情報処理学会論文誌 Vol.54 No.3 1109–1119 (Mar. 2013)

迅速な危機回復を目的とする大規模環境向け障害原因解析システム

永井 崇之[1,a)] 名倉 正剛[1,b)]

受付日 2012年6月22日, 採録日 2012年12月7日

概要：IT システムが大規模化する一方で，各企業では経験が十分なシステム管理者を必要な人数だけ確保できずに，障害発生時に危機回復を迅速に実現できないことが多い．迅速な危機回復のためには，障害検出から原因特定までに要する時間を短縮することが重要である．そこで本研究では，大規模 IT システムを対象に，障害事象と原因の対応をパターン化し，障害発生時にそれを利用して原因機器を特定する障害原因解析手法を提案する．既存手法では，管理対象の IT システムのすべての接続関係に対して障害事象と原因の関係をあらかじめパターン化する必要があり，大規模 IT システムで障害原因解析を実施できなかった．しかし，大規模システムであっても，一般的にある障害が発生した場合に，その障害により影響を受ける機器は限定的であり，実際の障害原因解析の際には，導出したパターンの一部しか利用されない．提案手法ではこの状況に着目し，障害発生時に障害発生機器に関連する範囲についてのみ障害事象と原因の関係をパターン化し，それを利用して障害原因解析を行う．提案手法を実装し，評価実験を行い，実際の大規模システムを想定した規模の IT システムを対象に，障害原因解析処理を実施できることを確認した．

キーワード：大規模 IT システム，危機回避，障害原因解析，障害イベント相関分析

Root-cause Analysis System for Rapid Hazard Recovery on Large-scale Environment

Takayuki Nagai[1,a)] Masataka Nagura[1,b)]

Received: June 22, 2012, Accepted: December 7, 2012

Abstract: Recently IT system grows widespread, but many companies don't have enough system administrators with abundant experience, thus recover from IT system hazard is difficult. To repair IT system quickly, it should be shorten the time to identify the root cause failure when the affected failure events are detected. In this study, we propose the root cause analysis method for large-scale IT system. On existing approach, it is necessary to prepare the patterns of correspondences between the failure events and the root cause failure, based on the topologies of overall managed apparatuses. This makes it difficult to apply existing approach for large-scale IT system. Generically, when a failure occures on large scale IT system, affected part is limited. Therefore, most of the prepared patterns is not used to identify the root cause failure. We focused on this fact. Our method generates limited patterns of the correspondences at the occurrence of the failure events. The generation is based on the topologies of the apparatuses which are related with the occurring events. And our method deducts the root cause failure by using the generated patterns. We implemented and evaluated our method. As the result, we verified that it can deduct the root cause failure in a large-scale IT system.

Keywords: large-scale IT system, hazard recovery, root-cause analysis, failure event correlation analysis

1. はじめに

IT システムの規模は年々拡大し，構成機器要素の種類も

[1] 株式会社日立製作所横浜研究所
Yokohama Reserach Laboratory, Hitachi, Ltd., Yokohama, Kanagawa 244–0817, Japan
[a)] takayuki.nagai.nu@hitachi.com
[b)] masataka.nagura.qj@hitachi.com

増大している．このため，システム運用管理者には高度な知識や豊富な経験が求められている．しかし各企業では，経験が十分な管理者を必要な人数だけ確保できずに，障害発生時に危機回復を迅速に実現できないことが多い．

IT システムに発生した障害に迅速に対処するためには，障害検出から原因特定までに要する時間を短縮することが重要である．そこで IT システム障害発生時に障害原因を解析する手法が提案されている [1], [2], [3], [4], [11], [12], [13]．これらの手法の多くは，障害事象と障害原因の対応付けに従い，原因となる機器を自動的に特定する．そのために，運用管理対象のシステムの構成を基に，事前にそれらの対応付けを知識ベースとして構築しておく．そして，障害発生時にはそれを用いて障害原因解析を実施する．

本研究ではまず，大規模 IT システムを対象にこれらの手法を用いて障害原因解析を実施する場合の問題点を分析した．その結果，次の点で問題があることが分かった．
[問題 A]障害事象と原因との関係構築処理が長時間化する．
[問題 B]構成変更が頻発すると，解析結果が適切でない状況が多く起こる．

そしてこのような問題を解決する大規模環境向けの障害原因解析方式を提案する．提案方式では，障害事象と障害原因との対応の構築処理を，障害発生時に障害発生箇所に関連する範囲にのみ限定して実施することで，大規模環境に障害が発生したときに迅速な危機回復を実現できるようにしている．以降，本論文では，まず 2 章で関連研究を述べ，3 章で本研究で対象にする既存研究である障害原因解析システムについてを述べた後で，大規模環境において障害原因の解析を実施する際の問題点を分析する．そして 4 章で提案方式について述べ，5 章で評価実験を行う．最後に 6 章で考察を行い，7 章で結論を述べる．

2. 関連研究

IT システムでは，1 つの機器に発生した障害が，ネットワークを介してその機器を利用している別の機器に伝播し，障害が同時に発生することがある．運用管理者は管理ソフトウェアを利用して，サーバ，ネットワーク機器，ストレージ装置等の複数の機器から発生した障害に関する情報を，ログファイルを監視したり，状態を監視したりすることにより，障害イベントとして受信する．管理者は，受信した多くの障害イベントの中から障害原因を示す障害イベントを特定する必要がある．障害原因に迅速に対処するためには，この作業を支援する必要がある．

IT システム障害発生時に障害原因を解析する RCA（Root Cause Analysis: 障害原因解析）技術 [1], [2], [3], [4] では，障害イベントの発生状況と障害原因の対応をあらかじめ定義しておき，障害発生時にはそれを利用して原因となる機器を自動的に特定する．Yemini らの手法 [1] や Gupta らの手法 [2] では，運用管理対象システムのすべての装置に対して，装置に発生した障害と原因の関係を縦横の軸に配置したマトリクスを管理者があらかじめ作成しておき，それを利用して障害原因を解析する．また Wang らの手法 [3] や Holub らの手法 [4] では，実際のシステムの障害発生状況を解析し，障害を同時発生させる装置群の種類と障害原因との対応を汎用化してパターンとして蓄積し，障害発生時にそのパターンを利用して原因解析を実施している．

また，障害発生時の運用管理者による運用ログ解析を支援する研究も多く行われている．Rouillard は，運用ログに含まれるイベントを時系列分析することにより，イベント発生のパターンを抽出し，着目しなくてもよいイベントを除去している [5]．敷田らは，部品間の関係を基に管理者が現在注目している事象に関連するログ情報を自動抽出する方法を提案している [6], [7]．Xu らは，コンソールログ文字列を解析し，障害発生に関連するログ情報を抽出する方法を提案している [8], [9]．Mirgorodskiy らは，複数の運用管理プロセスにおけるログを比較し，問題のある運用管理プロセスを特定する方法を提案している [10]．これらのログ解析技術は，障害発生時に障害原因を特定するものではないが，Diao らは運用管理者が記述したトラブルチケットに含まれる文字列と，障害の原因を関連付けたルールに基づいて原因特定を行う方法を提案している [11]．

