドを実行することで制御できる．提案システムからはそのコマンドを必要に応じて実行する．

仮想機械の制御はサーバが行う．まず，サーバはサービスプログラムが組み込まれた仮想機械のベースイメージを起動する．そして，定期的・またはアプリケーションが要求するときに仮想機械の実行状態を取得する．また，サーバを復元する際には，取得した実行状態から再生を行う．仮想機械の終了は，複数のサーバが合流する際に，以降サーバとして振る舞わないノードがサービスを終了するときに使用する（**表 2**）．

3.7 実行状態の共有

サービスプログラムを実行している仮想マシンの実行状態は，サステーナブルサービスに参加しているすべてのノードを利用して保存される．参加ノードは広域に分散して配置されていることを想定しているので，各ノードに実行状態を保存するということは，広域分散ストレージを実現するということを意味する．広域分散ストレージに関しては，これまでに多くの先行研究が発表されている．しかし，既存の実現方式は，特定のファイルを高速に発見するということに主眼が置かれている．それに対し，サステーナブルサービスの実現のためには，ストレージに保管されているすべてのデータを収集するという動作が求められる．なぜなら，サステーナブルサービスにおいては，実行状態は仮想マシンの実行状態はスナップショットの差分として各ノードに保存されているためである．既存の実現方式はそのような使い方を想定していないので，単純にそれらを適用してしまうと，収集すべきスナップショット差分の個数に比例して所用時間が増加してしまうという問題がある．

我々は，保管されているすべてのデータの収集を効率的に実現することに特化して設計された広域分散ストレージの開発を行っている[1)]．この方式では，保存すべきデータは分散配置され，すべてのデータを保持しているノードは存在しない．しかしながら，すべてのノードについて，ノード間に張りめぐらされているオーバレイネットワーク上で隣接しているノードにさえアクセスできれば，すべてのデータが必ず揃うように複製が配置されている．また，この方式は耐故障性について考慮されており，障害が発生している場合でもデータへの到達可能性がなるべく維持されるように設計されている．さらに，高速なデータ配布や，各ノードにおけるストレージコストの軽減についても考慮されている．

3.8 障害検知

提案システムにおける障害とは，サーバが一部のサーバ候補から通信不能になるかサービスに異常が発見された状態を指す．通信状態の監視方法は定期的にノードへの接続を試みることで行う．接続に失敗したノードはノードリストから削除され，通信不能なノードのリストへと移される．通信不能なノードは再び通信可能になるまで監視を行う．

通信不能に陥ったノードがサーバであれば，次のサーバ（以降，**次期サーバ**）の選出を行う．サーバ候補であれば，ノードリストからの削除のみを行う．通信が回復した場合，互いがサーバであれば必要に応じて合流作業へと移り，サーバ候補であればノードリストの管理だけを行う．

サーバ障害が誤って検出される場合が考えられる．サーバ障害が誤って検出された場合，元のサーバが再び発見されるまでサーバとして振る舞う．発見後はサーバの合流・あるいはアプリケーションレベルでの一貫性制御が行われ，致命的な問題は発生しない．

3.9 サーバの複製・復元

サーバに障害が発生すると，サーバ候補は通信可能なノードどうしで，次期サーバの選出を行う．選出された次期サーバは実行状態の収集を行い，サーバの復元を行う．最新の実行状態を収集できない場合は，収集できた実行状態の中で最新のものを再生する．サーバの復元機能は，障害発生時以外にも利用することが可能である．たとえばサーバの負荷が高まりサービスが不安定になっている場合，その負荷分散のためにサーバを複製することが考えられる．サービスプログラム側にデータなどの一貫性制御の機構が備わってい

れば，提案システムが提供する機能でサーバの複製による負荷分散が達成される．また，サーバが複数存在していると同じ ID を持つ異なる実行状態が生成される可能性がある．しかし，実行状態名にはユニークなサフィックスが付加されているため，目的の実行状態を探すことは可能である．次期サーバは障害検知されたサーバの実行状態だけを収集し，再生する．

次期サーバは優先度順で決定される．次期サーバ選出アルゴリズムは，優先度の決定方法は入れ換え可能である．選出のアルゴリズムを工夫することで，分散システムにおいてより相応しいリーダを選出するアルゴリズムに関する研究は様々行われてきた[2),4),13)]．現在は静的に割り振られた優先度に基づいて次期サーバを選出している．

3.10 サーバの合流

障害によってサーバがある場所で復元されるが，障害から復帰すると，サービスによっては障害復旧時にサーバが複数存在するとサービスに支障をきたす場合がある．また，支障をきたさない場合でも，必要以上に計算リソースを消費してしまう場合がある．そのため，複製されたサーバは合流できる必要がある．サーバ自体は仮想機械上で実行されているので，障害回復を検出できれば，仮想機械を終了させることでサーバを 1 カ所だけにすることは容易に実現できる．つまり，サービスの処理による仮想機械内のデータの更新が行われなければ，サーバを 1 つ残して他を終了させることで，上述の問題は防ぐことができる．

