情報処理学会論文誌　デジタルプラクティス　Vol.2　No.3　11–26 (July 2021)

時系列障害原因分析による推論 QoS 規則導出手法

今野 賢[1,2,a),b)]　デファゴ クサヴィエ[3,c)]　冨田 尭[2,d)]　井口 寧[2,e)]

受付日 2020年8月17日, 再受付日 2020年11月4日,
採録日 2021年4月12日

概要：クラウド環境における QoS（Quality of Service）保証はサービス利用者と提供者の両者に共通する重要な課題である．特に障害発生時の運用作業においては調査時間も限られ運用者の介在による遅延も発生し，即時の障害復旧には運用業務の自動化が不可欠である．本稿は，近年顕著である時系列監視データベースのインメモリへの移行動向に適した，実時間で動作可能な事例ベース推論と時系列形状ベースの障害原因分析に基づいたイベント駆動型の推論 QoS 監視規則生成手法を提案する．本提案手法の適用により，障害発生時の即時の暫定的復旧対応に加え，再発防止に向けた障害原因特定および恒久的対処の自動化を目的とする．本提案手法は評価実験にて，従来の時系列クラスタリング手法と比較して同等の高適合率を保ちつつ，実時間で高速に動作する障害原因分析性能を示した．また現実的障害環境下での即時の自律復旧対策による有用性も示した．

キーワード：障害分析，自律的復旧，監視システム，時系列データベース，クラウドコンピューティング

Inference QoS Rule Derivation Based on Time Series Root Cause Analysis

Satoshi Konno[1,2,a),b)]　Xavier Défago[3,c)]　Tomita Takashi[2,d)]　Inoguchi Yasushi[2,e)]

Received: August 17, 2020, Revised: November 4, 2020,
Accepted: April 12, 2021

Abstract: Ensuring quality of service (QoS) is essential for cloud service providers and customers alike. In particular, automated recovery is essential to ensure long-term reliability and availability because human intervention is too slow, and not every situation can be anticipated. This paper proposes an event-driven inference QoS monitoring rule generation method based on case-based reasoning in real-time and shape-based time-series root cause analysis for recent in-memory time-series monitoring databases. This study aims to immediate provisional recovery response when a failure occurs, automation of root cause analysis, and permanent coping to prevent recurrence is realized. The results show that our approach maintains the same high precision as the conventional time-series clustering method, but at the same time, increases the root cause analysis performance that can operate in real-time. In addition, the results show a gradual characteristic and the usefulness of the immediate and permanent autonomous recovery measures in a realistic obstacle environment.

Keywords: root cause analysis, autonomous recovery, monitoring system, time series database, cloud computing

1　ヤフー株式会社
Yahoo Japan Corporation, Chiyoda, Tokyo 102–8282, Japan
2　北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology, Ishikawa 923–1211, Japan
3　東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
a)　skonno@yahoo-corp.jp
b)　skonno@jaist.ac.jp
c)　defago@c.titech.ac.jp
d)　tomita@jaist.ac.jp
e)　inoguchi@jaist.ac.jp

1. はじめに

クラウド環境 [1] における QoS（Quality of Service）保証は，サービス利用者と提供者の両者に共通する重要な課題である [2]．クラウド環境のサービスは複数のクラスターから構成され，近年の大規模なクラスターは数千から数万のノードから構成される [3], [4]．各々のクラスターからは間断なく膨大な監視メトリクスデータが出力され，年々爆発的に増加する監視データはもはや人間が手動で管

理および分析できる規模ではない [5]–[7]．また運用者が介在する QoS 保守作業には少なからず遅延が発生するため，即時復旧には可能な限り自動化し自律的なシステムへの移行が望ましい [8]．自律的な障害対応には，発生障害への根本原因分析または将来的な障害発生の予測が効果的な手法であり [9]，特に障害根本原因分析は同一障害の恒久的な再発防止に重要である [9]．従来の監視システムは，警報通知に重点が置かれており [10]，障害検知後の即時復旧は運用者の重要な課題である．ただし，近年の監視対象メトリクス数は膨大 [3], [8], [11] であり，障害発生時の即時復旧が優先される状況下で運用者による網羅的分析は困難である．クラウド環境の障害復旧には，暫定的な即時の復旧対応と合わせ，同一障害の再発防止として障害原因の特定および恒久的対策が求められる．結果として，障害発生時の運用者は限定された調査にとどまり，障害根本原因への恒久的な対策まで至らなければ同一障害の再発にもつながる．

クラウド環境における QoS は，クラウド利用者のサービス要件や提供者のシステム要件が指標化された多様なメトリクスから構成され [12]，その抽象化レベルや目的に応じた多様な監視や保証手法が提案されている [12], [13]．QoS は，単一の目標値または許容範囲値の QoS メトリクスを示す SLO（Service Level Objective）から構成され，SLO の監視および保証はクラウド利用者と提供者の両者にとり QoS 保証への重要な指標である [8], [12]．パブリック（Public）クラウドでは SLO を含む SLA（Service Level Agreement）に基づきサービスが提供されおり，指標は限定的ではあるが QoS が明確に定義されたパブリッククラウドサービスが提供されている [8], [12]．一方，プライベート（Private）クラウド環境は SLA が明確に定義されておらず，SLO はサービスごとに定義され一律ではない [8]．また，パブリッククラウド環境の監視目的および SLO 対象は，サービス提供者と利用者により視点が異なる [14], [15]．端的には，提供者はクラウドサービスの効率的な利用と安定性のみに，利用者はクラウドサービスの品質と性能のみに関心があり，双方の監視対象となる SLO には相違がある [10], [15]．プライベートクラウド環境においても，利用者はサービス品質の SLO のみに興味があり，それを保証する方法自身に興味は少ない．一方，提供者はシステムの SLO を保証する知識および経験は有しているが，利用者の SLO との関連性は不明瞭な状況が発生している．

しかしながら，クラウド環境の膨大な監視メトリクスデータと QoS および SLO の評価メトリクスの関係性は不明瞭である．また，近年のクラウド環境において監視メトリクスデータ収集に用いられる監視系時系列データベースは，膨大なメトリクスデータの高速な処理を目的に従来のストレージ型からインメモリ型への移行が顕著である [5], [6], [8]．本稿では近年のインメモリで処理される監視系時系列データベースを「監視系インメモリ時系列データベース」と定義する．本提案手法は，実時間で動作する高適合率な障害原因特定手法の提案および，障害発生時の即時復旧および恒久対策の自律化を目指した推論による新たな QoS 監視規則導出手法である．すでに我々は近年の監視系インメモリ時系列データベースを活用した障害原因特定手法を試み，現実的障害環境下での自律復旧対策による有用性を得ている [16]．本稿では，単一サービスのみを対象とした文献 [16] から新たなサービスについての実験も加え，提案手法が幅広く適用可能であることを確認する．

本稿では，まず 2 章で QoS 保証問題および関連する関連研究について述べる．次に，3 章で本提案手法である QoS 監視によるイベント駆動型の障害原因分析に基づく自律的 QoS 保証手法について述べ，4 章で本提案手法の動作に必要となる分散型監視システム要件について説明する．5 章で本提案手法を人工的および現実的障害環境下で評価し，6 章に考察として本提案手法の有用性および課題を示す．

2. 関連研究

近年のクラウド環境では監視系インメモリ時系列データベースへの移行動向が顕著である [5], [6], [8]．本章では，クラウド監視環境の動向および障害原因分析の関連研究について述べる．

2.1 クラウド運用と自己復旧

近年のクラウド環境では SRE（Site Reliability Engineering）による運用視点が重要視されている [8]．SRE とはサービス目標の SLO 担保を目的に，障害を回避し運用業務の効率化と自動化を目的とする指針や活動を指す [8]．SRE における自動化はシステム運用において可用性の向上および運用者時間的な節約にもつながり，自動化の最終段階として運用者の介入なしに復旧するシステム構築が目標となる [8]．自律コンピューティング（Autonomic Computing）においてシステムの自己管理能力は，自己設定（Self-configuration），自己最適化（Self-optimization），自己復旧（Self-healing），自己防衛（Self-protection）の 4 種類に分類され [17]，SRE における自動復旧は自己復旧（Self-healing）に分類される．

2.2 監視系時系列データベースのインメモリ化

クラウド環境の QoS 保証には対象サービスおよび提供システムの監視が重要である [2]．クラウド環境は，従来から監視メトリクス収集およびグラフ表示を目的とする時系列データベースを持つ各種の監視システムが存在してきた [18]．しかし従来の監視システムはノード単体での稼働かつストレージ利用を前提とする時系列データベースとし

て設計されたものが多く，近年のクラウド環境のような膨大なメトリクスの収集や収集した膨大なメトリクスデータを迅速に読み取るのには適していない [5], [6], [8]．ゆえに近年のクラウド環境の監視システムでは膨大なメトリクスデータへの高速な読み取りを可能にすべく，DataGarage [19]，Borgmon [8]，Gorilla [6]，Atlas [5] など旧来のストレージ型の時系列データベースベースからインメモリ型への移行が顕著である．

2.3　離散監視データの分析手法

クラウド環境の QoS 保証において，障害分析は可用性の担保に効果的な手法であり [9]，離散的なイベント分析と連続的なイベント分析手法に大別できる．障害分析は障害の根本原因を特定し同一障害が再発防止する手法であり，クラウド環境のサービスおよびシステムから出力されるイベントログは非連続で離散的なものが多い．離散的なイベント間の分析については，相関分析による障害原因特定手法が多く提案されており，離散的なイベントを形式化し共起関係やクラスタリングによる分析による障害原因の特定手法 [20]–[24] や，Rouillard らの時系列分析による相関関係のない離散イベント除去の提案 [24] 等がある．ただし，いずれの障害原因特定手法もイベントログに代表される離散的イベントを対象としており，近年のクラウド環境で導入が進んでいる時系列監視データベースが対象とする連続的な時系列イベント分析への適用は考慮されていない．