障害原因を実際に特定する技術 [1], [2], [3], [4], [11] では，あらかじめルール化した解析ノウハウを蓄積しておき，それを利用して障害原因特定を迅速に実施できるようにしていた．これに対し，Bahl らは，このようなルール化した解析ノウハウを用意せず，ネットワーク上で送受信されるパケットを分析して，その依存関係からネットワーク接続を類推し，障害発生時の原因特定を実施している [12]．

3. 大規模環境での障害原因解析の課題

3.1 障害発生パターンと接続情報を用いた障害原因解析システム

2 章で述べた RCA 技術のうち，Yemini らの手法 [1] や Gupta らの手法 [2] では，障害イベントの発生状況と障害原因の対応を運用管理対象システムのすべての装置に対して定義する必要があり，管理者にとって負担が大きい．Wang らの手法 [3] や Holub らの手法 [4] では，障害装置の種類と原因との対応を汎用化しているため装置個別に対応関係を定義する必要はないが，同時発生したイベント群の関連のみに従って原因を解析しており，装置間の接続関係（トポロジ）を考慮していない．そのため，システム内の複数箇所で同時間帯に障害が発生した場合，それらにトポロジが存在しなくても同じ障害原因に起因すると見なしてしまうことがある．

そこで，工藤らの手法 [13] では障害発生装置の種類/構成要素と障害原因の対応を汎用的なパターンとして定義し（これを汎用ルールと呼ぶ），実際のトポロジにパターンを

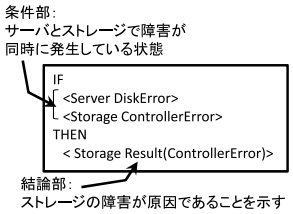

図 1 汎用ルールの例

Fig. 1 An example of general rule.

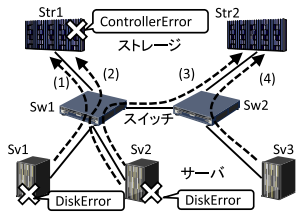

図 2 システム構成例

Fig. 2 An example of system structure.

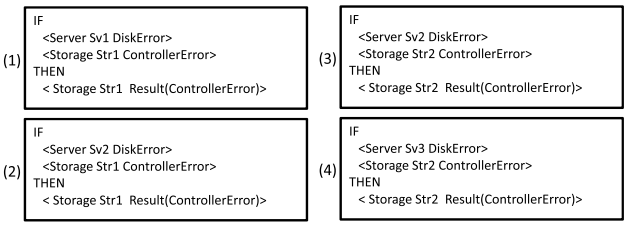

図 3 解析ルールの例

Fig. 3 An example of analysis rules.

機械的に適用したうえで，障害発生時に通知された障害イベントをマッチングさせて障害原因解析を実施している．この手法では，管理対象システムを構成する装置から障害が伝播した結果として連鎖的に通知される障害イベントを収集し，それらをパターンにマッチングさせることで，障害原因を示す障害イベントを推定している．

障害発生状況をパターン化した汎用ルールの例を，**図 1** に示す．汎用ルールは，装置内の構成要素に対して，あらかじめ IF-THEN 形式で記述しており，1 つの汎用ルールで特定の障害伝播の状況を表している．汎用ルールの IF 部には，IT システムに発生している障害イベントの情報（装置，装置内の構成要素の種類と，障害イベントの種類）を列挙している．THEN 部には，それらの障害イベント群が発生した際に，障害原因として推定できる箇所を示す障害イベントを記述している．図 1 のルールは，サーバの構成要素であるディスクとストレージのコントローラに障害が発生し，その原因がストレージのコントローラであり，その障害の影響がストレージを利用するサーバ群に波及している障害発生状況を表している．なお，この例では記載していないが，IP スイッチや FC スイッチの要素を記述することによって，装置間のネットワークトポロジも指定できる．そして，ネットワークに起因する障害伝搬状況を記述することができる．

図 2 に示すシステムでは，それぞれのサーバは，(1)〜(4) に示すような関係で，スイッチを介してストレージを利用している．このシステムに対して，汎用ルールを実際の装置の構成で発生しうる障害イベントと障害原因との対応へ変換する．図 1 の汎用ルールを図 2 に示すトポロジに合わせて変換すると，(1)〜(4) の関係に合わせて，**図 3** のような 4 個の解析ルールが構築される．

そして管理対象装置に対し，障害発生の有無を定期的に監視する．図 2 に示したシステムでは，ストレージ Str1 のストレージコントローラの故障によりサーバ Sv1，Sv2 からストレージにアクセスできなくなり，サーバ Sv1，Sv2 にもディスクエラーが発生している．Str1，Sv1，Sv2 に障害が発生していることを検知すると，検知した障害イベントを構築した解析ルールにマッチングすることによって，障害原因を導出する．なお，障害が発生しているものの何らかの原因で障害イベントを取得できていなかったり，障害イベントの取得のタイミングがずれていたりすることも考えられる．そのような状況を考慮すると，解析ルールに含まれる障害イベントのうちのいずれかが発生していれば，他のいくつかの障害イベントが欠落していても，そのルールによって指し示される装置が障害原因である可能性がないとはいいきれない．そのため，解析ルールに含まれる障害イベントのいずれかを受信した場合，そのルールの THEN 部によって指示される装置を障害原因の候補として推論する．しかし受信すべき障害イベントを受信していないため，推論結果の確からしさは低くなる．そこで解析ルールに含まれる障害イベント群のうち，実際に受信した障害イベントの割合を，その障害原因の確信度として算出する．確信度の算出後，障害発生箇所に関連する解析ルールのうち，最も確信度の高いもの（すなわち，事前に想定した障害発生パターンのうち，障害発生状況に最も適合しているもの）から順に，その解析ルールの示す障害発生箇所を，障害原因として提示する．たとえば図 2 に示すように障害イベントが発生していたのならば，ルール (1) および (2) によって Str1 が障害原因である確率が 100%になり，またルール (3) によって，Str2 が障害原因である確率が 50%になる．そしてそれらの障害原因を，確信度の高い順に障害原因候補として管理者に提示する．

また，ネットワーク機器の障害のように，障害発生時につねに検出手段や通知手段が失われており，障害イベントを取得できない場合もある．そのような状況に対応するため，イベントを取得できないことを “Unreachable” な障害イベントとして障害パターンに記述できるようにしている．