しかし，多くのサービスはサービスが保持するデータの更新・削除を繰り返しながら動作している．サーバが複数存在する状態では，このデータが複数存在していることになり，複数箇所で更新される状態になる．つまり，サーバが合流しようとするとき，分岐したデータを 1 つに戻さなくてはならない．しかし，サービスが扱うデータは固有のデータ構造を持っていることが多いので，提案システムとして汎用的なデータの結合機能を提供することは困難である．本研究においては，適用したいサービスに精通した技術者によって記述されたスクリプト（以降，補助スクリプト）が用意されることを前提とした設計を行う．サービスプログラムに依存している箇所は，サービスが扱うデータの保管場所と，データの差分抽出・データの結合の手続きの部分である．

補助スクリプトはサービスが扱うデータを，2 つのサーバ間で結合する役割を果たす．提案システムからは，障害の回復と発見されたサーバの IP アドレスとポート番号を通知するメッセージを補助スクリプトへと送る．補助スクリプトが行うことは，データのアーカイブ・データの転送・データの差分抽出・データの結合の 4 つにまとめることができる．補助スクリプトはサービスが動作している仮想機械上で動作させる．提案システムとの通信は，ホスト OS 上で動作している提案システムプログラムとのメッセージ交換で行われる．提案システムは補助スクリプトに，障害回復によって発見されたサーバの位置を知らせるメッセージを送り，合流作業が完了したことを返信する．このメッセージを受け取ると提案システムが仮想機械を終了させる．

合流できるサービスは，データの更新がまったくないサービスかデータ更新間に依存関係がないサービスである．前者は障害発生時にサーバが複製されても問題が生じることはまったくない．後者は実行状態取得のタイミングによって，複製されたサーバで更新されたデータが失われても，その影響が他へ波及しない．たとえばメールサーバや，複雑な処理を行わない HTTP サーバなどがそれにあたる．今後その他のサービスについても合流可能かどうかを検討し，本設計における合流可能なサービスの明確な切り分けを行う．

3.11 サステーナブルサービスと補助スクリプトの相互作用

サーバプログラムにとっては，サステーナブルサービスツールキットの存在は透明であるため，サーバプログラムが直接ツールキットを利用することはない．しかし，前述の補助スクリプトに関しては，ツールキットとの通信を必要とする場合がある．そのため，本ツールキットでは以下のような相互作用を行う機構を提供する．

ツールキットから補助スクリプトへ ツールキットは，サーバの合流が発生した際に補助スクリプトを起動する．補助スクリプトは，サービスを実行する仮想マシン内で実行されている監視プロセスによって起動される．サーバの合流が必要であると認識すると，サステーナブルシステムは監視システムにそのこと通知し，スクリプトを起動する．監視プロセスには，合流時に起動するべきスクリプトをあらかじめ指定しておく必要がある．

補助スクリプトからツールキットへ 補助スクリプトはサーバの合流に際して，サステーナブルシステム全体の情報を知る必要がある場合がある．そのような情報としては，最も間近に実行状態が取得された時刻，最も間近にすべてのサーバ候補がオンラインだった時刻，などが想定される．また，補助スクリプトの動作によっては，ツールキット

側に実行状態の取得を促すことが考えられる．情報の取得は実行状態の保存要求は，監視プロセスに問合せを行うことで実現される．

3.12 議　　論

サービス内容の監視

サービス内容の監視方法としては，サーバ候補がサーバに対してサービス要求を行い，その応答を検査する方法が考えられる．しかし，サービス要求はサービス固有のものであるうえ，応答の正しさの検証は非常に困難である．サービス内容の監視はサービスごとに検証機構が用意できる場合に可能となる．

本研究では現在サービス内容の監視機構は未実装である．サービス内容の監視方法が提供されているサービスであれば，上述の方法によって検証を行い，障害検出の精度を高めることができる．

障害のセマンティクス

提案システムは実行状態が取得された時点への復元を行うことができる．しかし，サーバが正常にユーザからの要求を処理したが，ユーザへの応答が障害によって失敗した場合，その処理が終了した時点での実行状態を取得できないことがありうる．つまり，この処理が正常にユーザへ応答を返したか否かを，復元されたサーバが知ることはできなくなってしまう．現段階では，この問題によって，どのような処理のセマンティクスも保証することができなくなってしまっている．

解決策としては，実行状態が配布された時点で，サーバがどの時点での実行状態の永続化が完了しているかをユーザへ知らせることでデータ更新を確定する．これは primary-backup replication[10] とほぼ同等の動作を行うことで，実行状態が保存されたことをユーザ側に認識させることができる方法である．このことによって更新が確定していない期間の要求に関して，ユーザからの要求が正常に処理できなかったことを，ユーザは知ることができるようになる．したがって，処理のセマンティクスの保証が可能となる．しかし，サーバが復元された際に，データの一貫性を保証することができないため，処理の正しさの保証が困難である．今後この問題について検討する．