2.4　時系列監視データの分析手法

従来の監視システムが対象とする離散的なイベントに対し [20]–[24]，近年のクラウド環境の監視システムでは連続的イベントであるインメモリ時系列監視データベースへの移行が顕著である [5], [6], [8]．一般的な時系列データ分析としてはピアソンの積率相関係数（PPMCC）[25] などの相関アルゴリズムの適用があり，近年のクラウド監視システムにおいても相関アルゴリズムとして PPMCC の適用事例がある [6], [19], [26]．ただし，各クラウド監視システムのインメモリ時系列データベースへの移行は必ずしも障害分析が主目的ではなく，集約や集計など目的は異なる [6], [8], [19]．さらに，監視システムに発生しうる課題，例として監視取集データ欠損時の対処や異なる時系列周期間の考慮などの現実的な問題への言及には乏しい [6], [19], [26]．クラウド処理分野では，BigRoot [27] の時系列監視データを用いたクラウド処理性能劣化の要因特定手法の提案については，本稿が対象とする異常系の原因特定や異なる時系列周期間の分析は考慮されていない．また，Automap [28] や Juan [29] らの複数ノードで構成されるクラウドサービスの因果関係および障害ノード特定手法の提案については，CPU やストレージなどの限定された数少ない同種のメトリクスを対象とした因果関係の分析手法であり，本稿が対象とする多種多様なメトリクスの因果関係分析やノード内で発生する障害根本原因そのものの特定については考慮されていない．

時系列データへのクラスタリング分析は時系列パターン発見の一般的な手法であり [30]，他の分析手法の前処理および内部処理にも用いられる最も一般的なデータマイニング手法である [31]．しかし，本稿は多種多様な新しいアプリケーションの膨大なデータセット分析を対象としており，教師ありの分類手法に必要なドメイン知識に乏しい場合も想定され，教師なしの分類手法であるクラスタリングの適用が現実的である [30]．時系列データクラスタリングは時系列データパターン発見を目的とし 2 種類に大別される．最初の目的は頻出データセットパターンの発見で，k-Shape [32] による時系列データに適したクラスタリングアルゴリズムの提案がある．もう一つの目的は，データセットの問題を発見するものであり，特に異常検出はデータセットで発生した異常または予期していないパターンを検出する手法である [30], [33]．ただし，本稿は障害原因分析が対象のため詳細は取り扱わない．

2.5　クラウド環境 QoS の管理・保証

クラウド環境の QoS 管理および監視を対象とする関連研究はパブリッククラウドの利用者または提供者のサービス要件管理を主題とし，その多くは要件のモデリング手法や記述方式自体を研究対象としている [34]．現実的な QoS 管理の関連研究では，MonSLAR [35] が利用者視点でクラウド環境の SLA 監視手法，MonPaaS [36] が利用者および提供者視点双方の SLA 監視手法を提案しているが，いずれも監視手法の提案にとどまり QoS 保証まで考慮された研究ではない [15], [34]．また，パブリッククラウド環境のトラフィック増減への対応やコスト最適化を目的とする関連研究 [13], [34], [37] ではクラスタノードの動的な増減手法をとる．ただし，これらの関連研究は正常環境下での最適化を目的とし，本稿が想定する異常系での QoS 保証や障害原因特定は考慮されていない．さらに，パブリッククラウド環境を対象とした関連研究は，SaaS，IaaS や PaaS [1] などのパブリッククラウド特有の提供サービスごとに細分化され [13], [37]，プライベートクラウド環境下における固有階層や複数階層の連携については考慮されていない．また，監視対象者についても，パブリッククラウド環境を対象とした関連研究では提供者およびその利用者の 2 者のみに簡略化 [12], [34]–[36] される傾向があり，プライベートクラウド環境下における提供者と利用者の境界が曖昧な環境下は考慮されていない．

2.6　QoS の知識表現と推論

近年の大規模化や運用管理の分散化の背景から，クラウ

ド環境におけるシステム全体の把握は管理者個人の限界を超えており，システムの安定運用には個人の運用経験によらない運用知識の知識ベース（Knowledge Base）の構築が必要とされる [38]．知識ベースは，人工知能（Artificial Intelligence）分野の主要領域であり [39], [40]，知識ベースは知識表現言語 [41] と呼ばれる一連の文で構成される言語にて表現される．本章は，AI における基本的な知識推表現と推論手法，ならびに関連研究について述べる．

規則ベースシステム [42] は「*if condition,then action*」という形式の規則が含まれる知識表現で，従来の監視システムにおいても標準的な監視条件の定義に適用されており [9]，近年のクラウド環境の監視システムでは独自のドメイン固有言語を適用し，より複雑な規則の定義を可能としている [8]．CCT [22] では相関表を用いて障害規則と障害原因を多対多で関連づける手法，工藤ら [43] は汎用ルールからシステム構成に応じた規則を生成する手法を示したが，いずれも既知の障害知識に基づいた静的なものであり，利用者の QoS 保証や未知の障害に対しての障害復旧については考慮されていない．また，規則ベースシステムの管理は，クラウド環境のような複雑なシステムまたは更新頻度の高いシステムにおいて網羅的な監視規則を設定および維持は困難であり [22], [43]，運用開始時から利用者の SLO を担保する明示的な監視規則の設定は難しい．

規則ベースシステムの監視規則の複雑な設定および管理コストを緩和する方法として事例ベース推論の適用がある [40]．事例ベース推論 [44] は，過去の経験 [45] に基づいて問題を解決するプロセスであり，次の 4 段階の推論プロセスとして形式化される．

(1) 事例検索：過去事例から現在の問題に類似した事例を検索.
(2) 事例修正：過去事例を現在の問題に適用し，必要に応じ適合するよう過去事例を修正.
(3) 事例修復：問題解決策を評価し，失敗する場合は問題解決策を修復
(4) 事例格納：現在の問題解決策を事例ベースに新規事例として保存.

従来の規則ベースのクラウド監視システム [6], [19], [26] は監視条件は静的であり，事例ベース推論による規則修正は考慮されていない．NetPilot [46] は，事例検索しその影響評価を推定する手法を示したが事例修復の評価は運用者に委ねている．FOSII [47] は事例修正として観測された SLA 違反の QoS 規則修正手法を示したが，相関分析による QoS 監視閾値の修正のみにとどまっている．

モデルベース推論 [48] は，数学モデルまたは計算モデルを用いており，障害予測には多種のモデルが提案されている [10]．近年の障害予測と異常検出の関連研究 [5], [9], [33] の多くも数学モデルに基づいているが，本稿は過去の障害原因分析が対象のため予測的な研究についての詳細は取り扱わない．

3. *μQoS*：提案手法

本稿は，近年のクラウド環境の監視系インメモリ時系列データベースの膨大な時系列監視メトリクス分析を前提とする，*μQoS* と呼ばれるイベント駆動型の障害原因分析に基づく自律的 QoS 保証手法を提案する．クラウド環境では，利用者が目標とする多種多様な SLO と運用者がシステム的に担保している SLO との関連性は運用当初は不明瞭な場合が多く，規則ベースの監視システムにおいて明示的な QoS 監視規則を設定するのは困難である．本提案手法は，利用者と運用者の SLO の因果関係が不明瞭な環境下においても，関連研究と比較し実時間で高速に動作し高適合率を示す時系列ベースの障害原因分析手法に基づく，新しい推論監視規則の生成による利用者の SLO を担保，即時の障害復旧および恒久的な障害対策の自動化を特徴とする．クラウド環境の QoS 保証の運用業務には，障害発生時に暫定的な即時復旧対処と合わせ，障害原因を特定した恒久的な対策が求められる．本章では提案手法の概要を述べる．

μQoS は，発生した QoS 障害の根本原因分析結果に基づく QoS 保証手法である．従来の監視システムは警告までを目的とするが，*μQoS* は QoS 障害検知から障害復旧まで対象としており，**図 1** に示す以下の 4 段階の処理順序により自動化を実現する．

(1) 監視メトリクス収集：対象システムおよびサービスの全メトリクスを監視対象として常時収集.
(2) QoS 条件監視：収集したメトリクスが QoS 監視条件を満たしているか判定．不成立であれば次のステップに移行.
(3) QoS 障害原因分析：不成立な QoS 監視条件式から障害原因候補となる監視メトリクスを特定.
(4) QoS 規則事例ベース推論：特定された障害原因候補と既存 QoS 監視規則から新規推論 QoS 監視規則を生成.

QoS 保証の関連研究 [13], [15], [34], [37] が監視規則を静的に取り扱うのに対し，*μQoS* は QoS 保証のため，新たな QoS 監視規則自身を動的に生成し補完する．*μQoS* は，運用者の介在を必要としない自律的な復旧を目的としており，その障害原因候補には実時間かつ高適合率な特定手法が求められる．*μQoS* は，特定された障害原因候補および既存 QoS 監視規則から，自律的な QoS 保証のため自動推論による新たな推論 QoS 監視規則を生成する．

3.1 監視メトリクス収集

μQoS は近年のクラウド監視システムの移行動向 [5], [6], [8], [19] を踏まえ，監視メトリクスを大規模かつ迅速に分析可能なインメモリの時系列監視データベースを前提とす

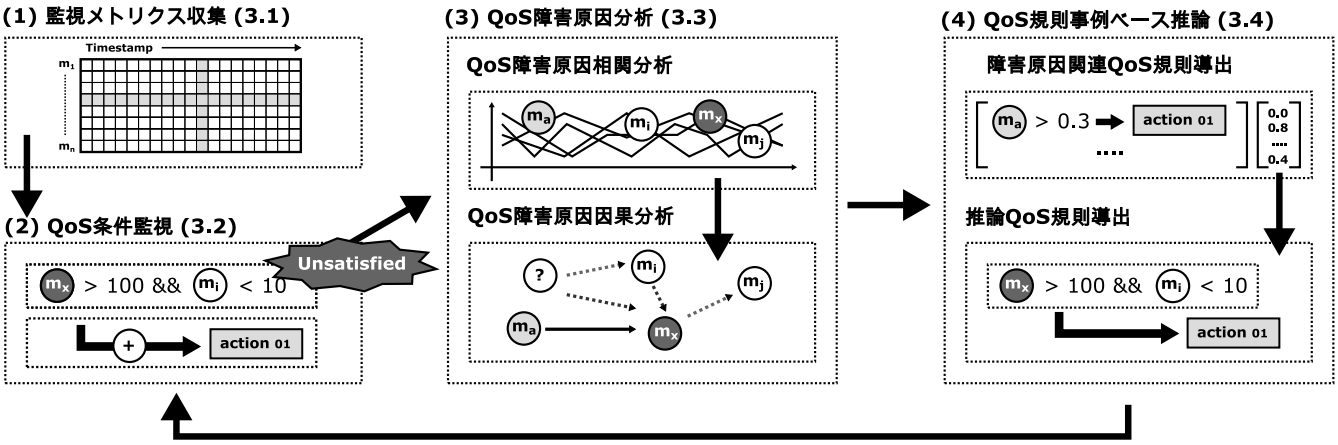

図 1　時系列障害分析による推論 QoS 規則生成処理手順

Fig. 1　Overall autonomous recovery framework based on root cause analysis.