なお，別の装置に発生した障害の影響によって連鎖的に障害事象が発生した結果，障害原因とされる障害発生箇所が，別の解析ルールの IF 部に含まれることがある．この場合には，ある解析ルールが示す障害発生箇所と，それを IF 部に含む別の解析ルールを合成して障害原因解析を実施した結果も，障害原因候補に含めて提示する．

3.2 大規模・複雑な環境に適用するための問題点

企業情報システムの大規模化にともない，運用管理者が管理する機器数は飛躍的に増加する．そのような状況で障害発生時に原因の切り分けを迅速に実施する必要がある場合に，多くの機器から障害原因を自動的に特定できる RCA

技術を効果的に利用できる．

しかし，大規模環境を対象に障害原因解析を実施する際に，Yemini らの手法 [1] や Gupta らの手法 [2] のように，管理対象の IT システムの全トポロジに対して，あらかじめ障害事象と原因との対応を構築しておくことは，管理者にとって困難である．そこで工藤らの手法 [13] や，Wang らの手法 [3]，Holub らの手法 [4] のように，障害イベントと原因との抽象化された対応を利用して障害原因解析を実施することが適切である．ただし Wang らの手法 [3]，Holub らの手法 [4] は，障害イベントを発生させた装置間のトポロジを考慮していない．一般的に大規模環境では，同時期に複数の障害の発生する可能性が，小規模な環境に比べて大きくなる．そのような場合に，トポロジの存在しない装置間で障害の影響が波及するような，誤った解析が行われる．大規模環境を対象に障害原因解析を実施するためには，工藤らの手法 [13] のように，障害イベントと原因との抽象化された対応に基づいて，トポロジの存在する装置間の障害に対して解析を実施できる方法が望ましい．しかしその一方で，汎用ルールのような抽象化された対応を，実際のトポロジに合わせて具体化する必要がある．大規模システムを対象に障害原因を解析する場合は，この際に適用できるトポロジが非常に多くなる．すなわち取得する情報量と，汎用パターンを適用可能な組合せを導出するための計算回数が増えるということであるため，この手法も大規模環境での障害原因解析に適さない可能性がある．

そこで，この状況を確認するための事前実験として，工藤らの手法 [13] を対象に，汎用ルールから解析ルールを構築する処理時間を測定した．この手法では，302 個の汎用ルールから解析ルールを構築している．この際に CMDB（Configuration Management Database:構成管理データベース）から運用管理対象システムの構成情報を取得している．そこで，CMDB に保持されている装置数を変化させて測定した．なお測定は，CPU が Intel Core2 Duo E6850 3.00 GHz で，メモリを 2 GB 搭載した PC で実施した．また，このシステムは Java で動作する．ここでは，Java 1.5.0 を利用した．測定結果を，**図 4** に示す．

事前実験の結果，装置数が 10,000 台規模になると，17 時間近い処理時間を要することが分かった．この手法では，

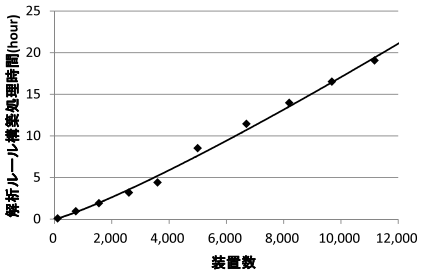

図 4 解析ルールの構築処理時間

Fig. 4 Processing time for analyzing rule construction.

システムの起動時に，その時点で CMDB に存在する構成情報を基に解析ルールを構築する．そして構成が変更された際には，変更された構成を基に解析ルールを再構築する．前述のように，解析ルールの構築が障害発生前に終了していないと，障害原因解析処理を実施できない．すなわち，このシステムは装置数が 10,000 台規模の場合，起動から 17 時間もの長時間にわたって障害原因を解析できない．

このように，RCA 技術を大規模システムに適用する際に，まず次の問題が存在する．

[問題 A] 障害事象と原因との関係構築処理が長時間化する．

さらにクラウド技術の普及により，仮想化機構を利用してクラウド上の任意の場所にサーバを移動させて実行させることが可能である [14]．このような環境では，運用時に構成の変更が頻繁に発生する．障害原因解析の実施において，障害事象と原因との関係構築処理実施後に構成変更が発生した際は，構成変更に影響を受ける汎用ルールに対して，関係構築処理を再実行する必要がある．再実行処理にどの程度の時間を要するかは構成がどの程度変更されるかに依存する．しかし，変更された構成の接続関係をすべてたどって影響を調べる必要があり，図 4 から考えるとそれほど短くないことが推測できる．そしてこの再実行の間には，障害原因の解析を適切に実施できない．したがって，RCA 技術をとりわけクラウド技術を利用した大規模システムに適用する際には，次の問題が存在する．

[問題 B] 構成変更が頻発すると，解析結果が適切でない状況が多く起こる．

一般的に構成が変更されるとそれに起因して障害が発生しやすくなると考えられる．しかし，変更の直後に発生した障害の原因を解析できない可能性が高いことになる．

これらの問題点は，工藤らの手法 [13] のように，障害イベントと原因との抽象化された対応に基づいた障害原因解析を，トポロジの存在を考慮して実施する場合に発生する．前述のように，大規模環境で障害原因解析を実施する場合には，この手法と同様に抽象化された対応に基づいた解析を，実際に存在するトポロジを考慮して実施することになると考えるのが妥当である．したがって，これらの問題点は，工藤らの手法 [13] だけではなく，大規模環境で障害原因解析を実施する場合に必ず発生する問題点である．

なお，そもそも Bahl らの手法 [12] のように，ルール化した解析ノウハウを利用しない場合には，このような問題が発生しない．しかし，解析ノウハウを利用しないため，運用管理者にとって既知の障害の発生パターンを解析できない場合がある．一般的に企業情報システムでは，システム構築時に異常時のユースケースも考慮して検証作業を実施している場合が多い．運用管理者が検証を実施済みの既知な障害の発生パターンについては，少なくとも確実に障害原因解析を実施すべきである．さらに大規模環境では，似

たような構成のサブシステムが，システム全体にいくつも存在することにより，同一の障害発生パターンにしたがった障害が起こりやすくなる．このため，まずは解析ノウハウをルール化して障害原因解析を実現する必要がある．

そこで本研究では，ルール化された解析ノウハウを利用する障害原因解析手法に対する前述の課題を解決し，大規模環境を対象に障害原因を自動で解析するための手法を提案する．次章以降にその詳細を述べる．