Primary-backup replication のようなアプローチによる一貫性制御手法の例としては，FT-CORBA[7] の Passive replication などがある．

4. 実　　装

これまでに説明した設計に基づいたプロトタイプシステムの実装を行った．本章ではその実装について述べる．

4.1 ノ ー ド

サーバ・サーバ候補それぞれ独立の役割と，ノードとして共通の役割を分割して実装を行った．ノードとしての役割は制御メッセージ（ファイルの要求など）の送受信，ノードテーブル・ファイルテーブルの管理，基本的な通信用の手続きなどを記述した．サーバはノードの管理とは独立の通信状態を知らせるメッセージ（ハートビートメッセージ）の送信，仮想機械の制御，実行状態の配布・収集の手続きを記述した．サーバ候補は，サーバ選出，サーバ監視の手続きを記述した．

4.2 仮 想 機 械

提案システムの実現には SBUML（ScrapBook for User-Mode-Linux）[16] を用いた．SBUML は，仮想機械モニタの一種である UML（User-Mode-Linux）を拡張し，UML 上で動作する仮想機械の実行状態をファイルの形で永続化することを可能にしている．さらに，2 つの実行状態から差分を抽出することが可能であり，差分を結合することで元の実行状態に復元することが可能である．SBUML はまた，外部から操作可能なコマンドインタプリタを持ち，コマンドを送ることで仮想機械の起動・終了・実行状態の取得などの操作が容易に行える．これらのコマンドをプログラムから実行することで仮想機械を制御する．実行状態取得のトリガはタイマとし，定期的に実行状態を取得する実装とした．

実行状態の差分を生成できる仮想機械を用いたが，結合する差分の数が増加すると結合に時間がかかるため，差分はつねにベースとなる実行状態との差分を生成した．このことによって差分の結合は一度で実行状態を復元できる．時間の経過とともに差分を保存するための領域が大きくなるので，閾値を設定し，閾値以上の領域を消費した場合には，ベースとなる実行状態の更新を行い，それ以前の差分情報を削除する．ベースとなる実行状態の更新はノード間でメッセージの交換と差分の収集を行うことで実現した．メッセージには新たな実行状態を作成した差分のファイル名を含め，その差分を持っていないノードはその差分を収集する．

4.3 障 害 検 知

設計ではノード間の通信状態を監視し，その情報をノードリストで管理する方法を述べた．これはノード共通の機能として実装した．本実装ではさらに，サーバの通信状態を通信可能な全ノードがいっせいに知ることができるように，サーバから全サーバ候補へ向けた定期的なメッセージ（ハートビートメッセージ）を送り，サーバ候補はこのメッセージを待ち受ける．サーバ候補はハートビートメッセージの待ち受け時間

を設定し，その時間切れによってもサーバの障害を検知する．

サーバの障害が検知されると，次期サーバの選出を行う．本実装ではすべてのノードがあらかじめ固定の優先度を与えられているので，ノードリストにあるノードどうしで優先度比較を行って次期サーバを決定する．

4.4 サーバの複製・復元

次期サーバが決定すると，次期サーバとなったノードは，差分情報の収集を行った後，通信可能なすべてのノードへ自ノードの IP アドレスとポート番号を知らせるメッセージを送信する．このメッセージを受け取ったサーバ候補は，新たなサーバに対してハートビートメッセージの待ち受け状態に入る．次期サーバは仮想機械の実行状態を再生して，以降サーバとして振る舞う．

負荷分散などを目的としたサーバの複製も，サービスプログラムが要求する一貫性が保たれれば現在の実装で可能である．複数のサーバが存在することを許すサービスとしては更新がない Web サービスなどが考えられる．

4.5 サーバの合流

サーバの合流は，複数のサーバが同一ネットワーク内に存在することを許さないサービスに対して行う．補助スクリプトを記述し，サービスの扱う静的なデータのアーカイブ・差分生成・結合を行えるように仮想機械上で動作させ，ホスト OS 上で動作しているシステムから送られるメッセージによって，合流作業を開始する．メッセージには発見されたサーバの IP アドレスとポート番号の情報を含み，合流する（今後サーバ候補に戻る）側のノードはシステムにアーカイブの転送が終了したことを知らせるメッセージをシステムに返送する．システムはこのメッセージを受けると仮想機械を終了し，サーバ候補へと戻る．合流される（今後もサーバとして振る舞う）側のノードは，アーカイブの転送を待ち受け，受け取ると補助スクリプトがデータの展開と結合を行う．システムとしてはサーバとして振る舞い続ける．合流する方向は発見されたサーバどうしが優先度を比較することで，優先度が高い方向への合流とした．

