2005－OS－99（2）
2005／5／25

Sustainable Serviceの実現構想

鈴木与範[†]　小磯知之[††]
阿部洋丈[†††]　加藤和彦[†,†††]

ネットワークやハードウェア，ソフトウェアなどの障害によって，ネットワーク上のサービスはしばしばサービス不能状態に陥ることがある．本研究の目的は，クライアントからの要求によってサーバの状態が更新されるような形態のインターネットサービスに対して，ハードウェア障害や通信障害などを自律的に乗り越えるシステムを設計することである．我々は自律的に環境変化を克服できるサービスを Sustainable Service と呼んでいる．本稿では，仮想機械の技術と Peer-to-Peer ネットワークを用いた Sustainable Service の実現方式について述べ，さらにそれらを発展させた内容についても考察する．

A Plan of Realizing Sustainable Service

Tomonori SUZUKI,[†] Tomoyuki KOISO,[††] Hirotake ABE[†††]
and Kazuhiko KATO[†,†††]

Network services sometimes fall into unavailable in some reason(network error, hardware error, software error, etc.). The objective of our research is to develop a system which can make survices on this system sustainable. Our system aims to cover not only static services but dynamic services. We describe a service as Sustainable Service when it can stand any error autonomically. In this paper, we introduce the method to realize Sustainable Service by using virtual machine technology and Peer-to-Peer networks. Moreover, we discuss advanced techniques for improving proposal system.

1. はじめに

近年，インターネットを介して提供されるサービスの重要性はますます高まってきている．しかし，障害によってサービスが提供不能に陥る事態もしばしば発生し，様々な損害をもたらしている．特に，通信障害が原因のサービス不能は，その影響が大きいにもかかわらず対処が困難である．対処を困難にしている一因は，その原因の多様性にあると考えられる．インターネットにおける通信障害は，ハードウェア障害や設定ミスなどの一般的な障害に加え，予期せぬ大量のトラフィック (ワーム，スキャン，大量のメール，Peer-to-Peer ファイル交換等) のような外的要素にも起因し得るからである．

† 筑波大学大学院システム情報工学研究科
University of Tsukuba, Graduate School of Systems and Information Engineering
†† 筑波大学大学院理工学研究科
Master's Program in Science and Engineering at University of Tsukuba
††† 科学技術振興機構
Japan Science and Technology Agency

もし複数の経路を持っていれば，一部の通信経路に障害が起きたとしても，別の経路に切替えることによって障害を回避できる可能性がある．しかし，インターネットにおける通信障害はエンドノードにごく近いところで起きやすいという傾向があり，広域的なルーティングの変更だけでは根本的な解決は難しいと言われている[4]．そのような障害によるサービス不能状態を回避するためには，サービスを提供するサーバをあらかじめ複製し，それらを広域にわたって分散配置しておくことが有効である．

インターネットにおける通信障害の回避には，CDN（Content Delivery Network）が有効に働く場合がある．CDN では，専用の高速回線で接続されたスレーブノードをネットワークトポロジ的に離れた場所に配置し，マスターサーバから配信されたデータをクライアントに提供している．CDN の第一の目標はデータ転送能力の向上であるが，その副次的な効果として，障害に対する耐性の向上も挙げられる．これは，いくつかのスレーブサーバが障害で停止していたとしても，生き残っている他のサーバによってコンテンツの提供を継続できるためである．しかし，動的にサーバの状

態が更新されるコンテンツの場合は，この仕組みはうまく働かない．これは，CDN ではマスターサーバとマスター/スレーブ間の接続が更新の単一故障点になっているからである．

CDN のような方法で通信障害等を回避出来たとしても，そもそも現在の人手による管理方法には限界があるという見方がある．近年のネットワークインフラの拡充や，個人のコンピュータ利用の浸透などに伴って，システム管理にかかるコストは爆発的に増加している．そこで，コンピュータがそれぞれ自律的に障害などの環境変化に対応し，互いに協調動作することで，全体としての正常さを保とうとするアプローチが注目を集めている[6)12)15)17)]．

我々は，特定のシステムを対象とした自律協調化による環境変化への対応を目指すのではなく，汎用的な自律協調型基盤システムの構築を目指す．サービスは，この基盤システムを利用することで，容易に環境変化に対応することが出来るようになる．本稿では，以降自律的に環境変化を乗り越えることが出来るサービスを **Sustainable Service** と呼ぶ．