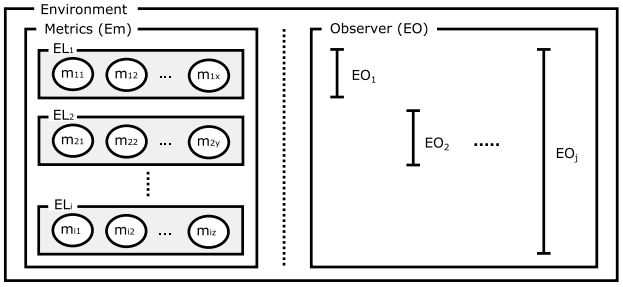

図 2　監視メトリクス環境の抽象化

Fig. 2　Abstraction of monitoring layers and observers.

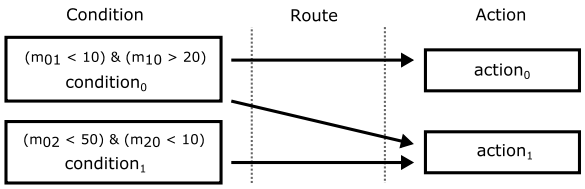

図 3　QoS 監視規則の分割知識表現

Fig. 3　Subdivided knowledge representation for QoS rule.

る手法である．本稿におけるクラウド環境の監視メトリクス環境および提供者と利用者の関係性を**図 2** に示す．

図 2 において本稿の監視対象となる監視メトリクス環境（Em）は，有限の i 個の環境層 $\{EL_1,...,EL_i\}$ から構成され，各環境層（EL_x）は有限の監視メトリクスの集合 $\overrightarrow{m_x} = \{m_{x1}, ..., m_{xn}\}$ として定義する．たとえば，図 2 の最上位層の EL_1 はサービスが稼働しているアプリケーション層，最下位層の EL_i はクラウド環境で監視可能なネットワーク層を表している．本稿で対象となるプライベートクラウド環境下では監視メトリクス層の境界はプライベートクラウドごとの状況により異なるため，パブリッククラウドの提供サービス分類視点の SaaS，IaaS や PaaS [1] のような具体化した監視メトリクス層の定義はしない．また，監視メトリクス層の観測者（Observer）についてもパブリッククラウド環境の提供者および利用者のような明確な境界はプライベートクラウド環境ごとの状況により同じく異なるため，本稿は観察者についても具体的な定義はしない．従来のパブリッククラウドでの階層ごとの利用者や提供者については図 2 に示す全体の監視環境の部分集合が観測できる観測者である EO として抽象化して表現される．たとえば，図 2 の対象者 EO_1 はサービスの SLO 指標を含む EL_1 層のみに興味がある現実的な利用者，対象者 EO_j はサービスおよびシステム稼働への全責任がありすべての監視メトリクス層への管理権限のある理想的な提供者を表している．

3.2　QoS 条件監視

本提案手法は，新規の QoS 監視規則を既定の QoS 監視規則と障害原因分析結果から導出するため，監視条件とアクションを明確に分離して管理する．従来の監視システム [3], [8] は監視条件とアクションを一連の規則として定義してきたが，本提案手法は，監視規則を規則ベースシステム [42] に基づき，新たな監視規則の導出を目的に QoS 監視規則の条件（Condition）式とアクション（Action）を**図 3** に示す分割した監視知識として管理する．

図 3 は QoS 監視条件式（$conditon_{0,1}$）が不成立（Unsatisfied）時に実行される複数の QoS 監視規則アクション（$action_{0,1}$）から構成される QoS 監視規則の定義例を示す．まず，QoS 監視条件式は監視メトリクス環境（Em）の収集メトリクスを変数とする命題論理式として定義される．図 3 に示す QoS 監視条件の $conditon_0$ は収集メトリクス m_{01} が 10 以下かつ m_{02} が 20 以上とする二つの節（Clause）の論理積（&）を成立（Satisfied）条件とする．また，QoS 監視条件式とアクションには一意の名称があり，これら名称により QoS 監視条件式とアクションは接続（Route）される．QoS 監視条件式とアクションは，必ずしも 1 対 1 の関係ではなく，条件によって 1 対多やその逆もあり得る．図 3 は，一つの QoS 監視条件式（$conditon_0$）に二つの QoS 監視規則（$action_{0,1}$）が接続されてお

り，QoS 監視不成立時に複数の復旧アクションが必要な状況を示す．なお，QoS 監視規則アクションは監視条件式に定義された収集メトリクス変数や障害分析手法変数など引数とする静的または動的プログラミング言語関数として運用者が定義する．

3.3　QoS 障害原因分析

μQoS は，監視条件式の不成立を契機とするイベント駆動型の障害原因分析手法である．本提案手法は，従来のログ監視を対象とする連続分析的な関連研究 [11], [23] と比較し，監視 QoS 条件式を契機とする非連続なかつ動的な障害原因分析手法である．*μQoS* は，対象とするクラウド環境のシステムやサービスに合わせた最適な障害原因分析手法の選択または複数手法の組み合わせを想定している．各々の障害原因分析処理は，不成立 QoS 監視条件式に含まれる不成立メトリクス（m_{unsat}）との関連性を，式 (1) に示す相関係数ベクトルを返却するよう正規化する．

$$\mu_c(\vec{Em}_c) = \mu_c(\{mc_1, mc_2, \cdots, mc_n\}) \tag{1}$$

ファジィ推論 [49] は曖昧さを許容した近似的な推論のための論理である [40]．関数 μ_c はファジィ推論を利用したファジィ制御 [50] のメンバーシップ関数であり障害原因分析処理ごとに定義され，mc_x は m_{unsat} と対象メトリクス m_x との相関係数を示す．ファジィ制御は単純な真偽判定だけではなく中間の状態を許容しており [40]，本提案手法では，ファジィ制御を線形および非線形の相関係数ベクトルを返却する障害原因分析結果の最終的な判定に用いる．各障害原因分析処理は式 (1) の正規化された相関係数ベクトル（$\vec{Em}_c$）を返却するが，その正規化手法や相関係数の範囲および因果関係判定基準は各障害原因分析処理ごとに異なる．すなわち関数 μ_c は各相関係数から，各障害原因分析処理ごとの最終的な因果関係の有無や強さを判定する関数として定義される．関数 μ_c の適用例としては，相関係数をある閾値や信頼区間にて因果関係の有無を判定したり，因果関係の有無を学習結果から動的な判定するような用途を想定している．*μQoS* の障害原因分析処理は，関数 μ_c により相関関係の有無や強さが判定され，不成立監視 QoS 条件式の変数 m_{unsat} と時系列監視データベースが保持する全監視メトリクス Em との最終的な因果関係を示す相関係数ベクトルを$\mu_c(\vec{Em}_c)$として返却される．

3.3.1　QoS 障害原因分析手法の高速化と高適合率化

本稿における障害原因分析には，再発防止を含む自動的な恒久対策に必要な高適合率な障害根本原因の特定手法，さらに即時の自動復旧を実現する実時間で動作可能な手法が求められる．また，本研究は運用者を非介在とする自動復旧および恒久的対策の自動化を最終的な目的としており，誤った障害原因候補による自動復旧は QoS への影響ならびに更なる障害の誘因にもつながる可能性もあり，障害原因分析の高適合率化は重要である．また，障害原因分析の関連研究は，一般的な相関関係 [5], [6], [8], [19] や類似度 [32] または特異性 [27] の分析を目的とし，本稿が対象とする障害原因とする因果関係のある障害分析や異なる時系列周期間の分析については考慮されていない．本稿では，*μQoS* のインメモリの時系列監視データに適した障害原因分析手法として，実時間での動作かつ高適合な障害根本原因の特定を実現する時系列形状ベースの障害原因分析手法（ShapeRoot）を合わせて提案する．

Algorithm 1 時系列形状相関距離算出アルゴリズム

Require: $\overrightarrow{m_u} = \{m_{u1}, ..., m_{un}\}$
Require: $\overrightarrow{m_x} = \{m_{x1}, ..., m_{xm}\}$
1: Let *cost* be a matrix containing the cost values
2: Let *path* be a matrix containing the warping paths
3: Let δ be a distance between coordinates of metrics
4: **for** $i = 1$ **to** n **do**
5:　**for** $j = 1$ **to** m **do**
6:　　$cost[i][j] = \delta(m_{ui}, m_{xj})$
7:　**end for**
8: **end for**
9: $path[1][1] = cost[1][1]$
10: **for** $i = 2$ **to** n **do**
11:　$path[i][1] = path[i-1][1] + cost[i][1]$
12: **end for**
13: **for** $j = 2$ **to** m **do**
14:　$path[1][j] = path[1][j-1] + cost[1][j]$
15: **end for**
16: **for** $i = 2$ **to** n **do**
17:　**for** $j = 2$ **to** m **do**
18:　　$path[i][j] = min(path[i-1][j],$
19:　　　　$path[i][j-1],$
20:　　　　$path[i-1][j-1]) + cost[i][j]$
21:　**end for**
22: **end for**
23: **return** $path[n][m]$

ShapeRoot は，時系列データの形状類似性評価手法である DTW（Dynamic Time Warping）[51] に基づく時系列データ間の最短経路距離情報を障害相関判定に用いる．ShapeRoot は異なる時系列周期データ間の相関判定も可能である．**Algorithm 1** に ShapeRoot の疑似コードを示す．ShapeRoot は，QoS 監視条件式の時系列メトリクス（$\overrightarrow{m_u}$）と他のすべての時系列メトリクス（$\overrightarrow{m_x}$）の 2 系列データ間の最短経路距離を求める．ShapeRoot は，最初に距離計算式 $\delta(m_{ui}, m_{xj})$ により時系列全 2 点データ間距離をコスト（cost）行列に格納する（行 4–8）．ShapeRoot は距離計算式にユークリッド距離を用いる．次に経路（path）行列の最初の行および列を計算済みコスト式の積算により初期化し（行 10–13），2 行および列からは各行列成分（$path[i][j]$）を計算済み 3 周辺の最小経路成分とコスト式の加算により漸進的に求める（行 14–20）．最後の行列成分（行 21）が本手法による時系列メトリクスデータ間の最短経路距離となり，最終的な ShapeRoot の障害原因分析の相関係数となる．