サービスごとに補助スクリプトを記述することは，サービスレベルでの一貫性制御を提案システムの役割から切り離すことである．システムレベルでの一貫性制御方法では文献 8) に詳しい．

5. 実　　験

実装したプロトタイプシステムを用いて，基本的な性能の計測を行った．本章ではその結果について示す．

5.1 準　　備

プロトタイプ実装を用いて，既存のメールサーバのへ適用実験を行った．実験によって，それぞれ起動にかかる時間と，障害発生からサーバが復元されるまでの時間を計測した．実験に用いたツールキットは Java 言語によって記述され，その規模はおよそ 1000 行である．メールサーバには補助スクリプトを組み込み，複数に分裂した場合でも合流を可能とした．補助スクリプトは Ruby 言語によって記述され，行数はおよそ 80 行である．補助スクリプトを作成することで，システムからのメッセージの受信，メールディレクトリのアーカイブ，複数のメールボックスの結合を可能にする．合流可能なサービスとするための具体的な変更は補助スクリプトのみであり，サービスプログラムにはいっさい変更を加えることなく実現した．

実装されたプログラムでは，システムに参加するノードの情報はあらかじめお互いが知っているという前提をおいた．そのため，実行プログラムにおいて，ノードに参加ノードすべての IP アドレス・ポート番号・優先度・開始時点のサーバの情報を渡す．優先度をあらかじめすべてのノードが既知の状態にすることで，通信可能なサーバ候補との優先度比較によって次期サーバを決定できる．個々のサーバ候補はサーバの障害検出時にそれぞれ通信可能なサーバ候補との優先度比較を行い，最も優先度が高いサーバ候補が次期サーバとなることを通信可能なすべてのサーバ候補に通知する．この通知を受け取ったサーバ候補は，次期サーバが決定したので次期サーバ選出作業を終了し，次期サーバを新しいサーバとして設定する．

実行したいサービスプログラムは，あらかじめ SBUML 内で構築しておき，メールサーバでは補助スクリプトも組み込み，その実行状態を保存する．この実行状態はシステム開始以前にすべての計算機が保持し，これをベースイメージとする．サーバが起動する前にすべてのサーバ候補を起動し，その後サーバを起動する手順でシステムを起動する．

予備実験によって，最大性能で毎分およそ 1,000 通のメールを受付けることができることが分かった．そこで，高負荷時の設定として毎分およそ 800 通のメールをベンチマークプログラムを使用してサーバにかける負荷とした．また，仮想機械が実行状態を取得している時間は仮想機械を一時停止しているため，サーバ

の稼働率が低下する．そのため，高負荷時には稼働率を上げるために，実行状態の取得間隔を長くとる必要がある．実験では，高負荷に耐える設定として予備実験の結果から間隔を 30 秒とした．

実験は 3 台の計算機を使用した．1 台が提案システムを実行し，2 台がサービスを利用するクライアントとする．ベンチマークプログラム postal[14] からメールを送信し，スプールが溢れないように定期的にスプールをクリアするスクリプトをサーバ側で実行させた．この状態においてサーバが動作し続けられることを，評価結果によって示す．

以上の環境の下で，本システムの実行時間のボトルネックとなっていた仮想機械の実行状態取得時間，差分情報のデータサイズ，実行状態復元を含めた仮想機械の起動時間を評価した．さらに，メールスプールに保存するメールのデータサイズを変化させたときの合流処理にかかる時間を評価した．

5.2 動作の説明

メールサーバ Postfix[15] を SBUML 上で動作させ，補助スクリプトを実装することで合流可能なメールサーバを実現した．実際にメールサービスを利用するユーザは，メールサービス自体が停止しているようには見えないため，通信不能なユーザ宛のメールであっても送信できるように見える．宛先に届けられないメールはサーバのキューに溜められ，合流の際にデータが結合されることで，宛先ユーザが閲覧可能になる（動作の流れ図：**図 2**，**図 3**，**図 4**）．

サーバが復元された際の新しいサーバの名前解決は，今回ローカルエリアネットワークでの実験であったため，同一の IP アドレスを持つ仮想機械を起動させることで行った．実際の広域ネットワーク上での名前解決方法としては，DDNS（ダイナミック DNS）を利用する方法や，提案システムに対する DNS 問合せを代替して答える機構をユーザ側に用意する方法などが考えられる．

Postfix は/var/spool/mail 以下にユーザ名のファイルを作り，その中に追記する形でメールを保存していた．そのため，合流の際には/var/spool/mail をアーカイブして送信し，合流先で差分を抽出して差分のみを追記することでデータの結合を行った（**図 5**）．しかし，この方法ではメールがすでにユーザによって取り出されていた場合に，合流元に同じメールが残っていると，2 通同じメールが届いてしまう恐れがある．この事態を起こさないためにサーバが複製された時点のタイムスタンプを保持し，それ以降のメールのみを結合する．メールサーバにおいては差分の抽出と，サーバが複製されたときの時間管理さえできていれば合流させることが可能である．

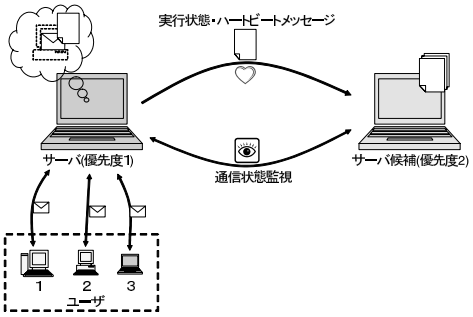


図 2 メールサービス開始時
Fig. 2 Behavior of mail service on start-up.