本システムの実現方式として，我々はサービスを提供しているサーバの状態を分散環境で共有し，障害発生時にはサービスの復旧にこの情報を利用する方法を採る．サーバの状態をサービスプログラムに手を加えずに取得するために，仮想機械の技術を用いる．また，分散環境で情報の共有をするためには，Peer-to-Peer ネットワークを用いる．仮想機械は実行状態を共有できる形で永続化する機能を有するものを用い，Peer-to-Peer ネットワーク上で仮想機械の実行状態を共有することで提案システムの実現を目指す．

2. 関 連 研 究

サーバの複製を広域に分散させて availability を向上する手法として，文献 11) 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．これは，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．リダイレクタはサービスを複製している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．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，関連する全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 13) 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．この研究では，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 sustainability を付与することを目的としている．

3. 提 案 手 法

3.1 概　　観

本提案システムは，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，この計算機群上で実行される仮想機械上で動作している．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，仮想機械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は Peer-to-Peer ネットワークを構成しており，必ずしも全対全の通信ができなくてもよい．システムは，サービスプログラムの状態をこの Peer-to-Peer ネットワーク上で共有し，障害によってサービス不能状態に陥った場合は，サーバ候補群から新たなサーバが選

ばれてサービスの提供を継続する．

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仮想機械の実行状態として取得し，サーバ候補の構成する Peer-to-Peer ネットワーク上で共有する（**実行状態の共有**）．

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，Peer-to-Peer 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**）．

以下では，Sustainable Service を実現するためのそれぞれのステップについて，より詳しく述べていく．

3.2 実行状態の共有

提案方式では，障害発生後にシステムでサーバを復元し，サービスの提供を継続する．このため，サーバは，サービスプログラムの状態を仮想機械の実行状態として取得し，サーバ候補の Peer-to-Peer ネットワーク上で共有しておく．

サーバは，定期的，またはサービスプログラムの要求に応じて，仮想機械の実行状態を取得する．そして，前回取得した状態との差分を作成して，サーバ候補群の構成する Peer-to-Peer ネットワークを介してそれを共有する．

実行状態の共有は，障害が発生した場合にサービスプログラムの実行状態の復元を可能にするためのステップである．このため，各実行状態の差分について，一定数以上の複製が必ず作成されるようにし，一部のサーバ候補に障害が起きてもシステムから完全に失われてしまうような実行状態の差分がないようにする必要がある．

一般的に，実用的な仮想機械の実行状態は数百 MB ～数 GB のサイズがあり，広域ネットワークを介して何度も転送を繰り返すことは現実的でない．しかし，一部の仮想機械モニタには，ある時点からの差分という形で実行状態を取り出す機能を備えているものがある．そのような機能を利用することで，実際にネットワークを介して転送される実行状態のサイズを削減することが可能である．

このような機能を備えている仮想機械モニタとして，SBUML（ScrapBook for User-Mode-Linux）[10] が挙げられる．SBUML は，仮想機械モニタの一種である UML（User-Mode-Linux）[3] 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．また，仮想機械の実行を一時停止し，実行状態のスナップショットを作成し，実行を再開するという一連の操作を数秒で終えることが可能である．このため，我々は SBUML を用いた本提案方式の実装を進めている．

仮想機械による実行状態の永続化は，ハードウェアの状態や OS カーネルの状態まで全てを含むために汎用性が高い．その反面，サービスの提供の継続には本質的に必要のない情報をも含み得るため，実行状態の差分が大きくなってしまう可能性がある．そこで我々は，SoftwarePot システム[5][16] を改良して，本提案方式に適用することも検討している．SoftwarePot システムは，移送可能な仮想ファイルシステム階層の中にプログラムを封じ込め，その状態でそのプログラムの実行を可能にするシステムである．現在，SoftwarePot 内に隔離実行されているプロセスのチェックポイントを行う機構が開発されている（文献 14)）．これを利用することで，仮想機械を用いるのに近い汎用性を保ちつつ，コストのかからないシステムを実現し得ると考えられる．

3.3 サーバの監視

各サーバ候補は，サーバが正常に動作しているかを監視する．しかし，全てのサーバ候補がサーバにメッセージを送信するという方式の監視を行うと，サーバにもネットワークにも余計な負荷がかかってしまう．このため，サーバ候補の構成する Peer-to-Peer ネットワークなどを利用した，サーバへ負担をかけない監視を行う必要がある．