3.3.2　QoS 障害原因因果分析による高適合率化

近年のクラウド監視系時系列データベースの関連研究 [6], [19], [26] で用いられる相関アルゴリズムや，時系列データへのクラスタリング分析 [30] では相関関係のみで因果関係が考慮されていない [52]．ShapeRoot においても Algorithm 1 のみ評価された障害原因候補については形状的な相関関係のみで因果関係が考慮されていない．クラウド環境で収集される監視メトリクスには，CPU 負荷や応答遅延など，同じ因果要因を上流に持ち交絡要因となる異なる周期の平均値やパーセンタイル値など同一種の統計的指標データが含まれる．たとえば，5 章の実験に用いた Apache Cassandra [53] には，読み込み遅延を示すメトリクスとして，各テーブルごとに平均値や最大値および最小値，さらには複数のパーセンタイル（50,75,95,98,99.9）の同一種の指標データが含まれている．そのため *μQoS* では，擬似相関を示す交絡要因については候補メトリクス名称の Levenshtein 距離係数 [54] から，因果関係の逆転については発生時刻の先行性から，後処理として ShapeRoot の障害原因候補から除外している．このように本手法が目的とする恒久的対策および将来の予防的対策の自動化には因果関係の厳密性が求められるが，更なる時系列を考慮した因果関係の分析手法については今後の課題とし本稿では詳細は取り扱わない．

3.4　QoS 事例ベース推論

本提案手法は，障害根本分析から推論された新規の QoS 復旧監視規則の自律的かつ動的な適用により，即時の障害復旧および恒久的な障害対策を目的とする QoS 保証手法である．*μQoS* は事例ベース推論 [44] とファジィ論理 [49] に基づき，自律的復旧のために新たな推論 QoS 監視規則自体を生成し補完する．

3.4.1　高適合な障害原因候補規則の導出

μQoS は，障害原因候補規則を 3.3 節の式 (1) の障害原因相関係数ベクトル（$\mu_c(E\vec{m}_c)$）をその導出に用いる．*μQoS* は，既定の全 QoS 監視規則（$RuleSet_{all}$）を対象に $\mu_c(E\vec{m}_c)$ の相関係数ベクトルから因果関係のある QoS 監視規則群（$RuleSet_{root}$）を式 (2) により導出する．

$$\begin{aligned} RuleSet_{root} &= RuleSet_{all} \times \mu_c(E\vec{m}_c) \\ &= \{Rule_{root_1}, \ldots, Rule_{root_n}\} \end{aligned} \tag{2}$$

本処理は，事例ベース推論の取得ステップに対応し，不成立となった QoS 監視条件規則と因果関係のある QoS 監視規則のみを最終推論候補として導出する．各 QoS 監視条件は，3.2 節に示すように監視環境（Em）のメトリクスを変数として定義され，各 QoS 監視条件（$Condition_x$）に含まれるメトリクス変数（m_x）が障害原因分析相関関数（$\mu_c(mc_x)$）的に閾値以下で因果関係がない場合，その監視規則は推論 QoS 監視規則候補から除外される．最終的に，*μQoS* は既定すべての QoS 監視規則（$RuleSet_{all}$）から障害原因として因果関係のある QoS 監視規則のみを障害原因関連 QoS 規則群（$RuleSet_{root}$）として導出する．

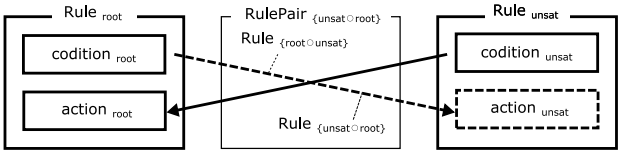

図 4　推定 QoS 監視規則対
Fig. 4　Approximate QoS monitoring rule pair.

3.4.2　推論規則導出による障害復旧の自動化

最終的に *μQoS* は，不成立となった QoS 監視条件規則（$Rule_{unsat}$）と，障害原因と関連のある QoS 監視規則候補群（$RuleSet_{root}$）に属する QoS 監視規則候補（$Rule_{root}$）との合成演算（∘）により，新しい推定 QoS 監視規則群を導出する．推定 QoS 監視規則群は推定 QoS 監視規則対（$RulePair_{\{unsat \circ root\}}$）から構成され，推定 QoS 監視対は**図 4** に示す，即時の暫定的復旧対応を目的とする本来過去に実行すべきであった推定 QoS 復旧監視規則（$Rule_{\{unsat \circ root\}}$）と，将来的な再発防止に向けた恒久的対処を目的とする推定 QoS 予防監視規則（$Rule_{\{root \circ unsat\}}$）から構成される．

本導出は事例ベース推論の再利用，改訂，および保持のステップに対応し，推定 QoS 監視規則対である推定 QoS 復旧監視規則および推定 QoS 予防監視規則は，不成立となった QoS 監視条件規則と QoS 監視規則候補から新しい 2 種類の監視規則が式 (3) から導出される．

$$\begin{aligned} &RulePair_{\{unsat \circ root\}} \\ &= Rule_{unsat} \circ Rule_{root} \\ &= \{Rule_{\{unsat \circ root\}}, Rule_{\{root \circ unsat\}}\} \\ &= \{\mu_c(condition_{unsat}, action_{root}), \\ &\quad \mu_c(condition_{root}, action_{unsat})\} \\ &= \{(\boldsymbol{if\ not}\ condition_{unsat}\ \boldsymbol{then}\ action_{root}(\mu_c)), \\ &\quad (\boldsymbol{if\ not}\ condition_{root}\ \boldsymbol{then}\ action_{unsat}(\mu_c))\} \end{aligned} \tag{3}$$

$Rule_{\{unsat \circ root\}}$ は，不成立となった QoS 監視条件規則のアクション（$action_{unsat}$）に加え，因果関係がある QoS 監視規則のアクション（$action_{root}$）を実行することにより即時復旧すべく導出される．一方，$Rule_{\{root \circ unsat\}}$は，不成立となった QoS 監視規則と因果関係がある QoS 監視規則条件式（$condition_{root}$）の不成立時に，予防的に不成立となった QoS 監視規則アクション（$action_{unsat}$）を実行すべく導出される．なお，障害原因分析手法のメンバーシップ関数（μ_c）は，新しく生成された推論 QoS 監視規則のアクション引数に指定され，アクション関数内でその相関係数からその実行を制御可能とする．

1 章で述べた背景を図 4 に置き換え，提供者がシステムの SLO を保証する監視規則（$Rule_{root}$）郡を適切に設定し安定的に運用している状況下で，新しいサービスが稼働す

る状況を説明する．利用者はサービス品質の SLO のみに興味があり担保する手法は不明なため，監視規則アクション（$action_{unsat}$）がない監視条件式（$condition_{unsat}$）のみの監視規則（$Rule_{unsat}$）を設定する．結果として利用者の SLO と運用者の監視規則との関連性が不明瞭なままサービス運用が開始されるが，利用者の QoS 監視規則条件が不成立となった時点で，本手法により即時復旧を目的とする新しい推論 QoS 監視規則（$Rule_{\{unsat \circ root\}}$）が生成され適切な既存の有効な監視規則の復旧アクション（$action_{root}$）が実行される．ただし，この事例の場合には利用者の監視規則アクションは未設定なため予防的な推論 QoS 監視規則（$Rule_{\{root \circ unsat\}}$）は生成されないが，現実的には利用者が監視規則アクションまで設定できる事例は少なく，本手法の想定の範囲内の運用となる．

4. 監視システム設計と実装

μQoS は，監視系インメモリ時系列データベースの全メトリクスを対象に障害原因を分析する手法であり，収集された監視メトリクスへの高速なアクセスが必要とされる．*μQoS* は，独自の分散監視システム（Foreman）での実行を想定された自律的復旧方法であり，監視系インメモリ時系列データベースへの高速なアクセスが可能である．Foreman は，クラウド環境の監視，警報，運用知識管理，自律的復旧含む総合的な運用基盤としてヤフー株式会社のデータベース運用部門で試験的に開発されている．本章では，*μQoS* を動作前提となる監視システムとしての Foreman の特徴および設計の概略を述べる．

4.1 監視システムの概要

μQoS は自律的な復旧処理を目的とし，監視システムの障害検知から対象システムへ復旧作業までの自動化が必要とされる．そのため，Foreman は**図 5** に示す対称分散型の監視システムとして，復旧対象サービスプロセスと同じインスタンスでの稼働を前提としている．

図 4 に示すように，Foreman は同一の監視対象サービス（Service）のインスタンスでクラスタ（Cluster）を構成し，他 Foreman インスタンス（External）の監視系インメモリ時系列データベースの取得や監視規則アクションの実行も可能である．図 4 において，3.1 節の監視メトリクスは監視系インメモリ時系列データベース，3.2 節の監視規則は QoS 監視知識データベースに保持される．3.2 節の監視規則判定は QoS ソルバー，本提案手法の *μQoS* の推論処理エンジンは QoS 推論エンジンが担当する．図 4 の静的な QoS 監視規則は利用者や提供者が初期に設定する監視規則，動的な推論 QoS 監視規則が *μQoS* により導出された監視規則を示し，新たに QoS 監視知識データベースに追加され保持される．

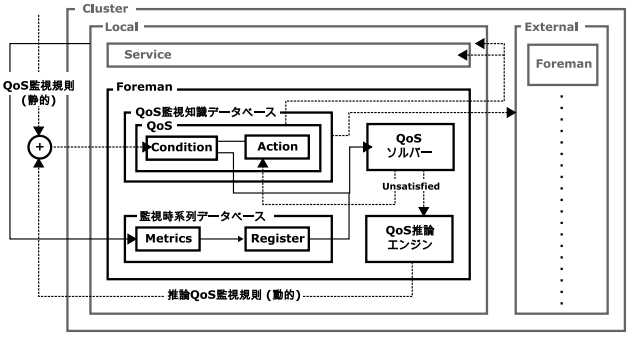

図 5　*μQoS* 自律型分散監視システム基盤

Fig. 5　*μQoS* autonomous distributed monitoring system infrastructure.