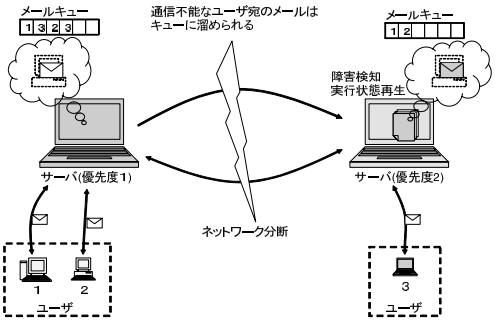


図 3 メールサービス分断時
Fig. 3 Behavior of mail service on its split-up.

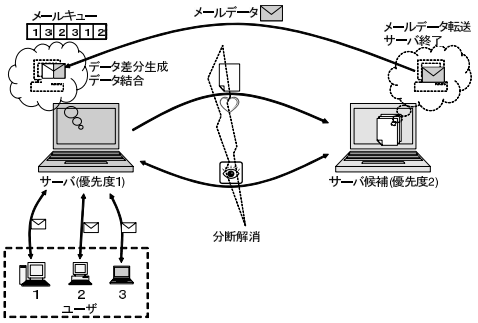


図 4 メールサービス合流時
Fig. 4 Behavior of mail service when recovering itself from split-up.

5.3 評　　価

今回我々が実現したサステーナブルなメールサービスの基本的な性質を確認するために，**表 3** に示した実験環境を用いて計測を行った．今回評価を行ったのは，サーバ負荷の増減がシステムに与える影響，スナップショット取得間隔がシステムに与える影響，および，メールスプールの合流に要する時間である．

```
/*メールサービス用補助スクリプト*/
/*メールスプールの差分抽出と結合*/
merge_mailbox(my_mailbox, your_mailbox){
  my_mailbox と your_mailbox の複製時点以後のメールを差分として抽出;
  return diff_mail;
}
merge_all(my_mail_spool, your_mail_spool){
  各メールスプールからユーザ毎にメールボックスを比較，結合
  merge_mailbox(my_mailbox, your_mailbox);
}
```

図 5 メールサービス用補助スクリプト疑似コード（一部）
Fig. 5 Pseudo code of helper script for mail service (excerpt).

表 3 実験環境
Table 3 Experimental environment.

CPU	Intel Xeon 3.60 GHz x 1
メモリ	1 GB
NIC	1000 base-T
ホスト OS	Linux 2.4.27
仮想機械	SBUML 062906a
メールサーバ	Postfix 1.1.11
メールベンチマーク	postal 0.62

サーバ負荷の影響

サーバに与える負荷が本提案システムに与える影響を調べるために，負荷を変化させた場合における実行状態取得時間，および，差分データサイズを計測した．サーバへ負荷を与えるためには，メールベンチマークプログラムによって生成された，最大 10 KB までのランダムな文字列を内容とするメールを与えた．その結果をそれぞれ**図 6**，**図 7** に示す．なお，本計測においては，実行状態取得間隔は 30 秒に固定されている．また，仮想マシン内の仮想ハードディスクがメールで溢れるのを防ぐために，定期的にメールスプールを空にするプログラムを動作させた．

計測の結果より，実行状態取得時間および差分データサイズはサーバの負荷が増大するにつれて大きくなることが確認された．特に，差分データサイズに関しては，ほぼ負荷に比例することが確認された．これは，メールサーバの場合は，メールを受信するたびにその内容がハードディスクに保存されるため，負荷の量がハードディスクの内容の変更量とほぼ一致したためであると考えられる．

実行状態取得時間と差分データサイズの両方に共通して見られる性質として，実行を開始した直後は急激に増加する傾向を見せるが，次第に収束するという点があげられる．これは，サステーナブル基盤ツールキットが継続的に動作可能であることを示唆している．

実行状態取得間隔の影響

次に，実行状態の取得間隔を変化させた場合の実行状態取得時間，および，差分データサイズを計測した．その結果をそれぞれ**図 8**，**図 9** に示す．計測に際しては，サーバへの負荷は 800 通毎分に固定し，その他の設定は 1 つ前の実験に準じた．