4. 大規模環境向け障害原因解析システム

4.1 概要

3.2 節で述べた 2 つの問題は，どちらも障害原因解析を実施するために，あらかじめ障害事象と原因との関係を，管理対象のシステム構成のすべてのトポロジに対応付けて構築しておかなければならないことに起因する．しかし我々は，発生する障害が影響する範囲は限定的になり，既存手法のようにすべてのトポロジに対してこの関係構築処理を実施する必要がない，と考えた．そしてその考えに従い，障害イベント発生時に，発生イベントと関連する装置についてのみ障害事象と原因との関係構築処理を実施することで，汎用化された障害発生パターンに基づき障害原因を解析する方式を提案する．この方式では，障害事象と原因とをあらかじめ対応づける必要がないので，3.2 節で述べた 2 つの問題が発生しない．なお本研究では提案方式を工藤らの手法 [13] を改良して実装したため，次節以降では説明の都合上この手法を基に提案方式を記述する．

まず提案方式を設計するうえでの前提が適切であるかどうかを確認するため，大規模システムで障害が発生する際の影響範囲について見積もった．この結果について 4.2 節に記述し，その後で提案方式の詳細について述べる．

4.2 障害発生による影響範囲の見積り

本節では実際に大規模システムにおいて障害が発生したときにどの程度の範囲に障害が影響するかを，大規模システムとして想定する構成を基に，検証する．

まず大規模システムとして，1,000 人規模の従業員が存在する 1 つまたは複数の大企業を対象にしたクラウドシステムを想定した．従業員 1 人あたりに個別の仮想マシンが割り当てられ，各従業員はネットワークを介して仮想マシンを利用する．この想定では，1,000〜10,000 台規模の仮想マシンを提供する必要がある．10,000 台規模の仮想マシンを収容するデータセンタの物理構成の例を**図 5** に示す．

この構成では，1 台の物理サーバに 5 台の仮想マシンが稼働しており，ユーザはそれぞれの仮想マシンをインターネット/イントラネットからシンクライアントで利用している．この想定では，10,000 台の仮想マシンが動作するために，2,000 台の物理サーバが必要になる．それぞれの物理サーバは IP ネットワークとストレージエリアネットワーク（SAN）に接続している．IP スイッチも FC スイッチも，どちらも 16 個ずつのポートを備えている．そして IP ネットワークと SAN は，物理サーバ 16 台（仮想マシン 80 台）の規模で 1 つの部署のネットワークとしてまとめられ（図中の Level 1），さらに Level 1 が 16 個単位（物理サーバ 256 台，仮想マシン 1,280 台）で 1 つの事業部門のネットワークとしてまとめられている（図中の Level 2）．それぞれの仮想マシンは，SAN を介して接続された中規模ストレージ（事業部門内ファイルシステムを想定）と，大規模ストレージ（全社ファイルシステムを想定）のボリュームをマウントしている．また，基幹の IP スイッチと大規模ストレージのポートは二重化されている．

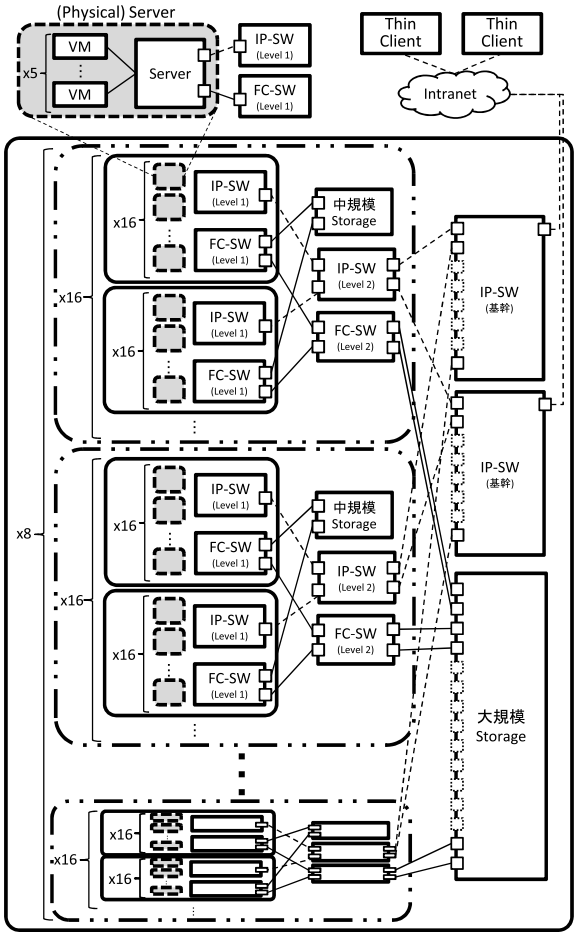

図 5 大規模な IT システムの例
Fig. 5 Example of large-scale IT system.

この構成では，物理サーバが 2,048 台，IP スイッチと FC スイッチが 137 台ずつ存在する．この構成で装置に障害が発生した場合に影響が波及する装置数を，障害発生機器種別ごとに算出した．また，工藤らの手法 [13] では，302 種類の汎用ルールから構築した解析ルールのうち，前述のように障害事象に関連する解析ルールしか障害原因解析に利用されない．そこで，それぞれの障害事象の解析を実施するために必要な，解析ルールの構築に用いられる汎用ルール数も算出した（**表 1**）．

表 1 障害発生時に影響が波及する機器数と障害原因に解析に必要な汎用ルール数
Table 1 Number of effected nodes and related general rules.

#	障害発生機器種別	影響波及機器種別	機器数	ルール数
1	VM	VM	1	6
2	Server	VM, Server	6	30
3	FC-SW(Level 1)	VM, Server, FC-SW(Level 1)	97	28
4	FC-SW(Level 2)	VM, Server, FC-SW(Level 1,2)	1,553	28
5	IP-SW(Level 1)	VM, Server, IP-SW(Level 1)	97	28
6	IP-SW(Level 2)	VM, Server, IP-SW(Level 1,2)	1,553	28
7	IP-SW (基幹)	VM, Server, IP-SW(Level 1,2, 基幹)	12,425	28
8	中規模 Storage	VM, Server, FC-SW(Level 1), 中規模 Storage	1,553	28
9	大規模 Storage	VM, Server, FC-SW(Level 1,2), 大規模 Storage	12,425	28

基幹の IP スイッチや，大規模ストレージに障害が発生すると影響が波及する装置数も膨大になる（表 1 の #7 および #9）．しかし，普通はそのような状況を見越し，それらの装置については図 5 のように多重化されている．この想定ではメールサーバや DNS サーバのような，全社から使われる基幹サーバを考慮していないが，それらの装置も同様に多重化されていると考えるのが適切である．そしてそれらに障害が発生した場合でもシステムの運用に影響はなく，障害原因解析を実施する必要がない．表 1 からこれらを除外すると，想定した構成では 1 つの障害によって影響が波及する装置数は，最大で 1,500 台程度である[*1]．