4.2 監視系インメモリ時系列データベース

Foreman は監視メトリクスを格納する時系列データベースを持ち，識別子，タイムスタンプ，値から構成される監視メトリクスを，**図 6** に示す抽象的インタフェースである循環メモリ内時系列バッファに格納され，保持期間が過ぎた監視メトリクスは自動的に削除される．

Foreman のコアフレームワークは C++，外部フレームワークは Go で実装されていいる．本循環型監視系インメモリ時系列データベースは抽象化インタフェースが定義されており，近年クラウド環境で用いられている Borgmon [8] および時系列圧縮型の Gorilla [6] 等を模した実装が提供され，運用者による選択が可能である．なお Foreman はストレージ型の監視系インメモリ時系列データベースである Graphite 互換のインタフェースを有しており，Graphite インタフェース対応サービスは改修の必要なく Foreman への監視メトリクスの収集が可能である．

4.3 監視規則アクション定義

QoS 監視規則のアクションは Python やスクリプト言語などの動的プログラミング言語を用いて定義する．*μQoS* では保持されている全期間の監視メトリクスを対象とした障害原因分析処理が必要となるが，Foreman の監視系インメモリ時系列データベースは Python から直接アクセス可能な配列として割り当てられており，*μQoS* では NumPy [55] や tslearn [56] などの科学計算用の既存ライブラリを活用とした分析が可能である．

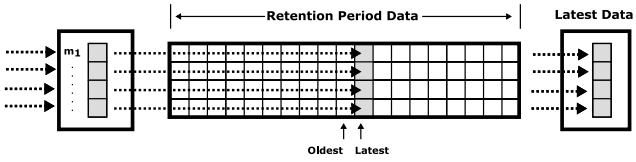

図 6　*μQoS* 循環型監視系インメモリ時系列データベース

Fig. 6　*μQoS* circular monitoring system in-memory time series database.

5. 実験

本章は，本稿で提案するイベント駆動型の自律的な復旧手法である *μQoS* および形状ベース時系列障害原因候補分析アルゴリズムである ShapeRoot の有効性を検証する．

5.1 評価対象および監視環境

本節では，本章の各実験に共通する監視環境や，評価対象となる障害分析手法の実装方法および評価指標など，全般的な共通事項について述べる．

5.1.1 監視環境

本実験は，評価対象となるクラウドサービスの監視メトリクスを Foreman インスタンスに収集した．Foreman インスタンスは対象サービスと同じノードで実行され，各サービスインスタンス単位に CPU 負荷やネットワーク負荷などのシステム情報とともにメトリクス情報を 5 分周期で収集した．なお，本評価の Foreman の時系列データベースには時系列圧縮された Gorilla [6] 相当ではなく，取得時に最も高速な Borgmon [8] 相当の非圧縮な C++ 実装を用いた．またすべての実験は，4 章の内部分散監視フレームワークの Foreman は g ++ 5.4.0，Go 1.12，Python 2.7.5 を使用してコンパイルされ，CentOS 7.6（Intel Core i7-4785T CPU @ 2.20 GHz，32 GB メモリ）上で実行された．

5.1.2 評価対象サービス

本実験は，典型的なクラウドサービスとして**表 1** に示す NoSQL データベースに分類される集中型の Redis [57] および分散型データベースである Apache Cassandra [53] を選択した．本実験において，各評価サービスは 5 分周期にサービスが稼働しているノードの CPU 負荷情報とともに Foreman の Graphite 互換インタフェース経由で監視メトリクスを収集した．

Redis は単体インスタンスで動作する小規模なインメモリデータベースであり収集される監視メトリクス（*Em*）は 78 種類，Cassandra は複数インスタンスで動作する永続型の大規模なデータベースシステムでありインスタンスごとに収集される *Em* は 11,754 種類であった．なお Redis は公式では Graphite 未対応のためクライアントツール（redis-cli）により出力される情報を Graphite 互換インタフェースに送信，Cassandra は公式の Graphite 送信モジュールにて監視メトリクスを収集した．

表 1　実験評価対象サービス
Table 1　Experiment evaluation services.

評価対象サービス	バージョン	収集メトリクス数
Apache Cassandra [53]	3.11.6	11,754
Redis [57]	3.2.13	78

5.1.3 評価対象ワークロード

評価対象サービスのワークロードについては，NoSQL データベースの標準的ベンチマークツールである YCSB [58] を用いた．YCSB による定常的なワークロードを Apache Cassandra および Redis に定常的に実行しつつ，サービス評価しようとなる応答遅延の QoS 監視条件を各実験ごとに設定し，障害分析手法の契機とした．なお，Redis [57] の監視メトリクスには統計的な応答遅延情報がないため，本 YCSB の測定値を監視メトリクス（*Em*）として収集した．

5.1.4 評価対象障害分析手法および実装

本稿の障害原因分析手法である ShapeRoot の有効性と性能を，2.4 節で示した以下の主要な関連研究の障害原因分析アルゴリズムと比較評価した．

- PPMCC [25]：関連研究 [6], [19] で言及されている相関分析アルゴリズム．本評価では NumPy [55] の既存実装ライブラリを用いた．
- k-Shape [32]：時系列データに特化したクラスタリング分析アルゴリズム．本評価では tslearn [56] の既存実装ライブラリを用いた．
- BigRoot [27]：特異性に注目した分散処理劣化要因検出アルゴリズム．本評価では数値メトリクス用の障害原因候補アルゴリズムのみ実装し評価対象とした．

各障害原因分析手法は，監視条件不成立時に時系列監視メトリクスを対象とする 3.2 節に示した監視規則アクション関数として実行され，同一条件下での評価のため，ShapeRoot を含めすべて障害原因分析アルゴリズムは Python（v2.7.16）で実装した．本評価における各障害原因分析手法は，QoS 監視条件の不成立を契機とし，3.3 節の式 (1) に示した正規化された相関係数ベクトルを返却する．

なお，ShapeRoot は異なる時系列周期データ間の相関判定も可能なアルゴリズムだが，本実験では比較対象手法の正常動作を考慮し，分析対象の全監視メトリクスを 5 分周期で固定し，欠損データについても補正したものを用いた．

5.1.5 評価指標

障害原因分析手法の有効性の評価指標は，適合率（Precision），再現率（Recall）および F 値（F-Score）を用いた．

- 適合率：障害原因候補と予測され，実際に正しい予測候補の割合：$Precision = \frac{TP}{TP+FP}$.
- 再現率：実際に正しい障害原因候補のうち，実際に候補と予測された割合：$Recall = \frac{TP}{TP+FN}$.
- F 値：適合率と再現率の調和平均．$FScore = \frac{2(Recall \times Precision)}{Recall+Precision}$

ここで TP は正しく検出された障害原因候補数，FP は誤って検出された障害原因候補数，FN は検出されなかっ

た障害原因候補数である．本研究は運用者を非介在とする自動復旧および恒久的対策の自動化を最終的な目的としており，誤った障害原因候補による自動復旧はQoSへの影響ならびに更なる障害の誘因にもつながる可能性もあり，正しい障害原因候補を示す適合率を最も重要な指標として評価した．適合率は1.0に近づくほど誤った障害原因候補が少ないことを示す．再現率は適合率と背反的な関係性があり，F値は再現率と適合率の両方を考慮した指標となる．適合率およびF値も1.0に近づくほど良好な指標となるが，本研究では参考指標として算出した．

なお，本評価は各実験ごとにQoS監視条件の不成立を契機とする長期間の複数障害分析結果データを無作為に抽出し，各障害原因分析手法が示した上位20個の障害原因候補を対象に，各評価指標をリーブワンアウト（Leave One Out）相互検証法により算出した．

5.2　実験1：小規模監視環境下の障害原因特定手法評価

本実験では，対象とする監視メトリクスが比較的小規模なサービスに対して人工的に障害を発生させ，理想的な因果関係のある障害に対してのShapeRootの障害原因分析の性能および有効性を評価した．障害原因とする因果関係のある障害監視メトリクス生成に人工的な障害挿入手法を用いた．評価対象サービスには，監視メトリクス（Em）がインスタンス単位で78種類と小規模なRedis [57]を選択し，YCSB [58]によるワークロード（Workload A）を定常的に実行しつつ，人工的障害として定期周期でCPU負荷ツール（stress）を実行した．本実験は，QoS監視条件式として以下のワークロード応答遅延（$read\_latency$）を閾値とするQoS監視規則（$Rule_{11}$）を設定し，不成立時に各障害原因特定手法の分析処理を実行した．

$$Rule_{11} : \boldsymbol{ifnot}\ read\_latency < 30\,msec$$

$Rule_{11}$は，不成立時に障害原因分析を開始する単純な監視条件式であり，復旧アクションに関連付けられていない．**図7**に，人工的障害原因であるCPU負荷は30分ごとに実行した場合の$Rule_{11}$の応答遅延（Read Latency）とCPU負荷（CPU Load）メトリクス値のみの関連を示す．

図7にて障害挿入手法による$Rule_{11}$の応答遅延と30分ごとに実行したCPU負荷メトリクスの高い相関がみられた．

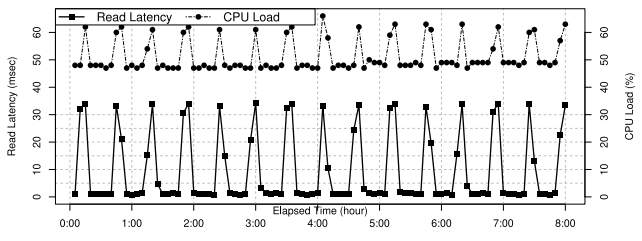

図7　小規模人工的障害監視メトリクス関連図

Fig. 7　Key monitoring metrics graph of failure injection experiment in a small service.

次に$Rule_{11}$から8時間前までの監視メトリクスを分析対象期間として，人工的な障害原因生成環境下の各障害原因候補検出手法を実行した．この結果を**表2**に示す．

小規模な人工的障害への分析結果としては，本稿の自動復旧および恒久的対策に重要な指標である適合率は，ShapeRootを含むいずれの障害分析手法も最高値を示し，いずれも正確な障害原因候補を示した．特に，k-Shapeについては分析時間は最長であるものの，再現率すべてにおいても最適な結果を示した．しかし，対象とする監視メトリクスが比較的小規模な本実験においては，適合率において各障害原因分析手法の有意差は確認できなかった．

5.3　実験2：大規模監視環境下の障害原因特定手法評価

本実験では，対象とする監視メトリクスが比較的大規模なサービスに対して人工的に障害を発生させ，理想的な因果関係のある障害に対してのShapeRootの障害原因分析の性能および有効性を評価した．評価対象サービスには，監視メトリクス（Em）がインスタンス単位で11,754種類と大規模なApache Cassandra [53]を選択し，実験1同様にYCSB [58]によるワークロード（Workload A）を定常的に実行しつつ，人工的障害として定期周期でCPU負荷ツール（stress）を実行した．本実験は，QoS監視条件式として以下のワークロード応答遅延（$read\_latency$）を閾値とするQoS監視規則（$Rule_{21}$）を設定し，不成立時に各障害原因特定手法の分析処理を実行した．

$$Rule_{21} : \boldsymbol{ifnot}\ read\_latency < 5\,msec$$

$Rule_{21}$は，不成立時に障害原因分析を開始する単純な監視条件式であり，復旧アクションに関連付けられていない．**図8**に，人工的障害原因であるCPU負荷は30分ごとに実行した場合の$Rule_{21}$の応答遅延（Read Latency）とCPU負荷（CPU Load）メトリクス値のみの関連を示す．

実験1同様，図8においても障害挿入手法による$Rule_{21}$の応答遅延と30分ごとに実行したCPU負荷メトリクスの高い相関がみられた．

次に$Rule_{21}$不成立時から8時間前までの監視メトリクスを分析対象期間として，人工的な障害原因生成環境下の各障害原因候補検出手法を実行した．この結果を**表3**に

表2　人工的障害への小規模障害原因候補分析結果

Table 2　Effect of root cause detection for injected faults in a small service.