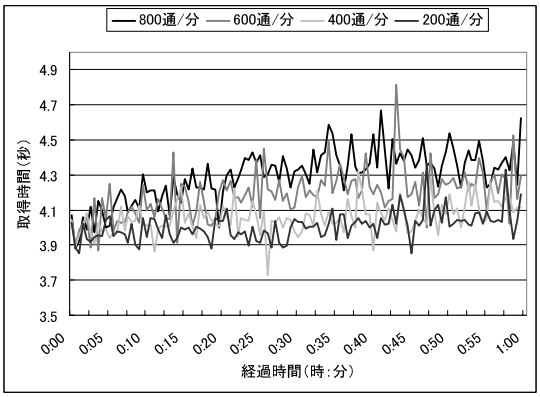


図 6 メール送信数 v.s. 実行状態取得時間
Fig. 6 Number of messages v.s. time for taking snapshot.

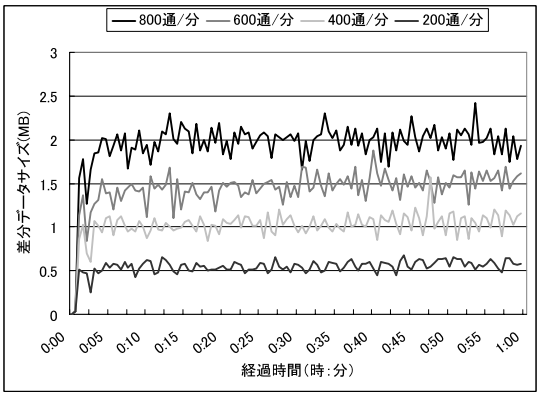


図 7 メール送信数 v.s. 差分データサイズ
Fig. 7 Number of messages v.s. size of delta-compressed snapshot.

計測の結果より，実行状態の取得時間および差分データのサイズは実行状態の取得間隔が短くなるにつれて増加することが確認された．また，メール負荷の場合と同じように，差分データのサイズは実行状態の取得間隔にほぼ比例することが確認された．理由も，メール負荷の場合と同じく，実行時間の取得間隔とハードディスクの内容の変更量がほぼ一致しているためであると考えられる．

メールスプールの合流に要する時間

メールスプールに保存するメールのデータサイズを変化させたときの合流処理にかかる時間を計測し，障害発生時のダウンタイムを評価した（**図 10**）．合流時間は，メールスプールに保存されたメールのデータサイズを変更しながら合流を繰り返し行って評価した．図 10 に示す棒グラフの表すメールスプールのデータ

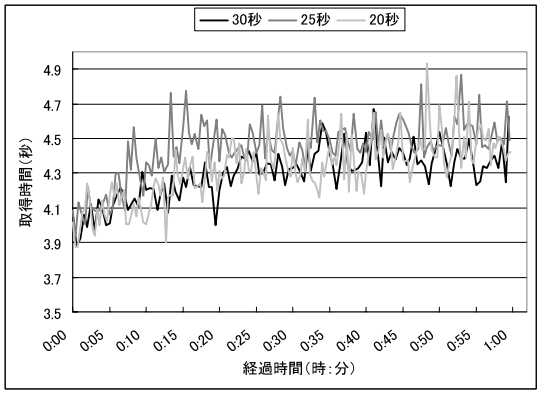


図 8 実行状態取得間隔 v.s. 実行状態取得時間
Fig. 8 Period of taking snapshot v.s. time for taking snapshot.

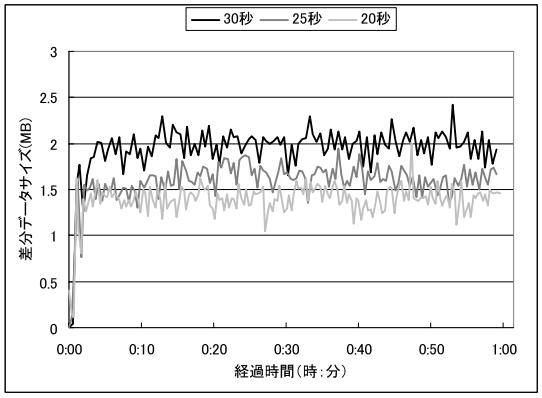


図 9 実行状態取得間隔 v.s. 差分データサイズ
Fig. 9 Period of taking snapshot v.s. size of delta-compressed snapshot.

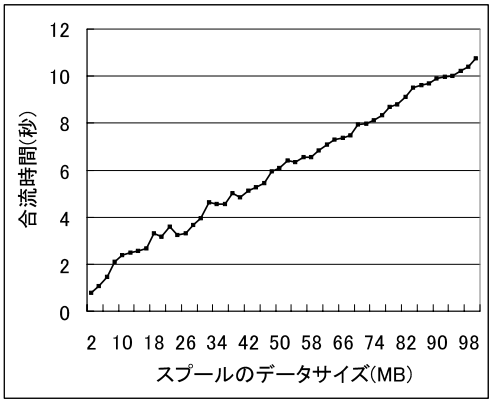


図 10 メールスプールの合流時間
Fig. 10 Time for merging mail spools.