このように，大規模システムに障害が発生しても，発生した障害の影響は接続関係にある周辺装置のみに伝播する．前述のように既存技術では，発生しうるすべての障害イベントと原因との対応を，システム構成内のすべてのトポロジに対して事前に構築している．しかし，実際にはそれらの対応関係のうちの一部のみしか障害原因解析に利用されない[*2]．大規模システムを対象にすると，実際の解析に利用されない解析ルールが非常に多くなる．表 1 において，#7 および #9 を除いて発生した障害が最も多くの装置に影響する場合（#4，#6，#8）では，発生した障害が 1,553 個の装置に影響する可能性がある．いいかえると，1 つの装置に障害が発生した際に，多く見積もると 1,500 台程度に障害の影響が波及するということになる．図 4 に示した測定結果から推測すると，この規模では解析ルールの構築に約 1.90 時間（約 6,840 秒）を要する．

さらに障害発生時には，発生した障害イベントに関連する汎用ルールから構築された解析ルールのみを利用して障害原因解析が実施される．このため，障害イベントの発生した装置に対してすべての汎用ルールから解析ルールを構築しても，実際の障害原因解析にはそのうちの一部しか利用されない．表 1 を参照すると，#7 および #9 を除いて，1 つの装置に発生した障害が最も多くの装置に影響する場合（#4，#6，#8），障害原因解析を実施するために必要なルールは，汎用ルール 302 個のうちの 28 個である．解析ルールの構築に要する処理時間は汎用ルールの複雑さに依存する．それを無視して概算すると，前述の約 6,840 秒のうち，28 個の汎用ルールから解析ルールを構築するために要する時間は，約 630 秒と見積もることができる．障害原因を解析する際には，この解析ルール構築の処理時間のほかに，構築した解析ルールに発生した障害イベントをマッチングさせる処理時間を要する．この処理に要する時間は，同時に障害を検知した機器数に依存する．#4 や #6 や #8 の場合には 1 つの装置の障害が 1,553 台に影響して，それらすべての装置から障害を検知する．構築した解析ルールに含まれるすべての障害イベントと比較することになるため，解析ルール構築の約 630 秒の処理時間と同程度の処理時間を要すると予測できる．したがって，#4 や #6 や #8 の場合に障害に関係する部分のみを解析すると，障害発生から約 1,200〜1,300 秒程度の時間で障害解析を完了できる．

SLA ガイドライン [15] によると障害発生時の問題切り分けに要する時間は，上位レベルの運用管理現場でも障害発生から 30 分以内である．障害原因解析システムによって提示された原因候補を運用管理者が確認する時間を含めても，約 1,200〜1,300 秒であれば，障害を検知した段階で解析ルールの構築を実施しても，十分に基準を満たせる．

そこで，本研究では前述の問題を解決するために，障害発生時に障害原因解析を実施する範囲の解析ルールのみを構築し，それらを基に障害原因を解析する方式を提案する．

4.3 提案方式の構成と処理の流れ

本研究による提案方式による障害原因解析システムの構成と処理の流れを，**図 6** に示す．障害原因解析システムは，障害イベント取得モジュール，原因解析モジュール，ルール構築モジュール，ルール選択モジュール，トポロジ取得

*1 この想定では，均等にネットワークが分割されているが，実際にはそうとは限らず，台数には多少の差が生じる．しかし，実際の構築事例を調査した限り，10,000 台規模のシステムであれば，少なくとも基幹ネットワークが 10 個程度に分割されて運用されている．このため，多く見積もってもこの想定の範囲内である．

*2 たとえば，3.1 節で述べた例では，図 3 に示す 4 種類の解析ルールのうちの (4) を障害原因解析に利用していない．

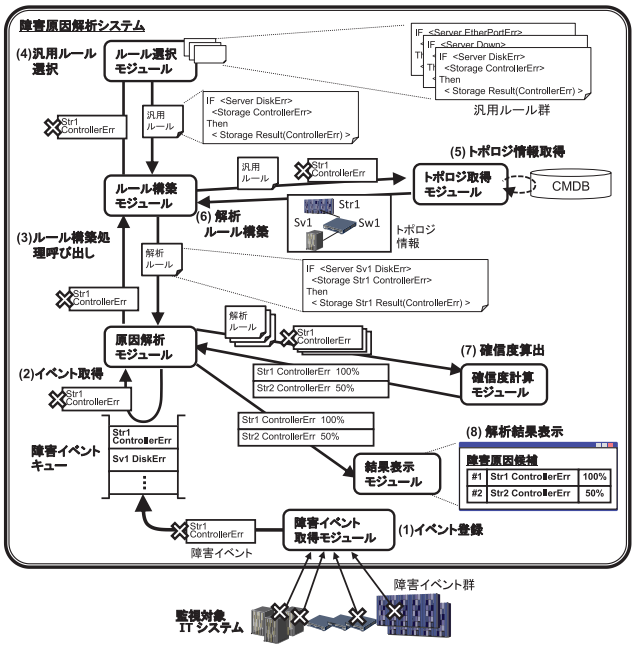

図 6 システム構成と処理の流れ
Fig. 6 System overview and execution flow.

モジュール，確信度計算モジュール，結果表示モジュールから構成される．なお，結果表示モジュールのみ Flash で実装しており，それ以外の部分は，Java 1.5 上で動作するアプリケーションとして実装した．CMDB は工藤らの手法 [13] と同様のものを用いた．CMDB は装置と装置内の構成要素の情報をテーブル化し，また装置間接続も別個のテーブルとして管理している．

以降に，図 6 に含まれるモジュールの概要を，システム全体の処理の流れとともに述べる．

(1) 障害イベント取得モジュールは，管理対象システムに含まれる各機器に対して，それぞれの装置のログファイルや，装置に特定の監視用コマンドにより状態を監視する．そして障害を検知すると，障害イベントとして，障害イベントキューに登録する．
(2) 原因解析モジュールは，障害イベントキューから障害イベントを取り出す．取り出した障害イベントに対して，手順 (3)〜(6) を繰り返す．
(3) 原因解析モジュールは，障害イベントをルール構築モジュールに渡し，解析ルール構築を指示する．
(4) ルール構築モジュールは，障害イベントを利用して，ルール選択モジュールを呼び出す．ルール選択モジュールは，汎用ルール群の条件部のイベントに，障害イベントを含む汎用ルールを選択し，応答として返す．
(5) ルール構築モジュールは，障害イベントと手順 (4) で抽出した汎用ルールを利用して，トポロジ取得モジュールを呼び出す．トポロジ取得モジュールは，CMDB から障害イベント発生機器の構成情報を取得し，汎用ルールの示す接続関係が存在するかチェックする．汎用ルールに別の機器への接続が定義してある場合は，該当する種類の機器の構成情報を CMDB から取得する．この一連の処理を，汎用ルールに記述されたトポロジすべてに対して実施する．汎用ルールに記述されたすべての機器への接続関係が存在している場合は，抽出できたすべてのトポロジ情報を応答として返す．
(6) ルール構築モジュールは，汎用ルールにトポロジ情報として受け取った機器の識別子を代入し，解析ルールを構築する．手順 (5) で複数のトポロジ情報が抽出された場合は，複数の解析ルールを構築する．そして解析ルールを，原因解析モジュールに応答として返す．
(7) 原因解析モジュールは，受け取ったすべての解析ルールと，手順 (2) によって障害イベントキューから取り出したすべての障害イベント群を使って，確信度計算モジュールを呼び出す．確信度計算モジュールは，それぞれの解析ルールに含まれる障害イベントのうち実際に発生した割合を，確信度として算出する．
(8) 原因解析モジュールは，受け取った障害原因を示す障害イベントと確信度の組合せを，解析結果表示モジュールを利用し，確信度の高い順に提示する．