	ShapeRoot	k-Shape [32]	BigRoot [27]	PPMCC [25]
分析時間 (秒)	0.915	1.556	0.162	0.153
適合率	1.0	1.0	1.0	1.0
再現率	0.585	1.0	0.566	0.769
F 値	0.735	1.0	0.640	0.869

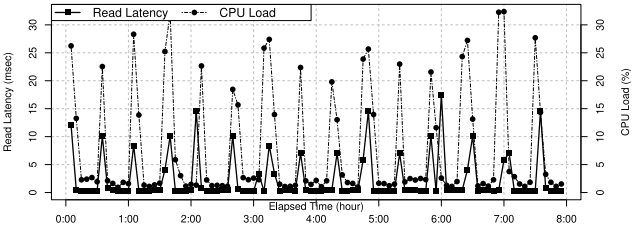

図 8　大規模人工的障害監視メトリクス関連図

Fig. 8　Key monitoring metrics graph of failure injection experiment in a big service.

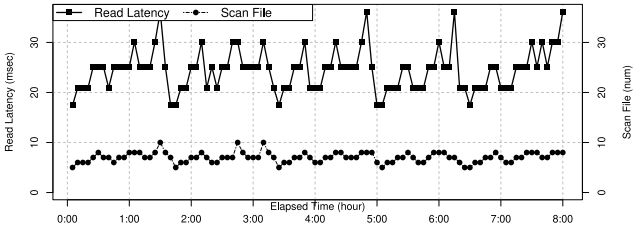

図 9　大規模現実的障害メトリクス関連図

Fig. 9　Key monitoring metrics graph of real failure experiments in a big service.

表 3　人工的障害への大規模障害原因候補分析結果

Table 3　Effect of root cause detection for injected faults in a big service.

	ShapeRoot	k-Shape [32]	BigRoot [27]	PPMCC [25]
分析時間 (秒)	174.02	1488.28	3.02	2.02
適合率	0.965	0.992	0.822	0.995
再現率	0.398	0.392	0.088	0.750
F 値	0.564	0.555	0.159	0.852

表 4　現実的障害への障害原因候補分析結果

Table 4　Effect of root cause detection for a big real service.

	ShapeRoot	k-Shape [32]	BigRoot [27]	PPMCC [25]
分析時間 (秒)	172.74	1,500.06	3.05	2.09
適合率	0.975	0.981	0.834	0.593
再現率	0.311	0.274	0.107	0.568
F 値	0.472	0.385	0.191	0.561

示す.

人工的障害への分析結果は，PPMCC が分析時間，適合率，再現率すべてにおいて最適な結果を示した．本稿の自動復旧および恒久的対策に重要な指標である適合率は PPMCC，ShapeRoot，k-Shape が優位な結果を示した.

PPMCC との比較：理想的な障害挿入環境下では，PPMCC は ShapeRoot を含むすべての障害原因分析手法よりも高い適合率と再現率を示した．また，処理速度についても動的な実時間の自動復旧に適する分析時間結果を示した.

k-Shape との比較：ShapeRoot は，k-Shape より 8 倍高速かつ同等の適合率と再現率を示した．k-Shape は ShapeRoot よりも若干高い適合率と再現率を示したが，分析時間から動的な実時間での分析は困難な結果を示した.

BigRoot との比較：ShapeRoot は BigRoot よりも高い適合率と再現率を示した．本結果は BigRoot が元来分散処理での劣化要因特定を目的としており，本稿が対象とする障害原因候補分析には適さないことを示唆した.

理想的な障害挿入環境下では，PPMCC が分析時間，適合率，再現率すべてにおいて最適な障害原因候補検出手法であった．ShapeRoot は再現率は PPMCC に劣るものの，PPMCC と k-Shape と同等の適合率，実時間処理可能な分析時間結果を示した.

5.4　実験 3：現実的障害下の障害原因特定手法評価

本実験は，実験 2 の理想的な障害発生環境下での評価に加え，現実的な障害に対する ShapeRoot の性能および有効性を評価した．現実的な障害監視メトリクス生成には，Apache Cassandra [53] のデータ削除に関するアンチパターン [59] を用いた．Apache Cassandra は追記型のストレージ構造を持ち，データ更新または削除により論理的な削除データ（Tombstone）が蓄積され，その蓄積により応答遅延が悪化する [59]．本実験は YCSB [58] にて大規模データ更新の継続的なワークロード負荷を生成し，応答遅延を劣化させる現実的な障害状況下で評価した．本実験は，実験 2 同様に QoS 監視条件式として以下のワークロード応答遅延（*read_latency*）を閾値とする QoS 監視規則（$Rule_{31}$）を設定し，不成立時に各障害原因特定手法の分析処理を実行した.

$$Rule_{31} : \boldsymbol{ifnot}\ read\_latency < 30\,msec$$

本実験の負荷により，QoS 監視条件式の応答遅延は約 4 時間ごとに悪化が観察され，不成立が発生した．Apache Cassandra [53] では直接的な論理削除データ数は把握できないが，関連メトリクスとしてデータ取得時の走査ファイル数の増加が間接的に観察される [59]．運用者はこの走査ファイル数の増加を管理対象に，定期的に物理削除し安定的なサービス提供を保証する [59]．**図 9** に，現実的障害原因のワークロード負荷を実行した場合の $Rule_{31}$ の応答遅延（Read Latency）と走査ファイル（Scan File）メトリクス値のみの関連を示す.

現実的な障害環境下の図 9 は，人工的な障害挿入環境下の実験 2 ほど，目視的には応答遅延と論理削除数との高い相関関係は示さなかった.

本実験は，実験 2 と同様に，$Rule_{31}$ は不成立時に障害原因分析を開始する単純な監視条件式であり，実験 2 同様に $Rule_{31}$ 不成立時から監視系インメモリ時系列データベースの 8 時間分の監視メトリクスを対象に，各障害原因候補検出手法を評価した．**表 4** に各障害原因候補検出手法の評価結果を示す.

本稿の重要指標である適合率は，実験 2 の人工的障害分析結果と比較し ShapeRoot，k-Shape，BigRoot は同等に

有意な水準を保ったが，PPMCC は顕著な低下を示した．また，再現率についても，いずれの手法も人工的障害分析結果と比較し低下を示したが，分析時間についてはいずれの手法も同等で，現実的な環境下においても処理速度の低下は示さなかった．

PPMCC との比較：実験 2 とは異なり，現実的な障害環境下では，PPMCC は高い適合率を示さず，ShapeRoot が PPMCC よりも高い適合率を示した．本結果は，本稿が提案した ShapeRoot の時系列形状ベースのアルゴリズムが，現実的な障害環境下ではより適切な障害原因分析手法であることを示唆した．

k-Shape との比較：実験 2 と同様，ShapeRoot は，k-Shape より 8 倍高速かつ同等の適合率と再現率を示した．ただし，再現率においては ShapeRoot が k-Shape よりやや優位な傾向が確認された．適合率については，実験 2 と同様に k-Shape が ShapeRoot より優位であったが，分析時間から動的な実時間での分析は困難な結果を示した．

BigRoot との比較：実験 2 と同様，ShapeRoot は BigRoot よりも高い適合率と再現率を示した．本結果は実験 2 同様，現実的な障害環境下でにおいても本稿が対象とする障害原因候補分析には BigRoot が適さないことを示唆した．

現実的な障害環境下ででも，ShapeRoot は他の障害原因分析方法と比較し，人工的障害分析結果と同等な適合率および処理速度結果を示した．特に，人工的障害分析結果で優位を示した PPMCC は，現実的な環境下において適合率の低下に課題を示した．

5.5　実験 4：障害原因分析期間と適合率

実験 2 および実験 3 にて優位であった障害原因候補検出手法の ShapeRoot および k-Shape のみを対象に，監視系インメモリ時系列データベースの分析期間と評価指標の関係性を検証した．クラウド環境のインメモリ時系列データベースの保持期間はメモリ容量制約から有限なリソースであり，各監視系インメモリデータベースの関連研究には各用途および目的に応じた適切な保持期間が評価されている [5], [6], [8]．本実験においても，実験 3 と同じ現実的な障害環境下で最適な監視データ保持期間を検証した．ShapeRoot および k-Shape の分析対象期間と各評価指標との関係性を**表 5** に示す．

実験 3 の図 9 に示すとおり，本実験の $Rule_{31}$ の監視条件式は約 4 時間ごとに悪化し不成立となる．ShapeRoot は，対象とする分析対象期間が長いほど適合率および再現率ともに改善され，有意な障害分析結果を得るにはある一定の分析期間の必要性を示唆した．

k-Shape との比較：ShapeRoot の結果とは対照的に，k-Shape では対象分析期間と適合率および再現率に有意な関係性は観察できなかった．また，本実験の障害発生周期期間の 4 時間以下の分析対象期間では，k-Shape は ShapeRoot より優位な適合率を示した．ただし，ShapeRoot の分析時間は全分析対象区間で k-Shape より 6 倍以上の処理速度を示した．さらに，両手法の 1 時間の分析対象時間を基準とすると，k-Shape の 8 時間の分析時間が $2^{6.9}$ 倍に対し ShapeRoot は $2^{5.76}$ 倍と，ShapeRoot は分析対象区間増加に対する分析時間の指数的増加も緩やかであった．

表 5　障害原因候補検出手法と分析対象期間
Table 5　Effect of root cause detection by analysis method and period.