サイズは合流後のサーバの持つスプールのデータサイズである．グラフからおよそデータサイズに正比例して合流時間が増加していることが分かる．メールスプールのデータサイズに上限が設定されていれば，合流時間も上限以上遅くなることがないことが示された．

6. 関連研究

サーバの複製を広域に分散させて availability を向上させる手法として，文献 17) ではホストサーバとリダイレクタを用いた方法が述べられている．ホストサーバはサーバの複製として動作している．リダイレクタはホストサーバの情報を管理しており，クライアントからのリクエストを関連付けられたホストサーバにリダイレクトする．この研究は，対象とするサービスを複数のホストサーバが複製し，それらをリダイレクタで関連付けることによって，サービスの提供の継続を目指すものである．また，クライアントからのリクエストは，リダイレクタを通して同じサービスを複製しているホストサーバに送られ，ホストサーバ間での整合性が保たれている．しかし，ネットワークにどのようにリダイレクタを配置すればよいかは人間が判断する必要があり，その具体的な方法については触れられていない．また，リダイレクタに障害が発生した場合は，関連するすべてのホストサーバが無駄になり，整合性も保てなくなってしまうという問題もある．本研究では，システムのどの部分に障害が起きたとしても，自律的にそれを乗り越えてサービスの提供が継続できるようなシステムを目指している．サーバの複製をすることで耐故障性や負荷分散を行うシステムにおいては，データの一貫性制御とシステムの性能はトレードオフの関係にあることが知られている．

文献 19) では，システムが持つデータの一貫性の度合いを表す指標を設け，このトレードオフに対して，ミドルウェア層で柔軟に複製間の一貫性を保つレベルを制御できるフレームワークを提案している．この方式により，システム開発者は厳密に一貫性を保って性能を犠牲にするか，性能を優先して一貫性を犠牲にするかという極端な選択を迫られることがなくなり，任意のレベルで複製間の一貫性を制御することが可能となる．同研究では，サーバの複製を前提としたシステムを対象としている．我々の提案方式では，一貫性制御と性能のトレードオフに焦点を絞らない代わりに，対象とするシステムの範囲を広くしている．仮想機械の複製を用いることで，サーバの複製を前提としたシステムのみではなく，複製が想定されていないシステムも対象とすることを目的としている．複製が想定されていないシステムであっても，上位層に影響を与えることなく，ミドルウェア層で複製可能なシステムとすることを目指している．このように，提案方式は提

供されるサービスにサステーナブル性を付与することを目的としている．

仮想機械の技術を分散システムに応用している事例として，文献 5)，11)，12) などがあげられる．これらは負荷分散を目的として，仮想機械に閉じ込めた実行環境を柔軟に扱う手法を提案している．環境の複製を仮想機械で行い負荷分散を達成する部分では，提案方式もこれらの方式のアプローチと共通している．しかし，我々はシステムに耐故障性を持たせることによって，ユーザから見えるサービスを継続させることに主眼を置いている．この点においてこれらの文献の研究目的と異なっている．

7. おわりに

本論文ではサステーナブルサービスを提供するための基盤システムとしてサステーナブルサービスツールキットを提案し，実装と評価実験を行った．既存のシステムのサステーナブルサービス化にはアプリケーションに依存してしまう問題があることを発見し，補助スクリプトを用意することで直接サービスプログラムを書き換えることなく適用できることを示した．約 14 万行の Postfix プログラムに対して，約 1000 行の提案システムと約 80 行の補助スクリプトを用意するだけで合流可能なメールサービスを実現できた．

今後は，提案システムの汎用性を高めるため，リーダ選出アルゴリズムの調査，効率的な実行状態共有方法の研究，障害のセマンティクスの解析を行う．さらに，仮想機械の実行状態取得方法の最適化や，仮想機械の高速化に関する研究を行う．また，既存のアプリケーションだけではなく，新しくサステーナブルサービスを開発するための基盤システムとして，本研究を基に，サステーナブルサービスのフレームワークを開発することを目指す．