Peer-to-Peer ネットワークを利用した監視の例としては，**図 1** のような方法が考えられる．これは，サーバ候補が自分の接続しているサーバ候補全員にサーバの監視結果を問い合わせ，その結果の和を自分の監視結果とするものである．この結果は有効期限付きでキャッシュされ，他のサーバ候補から問い合わせがあった場合はそれを基に応答する．図 1 では，サーバ候補 a がサーバの生存確認メッセージを接続しているサーバ候補 c，d に送信している．サーバ候補 c は，サーバが生存しているというキャッシュを持っているため，このメッセージに返答を行う．しかし，サーバ候補 d

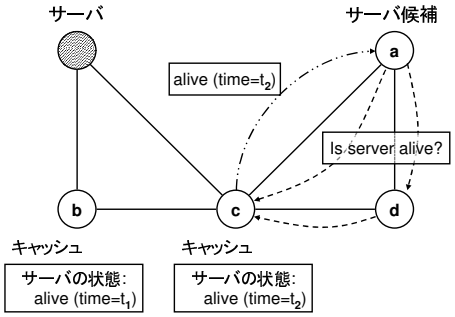

図 1 サーバ候補によるサーバの監視の例
Fig. 1 Example of erver monitoring by server candidates.

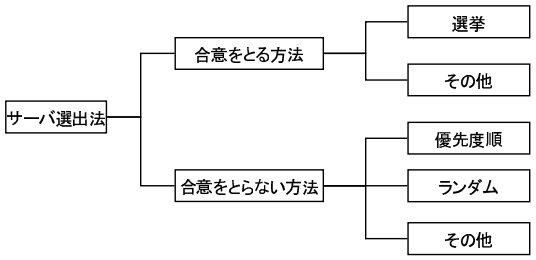

図 2 選出法分類図.
Fig. 2 Classification of election algorithms.

にはそのようなキャッシュが残っていないため，このメッセージを契機として隣接するサーバ候補 c にサーバの生存確認メッセージを送信する．この方法では，サーバに直接生存確認を行うのは実際にサーバと隣接しているサーバ候補だけなので，サーバにかかる負荷が少なくてすむ．また，サーバに直接確認を行うサーバ候補がダミーのサービス要求を送ることによって，サービス内容の確認も行うことも可能になる．

3.4 次期サーバの選出

提案システムでは，サーバ候補がサーバやサービスの障害を検知すると，次期サーバの選出が行われる．次期サーバの選出に際しては，ネットワークや計算機環境などが安定したノードが選ばれることが望ましい．このような性質を持つノードをサーバ候補の中から選出できるアルゴリズムを用いて，提案システム上での最適な次期サーバを選出する．

選出方法には，あらかじめノードに与えた優先度順に行う方法や，ランダムに選ぶ方法，または選挙等を行いノード間の合意を採る方法等が考えられる（**図 2**）．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ーを選出する研究は様々行われてきた[1)2)7)]．しかし，実際のシステムで運用している例は少なく，ネットワークの分断を考慮したり，比較的少数のノード間での選出を前提としたアルゴリズムの研究は見られない．また，我々の提案システムは，次期サーバの選出方法によっては availability や耐故障性に影響が出ることも考えられる．そのため，availability を落とさに耐故障性を向上できるような，安定したノードを選出できる負荷の小さいアルゴリズムが必要である．ランダムに選出する方法や優先度順に選出する方法は，選出を高速に行うことができ，実装が容易であるという利点はあるが，ネットワークが安定していないノードが選ばれたり，locality が考慮されずにノードが選ばれてしまうことが考えられる．このような事態を避けて最もサーバに相応しいノードを選ぶためには，適切な次期サーバの基準の設定と，この基準にそってノード間で合意を採れるアルゴリズムの採用が必要である．しかし，ノード間で合意を採るためには，メッセージ交換による通信の増加などが問題となる．このため，通信のコストと選出の精度のトレードオフを考慮したアルゴリズムの開発が必要である．

3.5 実行状態の収集

次期サーバに選出されたサーバ候補は，サーバ候補の構成する Peer-to-Peer ネットワーク上に散らばっている実行状態の差分の収集を行う．その後，それらを使用して障害が起きる直前のサービスプログラムの状態を復元し，サービスの提供を継続する．

障害が検出されて実行状態を収集する際には，ネットワークの分断が起きている可能性もあるため，次期サーバとその他のサーバ候補とが直接通信できるとは限らない．このため，直接は通信できないサーバ候補からも，中継するサーバ候補を経由して実行状態の差分をやりとりできるような方法をとる必要がある．我々のシステムでサーバ候補の数として想定している規模は比較的小さいので，次期サーバを起点として Peer-to-Peer ネットワーク上で flooding を用いても，これは実現可能であると考えられる．