	ShapeRoot				k-Shape [32]			
	Retention Period (Hour)				Retention Period (Hour)			
	1	2	4	8	1	2	4	8
分析時間 (秒)	5.18	12.65	45.12	172.74	31.50	466.80	840.06	1500.06
適合率	0.800	0.905	0.955	0.975	0.944	0.985	0.924	0.981
再現率	0.096	0.176	0.238	0.311	0.180	0.210	0.141	0.274
F 値	0.172	0.295	0.381	0.472	0.282	0.322	0.223	0.385

本実験で，分析対象期間を障害発生周期期間以上とした場合には ShapeRoot は k-Shape と同等の適合率および再現率を示した．また，ShapeRoot の分析時間は全分析対象区間で k-Shape より高速で，分析対象区間の増加に対しての分析時間の指数関数的成長も緩やかであった．ただし，障害発生周期期間未満を分析対象期間とした場合，ShapeRoot は適合率は低く，k-Shape などの適合率の高い他障害原因候補検出手法との組み合わせの必要性が示唆された．

5.6　実験 5：発生障害への自律的復旧効果

最後に，*μQoS* の適用事例として，現実的な障害に対して ShapeRoot による障害原因候補分析結果および *μQoS* による推論 QoS 監視規則導出により，即時の自律的復旧による QoS 保証の有効性を検証した．本実験は，1 章で述べたクラウド環境にてサービス提供者および利用者が個別に QoS 監視規則を設定する状況を模擬し，以下の QoS 監視規則 $Rule_{41}$ および $Rule_{42}$ を初期状態として設定した．なお，QoS 監視規則以外の負荷条件については実験 3 と同一である．

$$Rule_{41} : \boldsymbol{ifnot}\ read\_latency < 30\,msec$$
$$Rule_{42} : \boldsymbol{ifnot}\ read\_scan\_file\_num < 8$$
$$\boldsymbol{then}\ (compact\_tombstones)$$

$Rule_{41}$ は利用者の QoS 監視規則設定を模擬している．おおむね利用者はサービスの SLO 保証にのみ関心があり，SLO を保証および復旧手法には関心がない．そのため，$Rule_{41}$ は応答時間（*read_latency*）のみを監視条件式とする QoS 監視規則であり復旧アクションは未定義とした．一方，$Rule_{42}$ は提供者の QoS 監視規則設定を模擬している．提供者は豊富なシステム運用知識を有し，$Rule_{42}$ の監視条件式および復旧アクションは，実験 3 に示した走査ファイル数（*read_scan_file_num*）が増加するアンチパ

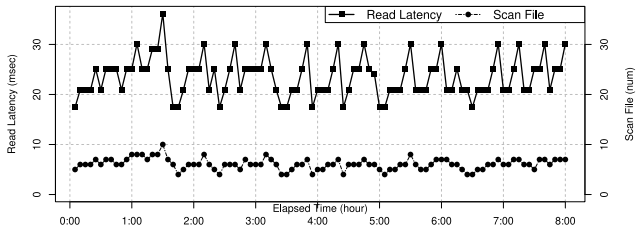

図 10　監視設定条件式関連メトリクス図

Fig. 10　Effect of applying an approximate compositional QoS rule.

ターンな障害要因に対して，論理削除済み（Tombstone）データを物理的に削除する復旧アクション（*compact_tombstones*）を定義した [59]．また，本模擬監視設定においても，利用者 $Rule_{41}$ と提供者 $Rule_{42}$ は個別に設定され，両者の QoS 監視規則に明確な関連性はない．**図 10** に，実験 3 と同じ現実的障害原因のワークロード負荷を実行した場合の $Rule_{41}$ の応答遅延（Read Latency）と $Rule_{42}$ の走査ファイル（Scan File）メトリクス値のみの関連を示す．

本実験において，図 10 の最初の 2 時間で $Rule_{41}$ の監視条件式が不成立となり，3.4 節に述べた *μQoS* による推論に基づいて，新たな推論 QoS 監視規則として，以下の $Rule_{33}$ の生成が確認された．

$$\begin{aligned} Rule_{33} &= (Rule_{41} \circ Rule_{42}) \\ &= \boldsymbol{ifnot}\ read\_latency < 30\,msec \\ &\quad\ \boldsymbol{then}\ (compact\_tombstones) \end{aligned}$$

$Rule_{33}$ の推論監視規則生成は，ShapeRoot による障害原因分析手法および *μQoS* の $Rule_{41}$ と $Rule_{42}$ のメトリクス因果関係発見により自律的な復旧規則が生成されたことを示した．また，実験 3 の図 9 に示すとおり，目視的に $Rule_{41}$ の応答遅延と $Rule_{42}$ の論理削除数の相関関係は判断は困難だが，実験 3 における ShapeRoot の障害原因分析結果には，その形状類似度から $Rule_{42}$ の監視メトリクス（*read_scan_file_num*）を含む論理削除数関連メトリクスが，分析対象となる総監視メトリクス数 11,754 個の中から，高適合率で常に上位 20 位以内に十数件含まれていることも併せて確認できた．$Rule_{33}$ は 3.4.2 項で述べた，過去に実行すべきである $Rule_{\{unsat \circ root\}}$ の導出に相当する．また，図 10 は *μQoS* による $Rule_{33}$ 追加後，即時の $Rule_{41}$ 監視条件式の成立を示しており，*μQoS* による即時復旧効果を示した．さらに図 10 は，*μQoS* による自律的推論が動作していない実験 3 の図 9 と比較し，最初の $Rule_{41}$ が不成立後の経過 2 時間以降においては安定した $Rule_{41}$ 監視条件式の成立および QoS の保証を示しており，*μQoS* による障害の再発防止および恒久対策の可能性も示した．

本実験は，*μQoS* および ShapeRoot が現実的な障害環境下においても，不成立な QoS 監視規則から障害原因候補を推論し，初期の QoS 監視規則を自律的に補完し即時復旧への有効性を示した．ただし，*μQoS* により正常な効果が確認できた推論 QoS 監視規則については事例ベース推論 [44] により正式な QoS 監視規則としての事例格納を想定しているが，恒久的な QoS 監視規則としての評価および自律的な事例格納については実運用での評価を含め今後の課題である．

6.　考察

クラウド環境の QoS 保証において信頼性の高い自律的復旧および正確な障害原因が必要とされる [8], [9]．本提案手法である ShapeRoot は現実的な時系列障害分析手法として，従来の時系列障害分析手法 [25], [27], [32] と比較し，異周期の時系列データ間や分析対象データの欠損も考慮され，評価実験にて高適合率かつ実時間で高速に動作する障害原因分析性能を示した．ただし，ShapeRoot の高適合率には障害原因に対して適切な期間の時系列データが必要とされ，短期間の突発的な障害に対しては k-Shape などクラスタリング分析などより適合率の高い手法との組み合わせの必要性も示唆された．また，本稿で提示した ShapeRoot は単純な距離による形状類似判定手法のため，主に時系列的に正の相関関係があるメトリクス間の適用に限定される課題がある．たとえば幾何的に相似的類似性のあるメトリクスであっても時系列的に負の相関関係がある場合は距離的に離れており障害根本原因候補として判定されず，本提案手法の時系列形状の類似判定には幾何的なアフィン変換の考慮などの更なる改良が必要である．

μQoS は，ShapeRoot による高適合率な障害原因分析結果から現実的障害環境下での即時の自律復旧による有用性も示したが，発生障害の再発防止および恒久的な有用性については更なる検証が必要である．また，*μQoS* の導出する推論規則は，暗黙的に既存の定義済み監視規則アクションの並行動作の独立性や冪等性を前提としており，本手法の適用により同じ監視規則アクションが並行または繰り返して実行される環境下での副作用や抑制手法については更なる検討が必要である．

なお本提案手法は既存 QoS 監視規則を合成する手法であり，全く新しい QoS 監視規則自体を創出する手法ではない．ゆえに，本提案手法はシステムが安定的に運用され豊富な QoS 監視規則が整備されている環境下で有効な手法ではあるが，新規運用のシステムまたはシステム自体が不安定な場合にや有効な復旧監視規則が導出できず適用が難しい．更に，本稿は発生障害からの時系列的に過去の障害原因分析のみに焦点を当てたが，モデルベース推論または異常検出に基づく将来における障害予測は潜在的な障害の防止に有効であり [9], [30], [33]，更なる研究が必要である．

7.　結論

クラウド環境における QoS 保証は重要な課題であり，

即時の障害復旧には自動化が不可欠である．また発生障害の再発防止に向けた恒久的対処も合わせて重要であり，自動化による障害復旧には，正確かつ実時間で動作可能な障害原因特定に基づく復旧手法が求められる．

本稿は，*μQoS* と呼ばれる事例ベース推論と障害原因分析に基づく新しいイベント駆動型の自律的な監視条件式推論手法および，インメモリ時系列監視データベース向けに最適な障害原因分析手法として，形状ベース時系列障害原因分析アルゴリズムとして ShapeRoot を提案し有用性を示した．

最後に，本稿に限らず QoS 監視規則や障害復旧手法などの運用知識は，組織内外で共有化されることが望ましい．本稿は今後の実運用における運用知識ベースの構築を目的にしており，将来的に本稿の基盤となった分散監視システムと合わせ公開することで，今後の更なる発展に期待している．

参考文献

[1] Mell, P., Grance, T. et al.: The NIST definition of cloud computing (2011).

[2] Aceto, G., Botta, A., De Donato, W. and Pescapè, A.: Cloud monitoring: A survey, *Computer Networks*, Vol.57, No.9, pp.2093–2115 (2013).

[3] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J.: Large-scale cluster management at Google with Borg, *Proceedings of the Tenth European Conference on Computer Systems*, ACM, p.18 (2015).

[4] Konno, S.: Cassandra@Yahoo Japan, *Cassandra Summit* (2016).

[5] Harrington, B. and Rapoport, R.: Introducing Atlas: Netflix's Primary Telemetry Platform (2014).

[6] Pelkonen, T., Franklin, S., Teller, J., Cavallaro, P., Huang, Q., Meza, J. and Veeraraghavan, K.: Gorilla: A fast, scalable, in-memory time series database, *Proceedings of the VLDB Endowment*, Vol.8, No.12, pp.1816–1827 (2015).

[7] Fox, A. and Patterson, D.: Self-repairing computers, *Scientific American*, Citeseer (2003).

[8] Beyer, B.: *Site reliability engineering: How Google runs production systems*, O'Reilly Media, Sebastopol, CA (2016).