参考文献

1) Abe, H. and Kato, K.: Sustor: Distributed Storage for Disaster Recovery Using the Small-World Model, *The 6th IEEE International Conference on Computer and Information Technology* (*CIT'06*), p.60 (2006).
2) Abu-Amara, H.H.: Fault-Tolerant Distributed Algorithm for Election in Complete Networks, *IEEE Trans. Comput.*, Vol.37, No.4, pp.449–453 (1988).
3) Adams, K.: Geographically Distributed System for Catastrophic Recovery, *USENIX LISA* (2002).
4) Afek, Y. and Gafni, E.: Time and message bounds for election in synchronous and asynchronous complete networks, *PODC '85: Proc. 4th annual ACM symposium on Principles of distributed computing*, New York, NY, USA, pp.186–195, ACM Press (1985).
5) Awadallah, A. and Rosenblum, M.: The vMatrix Server Switching, *IEEE 10th International Workshop on Future Trends in Distributed Computing Systems* (*IEEE FTDCS 2004*) (2004).
6) Chang, F., Ji, M., Leung, S.A., MacCormick, J., Perl, S. E. and Zhang, L.: Myriad: Cost-effective Disaster Tolerance, *USENIX FAST* (2002).
7) CORBA. http://www.omg.org/
8) Davidson, S.B., Garcia-Molina, H. and Skeen, D.: Consistency in a partitioned network, *A survey ACM Computing Surveys*, Vol.17, No.3, pp.341–370 (1985).
9) Gabber, E., Fellin, J., Flaster, M., Gu, F., Hillyer, B., Ng, W.T., Özden, B. and Shrive, E.: StarFish: Highly-Available Block Storage, *USENIX Annual Technical Conference* (2003).
10) Guerraoui, R. and Schiper, A.: Software Based Replication for Fault Tolerance, *IEEE Computer*, Vol.30, No.4, pp.68–74 (1997).
11) Hisayuki, M., Inoue, S., Kakuda, Y., Toda, K. and Suzaki, K.: Adaptable Load Balancing Using Network Transferable Computer Associated with Mobile IP, *23rd International Conference on Distributed Computing Systems Workshops* (*ICDCSW'03*) (2003).
12) Krsul, I., Ganguly, A., Zhang, J., Fortes, J.A.B. and Figueiredo, R.J.: VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing, *Supercomputing 2004* (2004).
13) Malpani, N., Welch, J.L. and Vaidya, N.: Leader election algorithms for mobile ad hoc networks, *DIALM '00: Proc. 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, New York, NY, USA, pp.96–103, ACM Press (2000).
14) postal. http://www.coker.com.au/postal/
15) Postfix. http://www.postfix.org/
16) Potter, R.: One-Click Distribution of Preconfigured Linux Runtime State, *VM* (2004).
17) Shenoy, G., Satapati, S.K. and Bettati, R.: HYDRANET-FT: Network Support for Dependable Services, *ICDCS* (2000).
18) 大和純一，菅 真樹，菊池芳秀：広域災害に対するストレージによるデータ保護，電子情報通信学会誌，Vol.89, No.9, pp.801–805 (2006).

19) Yu, H. and Vahdat, A.: Design and Evaluation of a Continuous Applications, *OSDI*, pp.305–318 (2000).

(平成 18 年 7 月 21 日受付)
(平成 18 年 12 月 10 日採録)

小磯　知之（正会員）

2004 年 3 月筑波大学第三学群情報学類中退．2006 年 3 月筑波大学大学院修士課程理工学研究科修了．2006 年 4 月東日本電信電話株式会社入社，現在に至る．分散システム，データマイニングに興味を持つ．

阿部　洋丈（正会員）

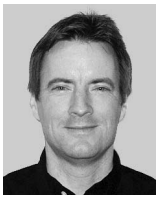
1999 年 3 月筑波大学第三学群情報学類卒業．2004 年 3 月筑波大学大学院博士課程工学研究科修了．博士（工学）．2004 年 4 月より科学技術振興機構戦略的創造研究推進事業 CREST 研究員，現在に至る．システムソフトウェア，特に分散システムとコンピュータセキュリティに興味を持つ．情報処理学会平成 16 年度山下記念研究賞，情報処理学会平成 16 年度論文賞受賞．日本ソフトウェア科学会，IEEE，ACM 各会員．

池嶋　　俊（学生会員）

2006 年 3 月筑波大学第三学群情報学類卒業．同年 4 月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博士前期課程に入学，現在に至る．分散システムに興味を持つ．

石川　宗寿（学生会員）

2006 年 3 月筑波大学第三学群情報学類卒業．同年 4 月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前期博士課程に入学，現在に至る．データマイニング，セキュアコンピューティング，システムソフトウェアに興味を持つ．



リチャード　ポッター

1999 年メリーランド大学カレッジパーク校博士課程修了．Ph.D. in Computer Science．2000 年より科学技術振興事業団さきがけ研究員．2004 年より科学技術振興機構戦略的創造研究推進事業 CREST 研究員，現在に至る．仮想マシンの管理ツールの実現，およびエンドユーザプログラミングの研究に興味を持つ．ACM 会員．

加藤　和彦（正会員）

1985 年筑波大学第三学群情報学類卒業．1992 年博士（理学）（東京大学大学院理学系研究科）．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 年筑波大学大学院システム情報工学研究科教授，現在に至る．2003 年より情報処理学会システムソフトウェアとオペレーティングシステム研究会主査．オペレーティングシステム，セキュアコンピューティング，自律連合型分散システムに興味を持つ．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．