以上の手順によって，障害原因解析システムは障害の原因を特定する．手順 (4) において，ルール選択モジュールは発生した障害イベントを含むルールのみを選択し，手順 (5) において障害イベントの発生した機器のみを対象に汎用ルールに合致するトポロジを取得し，手順 (6) で解析ルールを構築している．前述のように大規模システムになればなるほど障害の影響は局所的になるため，構築する対象となるルールやトポロジを絞り込み，必要最低限の解析ルールのみを構築している．したがって，図 3 のルール (4) のような，障害解析に不要なルールを構築しない．構築されなかった解析ルールは，既存手法 [13] において障害解析に利用していなかったルールであるため，障害原因解析精度には影響を与えない．

なお，大規模システムを対象にした場合，管理対象になる装置台数が多くなる．このため，上記の手順 (1) では，監視対象の各機器に対して追加で監視用のソフトウェアをインストールせずに，SSH や WMI（Windows Management Instrumentation）や，SNMP 等の一般的なプロトコルを利用して，障害イベント取得モジュールが外部から障害イベントの発生を定期的に監視するようにしている．

4.4 装置内構成要素に対する障害原因解析

図 5 では，単純化のためにストレージや，スイッチ，サーバをそれぞれ 1 つのノードとして表現している．しかし本研究で提案する障害原因解析システムは，前述のように装置内に存在するさまざまな物理的・論理的な構成要素（たとえば，ストレージ内であればコントローラや，ボリューム等）を対象に，障害原因解析を実施できるようにしている．

それらの構成要素間や，他の装置からそれらの構成要素に対する依存関係が複雑な場合には，1 つの構成要素に対する障害発生が，多くの他の装置に関連することがある．その場合は，構成要素とそれに影響を受ける装置の組合せが多くなり，解析ルールの構築時間が長くなる．

そこでそのような状況で解析時間を増加させないようにするために，解析ルールを装置内の構成要素間での障害発生パターンと，装置間での障害発生パターンに分割して記述できるようにした．障害発生時には分割した部分ごとに障害原因分析を実施し，装置内の構成要素間での障害原因分析結果と装置間での障害原因分析結果を組み合わせて，全体としての障害原因分析結果を導出する．

たとえばストレージでコントローラが故障した場合や，ボリュームに障害が発生した場合は，マウントしているサーバからは，どちらもマウントしているボリュームを提供するストレージに障害が発生していることに変わりはない．そのためストレージ装置を利用しているサーバとストレージ装置のノードとの関係を汎用ルールとしてルール化しておき，ストレージ装置のノードとそれぞれの構成要素との関係を汎用ルールとしてルール化することで，ストレージコントローラやボリュームとサーバ間の障害原因分析を実施できるようにしている．

装置内の構成が複雑であったとしても，装置間での汎用ルールと分割して記述すれば，装置間での汎用ルールに関連する解析時間には影響しない．装置内の構成要素についての汎用ルールに関連する解析時間が増加するが，装置内の構成要素の個数がそれほど多いとは考えにくい[*3]．このように，装置内の構成が複雑だった場合でも，解析時間に与える影響が大きくならないようにしている．

5. 評価実験

提案方式は，障害原因解析に必要な解析ルールを障害イベント受信時に構築しながら，障害原因を特定する．従来方式では解析ルールがあらかじめ構築され，それを利用して障害イベント受信時に障害原因解析を実施している．そのため提案方式は従来方式に比べ，障害イベント受信時に障害原因を特定するまでの時間が長くなるはずである．

そこで，障害発生時に提案方式を実用的に利用できるかどうかを判断するため，評価実験を行った．評価実験は，3.2 節で実施した事前実験と同一の，CPU が Intel Core2 Duo E6850 3.00 GHz で，メモリを 2 GB 搭載した PC で，Java 1.5.0 を利用して実施した．4.2 節で記述したように，提案方式ではある障害原因に起因して，約 1,500 台の装置に障害が波及する．そこで，CMDB 内にサーバと IP スイッチの接続関係を作成し，その接続関係をコピーすることにより，1 つの IP スイッチに発生した障害によって，多数のサーバに障害が発生するような接続関係を作成した．そして表 1 の #6 の状況を想定して，IP スイッチに発生した障害の影響波及装置を 1 台から 1,500 台程度まで変化させて，それらの IP スイッチとサーバに同時に障害イベントを発生させた．この障害イベントの発生は，各装置からの障害イベントを作成し，障害イベント取得モジュールに擬似的に送信することにより実施している．なお図 5 との構成の詳細の相違により，表 1 の #6 の状況で想定した汎用ルールのうち一部は障害原因解析に利用されず，20 個の汎用ルールを利用して障害原因解析を実施した．

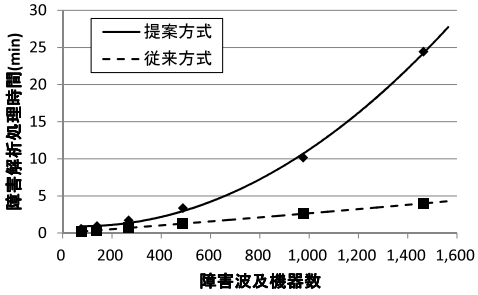

図 7 障害原因解析処理時間
Fig. 7 Evaluated time of root cause analysis.