より効率的な収集を行うために，次期サーバの選出のステップでの情報を使用することが考えられる．図 2 の分類において合意を採る方式を採用すれば，サーバ候補間でプロファイル情報を交換することになる．この情報を利用すれば，不足している実行状態の差分の位置やネットワークの接続状況が判明するため，実行状態の収集にかかるコストを大幅に削減できると考えられる．

この他にも，実行状態の差分の収集を高速化するために複数のサーバ候補から同一の差分を受信する方法や，実行状態の収集と復元を同時に進行させるために古い差分により高い優先度を付けて収集する方法が必

要になる．

3.6 サーバの複製と合流

サーバの複製

本提案システムでは，サーバに障害が発生した場合に次期サーバが選出され，サービスの提供が継続する．しかし，このサーバを複製するという機能は，障害が発生した場合に限ってのみ有効なものではない．サービスプログラムが重複を許すようなアプリケーションであれば，障害が発生していなくてもサーバの負荷分散とスループットの向上の目的でサーバを複製することは有用である．本システムでは，複製されたサービス間での一貫性制御の機構がアプリケーション側にあれば，複数サーバによるサービスの提供が可能である．

一貫性制御の機構を持たないアプリケーションに対しても，サービスに更新が起こった際にその差分を通知する機構等を用意して，容易に一貫性制御が可能になるような機構を提供することを考えている．

サーバの合流

ネットワークの分断やサービスの複製等によってサーバが複数存在するような状況では，サービス要求の変化や分断の解消等によってサーバの数を減らしたいという要求が予想される．提案システムでは，このような要求に対して，サーバ同士を合流させてサーバの数を減らす機構を提供することを考えている．

サーバ同士を合流させるためには，それらの上で実行されているサービスプログラムのサービス内容をマージする必要があるため，サービスプログラム側にマージ用の処理を用意させる必要がある．しかし，その処理さえ用意すれば，サービスプログラムはサービス要求の変化に柔軟に対応した可変個のサーバ群による Sustainable Service の提供を容易に実現できる．また，この方式では，サービスプログラムごとに本システムの対応を変更する必要がないという利点がある．

4. 議　　論

4.1 サーバプロキシの導入による効率化

これまでの提案システムでは，サービスプログラムを実行している仮想機械を公開している先に障害が発生した場合に，サービスが利用不能状態に陥ってしまう可能性があった．このため，システムの sustainability の一部が外部に依存することになってしまっていた．そこで，外部との独立性を高め，内部に閉じたより安定したシステムにするために，サーバ候補をサーバプロキシとして動作させる方法を考えている．この方法では，サービスを受けようとするクライアントは，システムのどの計算機にアクセスしてもサービスを受けられるようになる．

サーバ候補がサーバへのアクセスとその応答を中継するというこの方法は，サーバの監視のステップにおいても有効に働く．サーバ候補がサーバの応答の内容まで確認することによって，これまではダミーのリクエストでしかサーバの状態を監視できなかったが，より自然な形でサービス内容の状態まで監視することができるようなると考えられる☆．

また，サーバ候補が主導で行うタイプの監視の場合は，その方法や頻度などのパラメータによっては，逆にサーバの負荷を増大させてしまう危険性をはらんでいた．しかし，クライアントからのリクエストに対するサーバの応答状況を監視することで，サーバに新たに負荷をかけずにこれまでと同程度の精度の監視が行えるようになると考えられる．

サーバ候補をサーバプロキシとして動作させることによってサービスのスループットが低下したり，サーバの応答の内容を確認することでプライバシーの問題が発生することも予想される．しかし，上述したように，システムの独立性の向上（及び，これに伴う sustainability の向上），サーバの監視の精緻化などの利点も多いため，我々はこの方法が総じて有効であると考える．

4.2 トポロジを考慮した sustainability の向上

3.2 節において，配布する実行状態の差分の複製の数について，障害が起きても復元可能な冗長性を持たせると述べた．そのため，この方法ではシステムの sustainability が実行状態の差分の複製の数に依存していた．これを効率化する方法として，サーバ候補間のネットワークトポロジを調べ，トポロジ的に独立度の高いサーバ候補に実行状態の差分を配布する手法を考えている．この方法によって，より低いコストで高い sustainability を達成することが可能になる．

ネットワークトポロジを調べる方法としては，文献 9), 8) などが挙げられる．これらで用いられている主な手法は，グローバル IP を持った参加サイトを募り，それらの間で ICMP パケット（主に traceroute や ping など）を使用することでサイト間の経路を調べるというものである．この手法はインターネットのプロトコルに沿ったものであるため，我々も基本的にこの手法を踏襲する方針をとることにする．