障害イベント発生から，確信度計算モジュールによって発生したすべての障害イベントを利用して確信度が算出されるまでの時間を測定した（**図 7**）．なお，参考に従来方式で障害原因解析処理を実施した場合の時間も測定し記載している．測定の結果，1 つの IP スイッチに発生した障害が 1,552 台の他の装置に波及する際に，約 27.3 分で障害原因を解析して表示できることが分かった．

6. 考察

6.1 実験結果について

実験結果より，4.2 節に記述したような約 1,500 台に影響が波及するような装置に障害が発生した場合，その障害原因解析に 27 分程度の時間を要した．表 1 の #6 の状況では 28 個の汎用ルールを利用して解析を実施したが，前述のようにこの評価実験では 20 個のルールしか利用していなかった．28 個の汎用ルールを利用したと仮定すると，障害原因解析に，約 38 分（約 2,300 秒）の時間を要する．

4.2 節では，解析ルールの構築に約 1,200〜1,300 秒程度の時間を要すると予想した．障害原因解析のための解析ルール構築に要する時間は，元になる汎用ルールの示す接続関係の複雑さによって変化する．ここでは多段の IP スイッチによって構成される装置間のネットワークトポロジを示す汎用ルールを利用した．そのため，1 つの汎用ルールに示される接続関係の取得のために，多くの IP スイッチの情報を取得する必要があった．また，提案方式では

[*3] 構成要素の最小単位をどのレベルにするかは，障害発生時にどのレベルの構成要素に対して保守交換/設定変更を実施するかと，どのような障害イベントを取得するかに依存する．しかし一般に，1 つの装置に対して，保守交換可能で，かつ障害イベントを取得できる構成要素が膨大にあることは考えにくい．

CMDB から取得した情報をキャッシュしながら障害原因解析処理を実施する．実験結果を精査すると，700～800 台程度の装置に障害が波及した場合に，Java の VM が利用するヒープ領域の上限（1 GB）に達していた．このような場合にキャッシュをすべてクリアして，情報を再取得するように実装した．これらの原因により，障害原因解析に予想より多くの時間を要した．しかし SLA ガイドライン [15] にある障害発生時の問題切り分けに要する時間の上位レベル基準である 30 分の時間と同程度であり，中位レベル基準の 1 時間を十分に満たすことができた．

従来方式では障害原因解析のために，図 7 に示した解析処理時間を要する．この時間は障害イベントを解析ルールにマッチングする処理のみに要する時間であり，提案方式での障害原因解析処理時間より短い．しかし従来方式では，このほかに 3.2 節で考察した解析ルール構築処理の時間を事前に要する．この時間は障害の影響波及範囲に関係なく，運用管理対象システムの規模に依存する．そのため前述のように，10,000 台規模のシステムを運用管理対象にする場合に，17 時間近い長時間を要する．

提案方式では，解析ルール構築処理時間は障害イベント受信から原因を解析するまでに要する時間に含まれ，運用管理対象システムの規模ではなく，障害の影響波及範囲に依存する．そのため，局所的に発生した障害原因を解析する場合は，障害イベント受信から原因解析までに要する時間が短くなる．4.2 節に記述したように，通常は大規模システムであっても 1 つの装置の障害の波及範囲は限定的である．提案方式は，障害イベント受信時の解析処理時間が従来方式より増加するが，障害波及範囲のみを対象に解析ルールを構築するため，大規模環境で実用的に利用できる．

6.2 複数箇所で障害が発生した場合の動作

提案方式では，複数箇所で同時に障害が発生している場合には，それぞれの原因装置で発生した障害と，波及して障害が発生した装置についての障害イベントを検出する．そして，それらの障害イベントを適用できるすべての解析ルールを構築し，障害原因解析を実施する．その結果，複数の障害原因候補に対して 100%の確信度を提示する場合がある．その場合，100%と提示された複数の障害原因候補すべてが実際の障害原因であるため，運用管理者は提示された障害原因候補すべてに対応すればよい．

また，それぞれの箇所の障害に関連する解析ルール群が同じ装置を含む場合には，それらの障害が単一の原因によって発生したと仮定した場合の障害原因解析も実施される．このため 3.1 節で述べたように，それらの解析ルール群を合成して実施した障害原因解析結果も追加的に導出される．合成して実施した障害原因解析結果が実際の障害原因でない場合は不必要な情報が提示されることになり，運用管理者が原因に対処する際の効率を下げる可能性がある．障害原因対処作業の効率化のためには，複数箇所で障害が発生したと想定した場合と，それらの障害を単一の原因によって障害が発生したと想定した場合で，障害原因解析結果を分類して提示する必要がある．

6.3 適用可能なシステムについて

提案方式では，大規模システムで生じた障害原因を解析できるように，障害発生機器に関連する範囲についてのみ，障害事象と原因との関係をパターン化した上で原因解析を実施している．これは 4.2 節に記載したように，大規模システムに障害が発生しても，発生した障害の影響が接続関係にある周辺装置のみに伝搬するという仮定に基づいている．このため，特定の障害発生時に非常に多くの装置に障害が伝搬する場合に，解析時間が長くなる．

基幹の装置に障害が発生すると，影響が波及する装置数が膨大になる．4.2 節に記述したように，通常はそのような状況を見越してあらかじめ多重化を実施する．しかし多重化された装置間の切替えには，ある程度の時間を要する．この切替えが完了するまでは，接続されている各装置に障害の影響が波及する．4.2 節の想定では，多重化された基幹装置を障害原因解析範囲に含んでいないため，影響を受けたそれぞれの装置を原因とした障害原因解析結果が，一時的に提示される．

4.3 節に記載したように，提案方式では障害イベント取得モジュールが外部から障害イベントの発生を定期的に監視している．切替えが完了するまでの時間がこの監視サイクルよりも短い場合には障害の発生を検知しないため，障害原因解析が実施されない．提案方式の実装では，この監視サイクルを運用管理者によって設定できるようにしている．短いサイクルで頻繁に監視すると，管理対象装置の本来の業務を圧迫する可能性があり，また障害原因解析システムの負荷も高くなる．そのため，SLA ガイドライン [15] において障害通知時間の上位レベル基準が 10 分であることを考慮し，5 分以内の値を設定できないようにしている．したがって，切替えに要する時間が秒単位であれば，提案方式では障害の発生を検知しない可能性が高い．

一般的な多重化機構ではその目的から，切替えの際のダウンタイムをなるべく短くするように実装されている．たとえば，図 5 であげた IP スイッチのうち HA（High Availability）構成に対応したものであれば，ダウンタイムは数秒であり [16]，十分に短い．しかし切替えに時間のかかるアプリケーションサーバの HA 構成では，10 分程度の時間を要する場合がある [17]．このように，アプリケーションサーバのような上位レイヤの装置に障害が発生すると切替え処理に要する時間が長くなり，監視サイクルよりも切替え処理に要する時間が長くなる場合がある．その場合に，切替え対象の装置を利用する装置群（アプリケーションサーバであれば，クライアント装置群）が大量に存

在すれば，障害が波及して障害原因候補が大量に導出される可能性がある．これを防ぐためには，同時多発的に障害原因候補が導出された場合に，その候補群を除外して提示するような対策が必要である．