しかし，最近はセキュリティ面での問題によってゲー

☆ 正常/異常の判断はアプリケーション依存のため，実際には判断の基準となるスクリプトなどをアプリケーション側で用意してもらう必要があり，この結果を監視の結果として用いることになる．

トウェイなどで ICMP パケットがフィルタリングされてしまう場合も少なくない．このような状況下では ICMP のみを用いた方法はうまく働かなくなるため，単純に ICMP を用いる以外にも方法が必要になる．

サーバ候補間のホップ数を測るための方法として，IP パケットの TTL 値をのぞく手法が考えられる．計測を行うサーバ候補間で合意を行った値に TTL を設定した IP パケットを投げ合うことで，その減少値からホップ数を測定することができる．

実行状態の共有においては，なるべく別の経路を通った先にあるサーバ候補に実行状態の差分を配布したほうが，より効果的に sustainability を向上させることができると考えられる．しかし，上記の方法では，ある 2 つの経路が同一のものであるかどうかまでは分からない．このため，ある 2 つの経路について，それらが同一のものであるかどうかを判別する方法が必要となる．

5. おわりに

様々な障害が発生したとしてもサービスを継続して提供できるようなシステムとして，Sustainable Service という新しい概念を提案した．また，仮想機械の技術と Peer-to-Peer ネットワークを用いてそれを実現する方法について述べた．

現在，仮想機械として SBUML を使用した実装を進めている．今後は，3 章や 4 章で述べた提案手法について詳細を詰め，サービスプログラムの要求により柔軟に対応できるような仕様を設計したい．

参考文献

1) Abu-Amara, H.H.: Fault-Tolerant Distributed Algorithm for Election in Complete Networks, *IEEE Trans. Comput.*, Vol. 37, No. 4, pp. 449–453 (1988).
2) Afek, Y. and Gafni, E.: Time and message bounds for election in synchronous and asynchronous complete networks, *PODC*, pp. 186–195 (1985).
3) Dike, J.: A user-mode port of the Linux kernel., *ALS* (2000).
4) Gummadi, P. K., Madhyastha, H. V., Gribble, S.D., Levy, H.M. and Wetherall, D.: Improving the Reliability of Internet Paths with One-hop Source Routing., *OSDI* (2004).
5) Kato, K. and Oyama, Y.: SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation, *ISSS* (2003).
6) Kephart, J.O. and Chess, D.M.: The Vision of Autonomic Computing, *IEEE Computer Magazine*, Vol. 36, IEEE Computer Society, pp. 41–49 (2003).
7) Malpani, N., Welch, J. L. and Vaidya, N.: Leader election algorithms for mobile ad hoc networks, *DIALM*, pp. 96–103 (2000).
8) Matthews, W. and Cottrell, L.: The PingER project: Active internet performance monitoring for the HENP community, *IEEE Communications Magazine*, Vol. 38, No. 5, pp. 130–136 (2000).
9) Paxson, V.: End-to-End Routing Behavior in the Internet, *SIGCOMM*, pp. 25–38 (1996).
10) Potter, R.: One-Click Distribution of Preconfigured Linux Runtime State., *VM* (2004).
11) Shenoy, G., Satapati, S. K. and Bettati, R.: HYDRANET-FT: Network Support for Dependable Services., *ICDCS* (2000).
12) Yokota, H.: Autonomous Disks for Advanced Database Applications., *DANTE*, pp. 435–442 (1999).
13) Yu, H. and Vahdat, A.: Design and Evaluation of a Continuous Consistency Model for Replicated Services, *OSDI*, pp. 305–318 (2000).
14) 横山陽介, 大山恵弘, 米澤明憲: SoftwarePot へのチェックポイント機構の導入, 情報科学技術フォーラム（FIT） (2004).
15) 西尾章治郎: ネットワーク共生環境を築く情報技術の創出, 情報処理, Vol. 46, No. 4, 情報処理学会, pp. 385–390 (2005).
16) 大山恵弘, 神田勝規, 加藤和彦: 安全なソフトウェア実行システム SoftwarePot の設計と実装, コンピュータソフトウェア, Vol. 19, No. 6, pp. 45–52 (2002). 日本ソフトウェア科学会 2003 年度論文賞受賞.
17) 武理一郎, 野口康生, 土屋芳浩, 荻原一隆, 田村雅寿, 丸山哲太郎: オーガニックストレージシステム ～ 自律し成長するストレージシステム ～, 信学技報, Vol. 104, No. 537, pp. 55–60 (2004).