企業 IT システムにおいて，接続するクライアントが多いアプリケーションサーバとして，一般的には Web アプリケーション等の組織外に公開しているサーバがあげられる．クライアントの多くは組織外から接続をしており，通常はその多数のクライアントまでを含めて一元的に管理をしていない．すなわち，一般的にレイヤが上位になればなるほど管理範囲の局所性が失われる．このため，影響波及装置数が膨大になるようなアプリケーションサーバを対象に，影響波及先も含めて一元的に管理していることは，まずありえない．したがって，切替え時間の長いアプリケーションサーバの多重化を想定すれば切替え時のダウンタイムを無視できないが，提案方式での障害原因解析の対象にはならないと考えることができる．

6.4 制限事項

提案方式を利用するためには，次のような制限がある．

(1) 管理者の管理範囲について

一般的に大規模システムは複数の運用管理グループで分担して管理していると考えるのが妥当であるが，提案手法はそのような状況を想定していない．このため管理対象範囲内の装置群を分割できない．仮に障害原因解析システムを運用管理グループごとに用意し別個に運用管理を実施した場合，他の運用管理グループが管理する装置が原因で発生した障害を解析できない．

(2) 情報取得方式について

4.3 節に記載したように，提案方式では障害イベント取得モジュールが定期的に障害発生を監視している．このため，障害が発生しても最大で取得間隔だけの時間は，障害原因解析を開始できない場合がある．

7. まとめ

本研究では，大規模な IT システムを対象に，障害原因の解析を実施する手法を提案した．提案手法は，大規模システムにおいて，ある障害が発生した場合に，その障害により影響を受ける機器は一般的に限定的であるという状況に着目し，障害発生時に障害発生機器に関連するトポロジのみについて解析ルールを構築して障害原因解析を行う手法を提案した．そして評価実験を行うことにより，実際の大規模システムを対象にした運用現場で，実用的な解析処理時間内に障害原因解析を実施できることを確認した．

今後の課題としては，6.1 節で述べた Java VM のヒープメモリの上限の問題を回避するためのキャッシュ制御方法，6.2 節で述べた複数箇所で障害が発生した場合の障害原因分析結果の提示方法，6.4 節（1）で述べた状況に対応するための障害原因解析システム間での分散協調原因解析方法があげられる．

参考文献

[1] Yemini, S.A., Kliger, S., Mozes, E., et al.: High Speed and Robust Event Correlation, *IEEE Communications Magazine*, Vol.34, pp.82–90 (1996).

[2] Gupta, M. and Subramanian, M.: Preprocessor Algorithm for Network Managmeent Codebook, *Proc. Workshop on Intrusion Detection and Network Monitoring*, pp.93–102 (1999).

[3] Wang, M., Holub, V., Parsons, T., et al.: Scalable Run-Time Correlation Engine for Monitoring in a Cloud Computing Environment, *Proc. 17th IEEE International Conference and Workshops on Engineering Component-Based Systems* (*ECBS 2010*), pp.29–38 (2010).

[4] Holub, V., Parsons, T., O'Sullivan, P., et al.: Run-time correlation engine for system monitoring and testing, *Proc. 6th International Conference on Autonomic Computing* (*ICAC 2009*), pp.43–44 (2009).

[5] Rouillard, J.P.: Real-time Logfile Analysis Using the Simple Event Correlzator (SEC), *Proc. 18th USENIX System Administration Conference* (*LISA'04*), pp.133–149 (2004).

[6] 敷田幹文：大規模サーバ間の部品依存関係に基づく障害通知方式の提案，情報処理学会論文誌，Vol.49, No.3, pp.1185–1193 (2008).

[7] 敷田幹文，後藤宏志：大規模サーバ間の部品依存関係に基づくログ管理支援法，情報処理学会論文誌，Vol.49, No.3, pp.1081–1089 (2008).

[8] Xu, W., Huang, L., Fox, A., et al.: Detecting large-scale system problems by mining console logs, *Proc. ACM SIGOPS 22nd Symposium on Operating Systems Principles* (*SOSP'09*), pp.117–132 (2009).

[9] Xu, W., Huang, L., Fox, A., et al.: Mining console logs for large-scale system problem detection, *Proc. 3rd Conference on Tackling Computer Systems Problems with Machine Learning Techniques* (*SysML'08*) (2008).

[10] Mirgorodskiy, A.V., Maruyama, N. and Miller, B.P.: Problem Diagnosis in Large-Scale Computing Environments, *Proc. 2006 ACM/IEEE Conference on Supercomputing* (*SC'06*) (2006).

[11] Diao, Y., Jamjoom, H. and Loewenstern, D.: Rule-Based Problem Classification in IT Service Management, *IEEE International Conference on Cloud Computing*, pp.221–228 (2009).

[12] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D.A., Zhang, M.: Towards highly reliable enterprise network services via inference of multi-level dependencies, *Proc. 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (*SIGCOMM'07*), pp.13–24 (2007).

[13] 工藤　裕，森村知弘，菅内公徳，鷹田憲久：障害原因解析のためのルール記述方法とその実行方式，電気学会情報システム研究会，IS-09-71, pp.1–6 (2009).

[14] Dikaiakos, M.D., Katsaros, D., Pallis, G., et al.: Cloud Computing: Distributed Internet Computing for IT and Scientific Research, *IEEE Internet Computing*, Vol.13 Sep., pp.10–13 (2009).

[15] 社団法人電子情報技術産業協会：民間向け IT システムの SLA ガイドライン第三版，日経 BP 社 (2006).

[16] Cisco Systems, Inc.: High Availability for the Cisco Catalyst 6500 Series Switches, Whitepaper of Cisco Systems, Inc. (online), available from ⟨http://www.cisco.com/warp/public/cc/pd/si/casi/ca6000/tech/hafc6_wp.pdf⟩ (accessed accessed 2012-09-30).
[17] VMware, Inc.: VMware vSphere^{TM}4 Evaluator's Guide, VMware, Inc. (online), available from ⟨http://www.vmware.com/files/pdf/vsphere-evaluators-guide.pdf⟩ (accessed accessed 2012-09-30).

永井 崇之

2001 年東京大学工学部卒業．2003 年同大学大学院工学系研究科修士課程修了．2003 年より株式会社日立製作所．現在，同社横浜研究所研究員．システム運用管理に関する研究開発に従事．

名倉 正剛 （正会員）

1999 年慶應義塾大学理工学部卒業．2001 年同大学大学院理工学研究科修士課程修了．2006 年同大学院理工学研究科博士課程単位取得退学．博士（工学）．2001～2003 年日本電気株式会社．2006～2007 年慶應義塾大学大学院理工学研究科特別研究助手．2007～2009 年奈良先端科学技術大学院大学情報科学研究科特任助教．2009 年より株式会社日立製作所．現在，同社横浜研究所研究員．ソフトウェア工学，システム運用管理に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会各会員．