[9] Salfner, F., Lenk, M. and Malek, M.: A survey of online failure prediction methods, *ACM Computing Surveys (CSUR)*, Vol.42, No.3, p.10 (2010).

[10] Fatema, K., Emeakaroha, V. C., Healy, P. D., Morrison, J. P. and Lynn, T.: A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives, *Journal of Parallel and Distributed Computing*, Vol.74, No.10, pp.2918–2933 (2014).

[11] He, S., Lin, Q., Lou, J.-G., Zhang, H., Lyu, M. R. and Zhang, D.: Identifying impactful service system problems via log analysis, *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ACM, pp.60–70 (2018).

[12] Serrano, D., Bouchenak, S., Kouki, Y., de Oliveira Jr, F. A., Ledoux, T., Lejeune, J., Sopena, J., Arantes, L. and Sens, P.: SLA guarantees for cloud services, *Future Generation Computer Systems*, Vol.54, pp.233–246 (2016).

[13] Abdelmaboud, A., Jawawi, D. N., Ghani, I., Elsafi, A. and Kitchenham, B.: Quality of service approaches in cloud computing: A systematic mapping study, *Journal of Systems and Software*, Vol.101, pp.159–179 (2015).

[14] Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J. and Fu, C.: Cloud computing: a perspective study, *New Generation Computing*, Vol.28, No.2, pp.137–146 (2010).

[15] Syed, H. J., Gani, A., Ahmad, R. W., Khan, M. K. and Ahmed, A. I. A.: Cloud monitoring: A review, taxonomy, and open research issues, *Journal of Network and Computer Applications* (2017).

[16] Konno, S. and Défago, X.: Approximate QoS Rule Derivation Based on Root Cause Analysis for Cloud Computing, *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE, pp.33–3309 (2019).

[17] Kephart, J. O. and Chess, D. M.: The vision of autonomic computing, *Computer*, Vol.36, No.1, pp.41–50 (2003).

[18] Dickson, C. L.: A working Theory of Monitoring, LISA (2013).

[19] Loboz, C., Smyl, S. and Nath, S.: DataGarage: Warehousing massive performance data on commodity servers, *Proceedings of the VLDB Endowment*, Vol.3, No.1–2, pp.1447–1458 (2010).

[20] Julisch, K.: Clustering intrusion detection alarms to support root cause analysis, *ACM transactions on information and system security (TISSEC)*, Vol.6, No.4, pp.443–471 (2003).

[21] Wang, M., Holub, V., Parsons, T., Murphy, J. and O'Sullivan, P.: Scalable run-time correlation engine for monitoring in a cloud computing environment, *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, IEEE, pp.29–38 (2010).

[22] Corporation, E.: Automating Root-Cause Analysis: EMC Ionix Codebook Correlation Technology vs. Rules-based Analysis Technology Concepts and Business Considerations (2009).

[23] Fu, X., Ren, R., Zhan, J., Zhou, W., Jia, Z. and Lu, G.: LogMaster: mining event correlations in logs of large-scale cluster systems, *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, IEEE, pp.71–80 (2012).

[24] Rouillard, J. P.: Real-time log file analysis using the Simple Event Correlator (SEC) (2004).

[25] Pearson, K.: Note on regression and inheritance in the case of two parents, *Proceedings of the Royal Society of London*, Vol.58, pp.240–242 (1895).

[26] Mueen, A., Nath, S. and Liu, J.: Fast approximate correlation for massive time-series data, *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ACM, pp.171–182 (2010).

[27] Zhou, H., Li, Y., Yang, H., Jia, J. and Li, W.: BigRoots: An Effective Approach for Root-Cause Analysis of Stragglers in Big Data System, *IEEE Access*, Vol.6, pp.41966–41977 (2018).

[28] Ma, M., Xu, J., Wang, Y., Chen, P., Zhang, Z. and Wang, P.: Automap: Diagnose your microservice-based web applications automatically, *Proceedings of The Web Conference 2020*, pp.246–258 (2020).

[29] Qiu, J., Du, Q., Yin, K., Zhang, S.-L. and Qian, C.: A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud

applications, *Applied Sciences*, Vol.10, No.6, p.2166 (2020).
[30] Aghabozorgi, S., Shirkhorshidi, A. S. and Wah, T. Y.: Time-series clustering-A decade review, *Information Systems*, Vol.53, pp.16–38 (2015).
[31] Rai, P. and Singh, S.: A survey of clustering techniques, *International Journal of Computer Applications*, Vol.7, No.12, pp.1–5 (2010).
[32] Paparrizos, J. and Gravano, L.: k-shape: Efficient and accurate clustering of time series, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM, pp.1855–1870 (2015).
[33] Chandola, V., Banerjee, A. and Kumar, V.: Anomaly detection: A survey, *ACM computing surveys (CSUR)*, Vol.41, No.3, p.15 (2009).
[34] Anithakumari, S. and Chandrasekaran, K.: Monitoring and management of service level agreements in cloud computing, *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, IEEE, pp.204–207 (2015).
[35] Al-Shammari, S. and Al-Yasiri, A.: MonSLAR: a middleware for monitoring SLA for RESTFUL services in cloud computing, *Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA), 2015 IEEE 9th International Symposium on the*, IEEE, pp.46–50 (2015).
[36] Calero, J. M. A. and Aguado, J. G.: MonPaaS: an adaptive monitoring platformas a service for cloud computing infrastructures and services, *IEEE Transactions on Services Computing*, Vol.8, No.1, pp.65–78 (2015).
[37] Zhao, L., Sakr, S. and Liu, A.: A framework for consumer-centric SLA management of cloud-hosted databases, *IEEE Transactions on Services Computing*, Vol.8, No.4, pp.534–549 (2015).
[38] 西野博之：大規模データセンターにおける運用ノウハウ共有による障害再発防止方式 (2014).
[39] Russell, S. J. and Norvig, P.: *Artificial intelligence: a modern approach*, Malaysia; Pearson Education Limited, (2016).
[40] 人工知能学会：人工知能学大事典 = *Encyclopedia of artificial intelligence*, 共立出版 (2017).
[41] Russell, S., Norvig, P. and Intelligence, A.: A modern approach, *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, Vol.25, p.27 (1995).
[42] Buchanan, B. G., Shortliffe, E. H. et al.: *Rule-based expert systems*, Vol.3, Addison-Wesley Reading, MA (1984).
[43] 工藤　裕，森村知弘，菅内公徳，薦田憲久：障害原因解析のためのルール記述方法とその実行方式，電気学会研究会資料．*IS*，情報システム研究会 , Vol.2009, No.71, pp.1–6 (2009).
[44] Kolodner, J.: *Case-based reasoning*, Morgan Kaufmann (2014).
[45] Aamodt, A. and Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches, *AI communications*, Vol.7, No.1, pp.39–59 (1994).
[46] Wu, X., Turner, D., Chen, C.-C., Maltz, D. A., Yang, X., Yuan, L. and Zhang, M.: NetPilot: automating datacenter network failure mitigation, *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pp.419–430 (2012).
[47] Emeakaroha, V. C., Netto, M. A., Calheiros, R. N., Brandic, I., Buyya, R. and De Rose, C. A.: Towards autonomic detection of SLA violations in Cloud infrastructures, *Future Generation Computer Systems*, Vol.28, No.7, pp.1017–1029 (2012).
[48] Nersessian, N. J.: Model-based reasoning in conceptual change, *Model-based reasoning in scientific discovery*, Springer, pp.5–22 (1999).
[49] Zadeh, L. A.: Fuzzy logic and approximate reasoning, *Synthese*, Vol.30, No.3–4, pp.407–428 (1975).
[50] Takagi, T. and Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control, *IEEE transactions on systems, man, and cybernetics*, No.1, pp.116–132 (1985).
[51] Sakoe, H., Chiba, S., Waibel, A. and Lee, K.: Dynamic programming algorithm optimization for spoken word recognition, *Readings in speech recognition*, Vol.159, p.224 (1990).
[52] Pearl, J.: Causality: models, reasoning, and inference, *Econometric Theory*, Vol.19, No.675–685, p.46 (2003).
[53] Lakshman, A. and Malik, P.: Cassandra: a decentralized structured storage system, *ACM SIGOPS Operating Systems Review*, Vol.44, No.2, pp.35–40 (2010).
[54] Gusfield, D.: Algorithms on stings, trees, and sequences: Computer science and computational biology, *Acm Sigact News*, Vol.28, No.4, pp.41–60 (1997).
[55] Van Der Walt, S., Colbert, S. C. and Varoquaux, G.: The NumPy array: a structure for efficient numerical computation, *Computing in Science & Engineering*, Vol.13, No.2, p.22 (2011).
[56] Tavenard, R.: tslearn: A machine learning toolkit dedicated to time-series data (2017).
[57] Carlson, J. L.: *Redis in action*, Manning Publications Co. (2013).
[58] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking cloud serving systems with YCSB, *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, pp.143–154 (2010).
[59] Strickland, R.(ed.): *Cassandra 3.x High Availability - Second Edition*, Packt Publishing (2016).

今野 賢

2004 年放送大学教養学部卒業．2005 年ヤフー株式会社入社，デジタル家電およびスマートフォン向けサービス開発業務，分散データベースに関する研究開発業務に従事．2013 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．2015 年北陸先端科学技術大学院大学情報科学研究科博士後期課程．

デファゴ クサヴィエ（正会員）

1995 年スイス連邦工科大学・ローザンヌ校（EPFL）卒業．2000 年同大学博士課程修了．理学博士．2000 年北陸先端科学技術大学院大学助手．2003 年同特任助教授．2006 年同准教授．2016 年東京工業大学情報理工学院教授．分散システム，高信頼性，群ロボットの研究に従事．IFIP WG10.4 会員．IEEE，IEEE-CS，ACM 各会員．

冨田 尭（正会員）

2007 年東京工業大学工学部情報工学科卒業．2013 年同大学大学院情報理工学研究科博士後期課程修了．博士（工学）．2013 年同研究員．2015 年北陸先端科学技術大学院大学情報社会基盤研究センター助教．2020 年から同講師．仕様検証，モデル検査，プログラム検証，自動合成等の形式手法およびその車載システム開発への応用の研究に従事．日本ソフトウェア科学会，日本バイオインフォマティクス学会，IEEE，ACM，情報処理学会各会員．

井口 寧（正会員）

1991 年東北大学工学部機械工学科卒業．1994 年〜1997 年日本学術振興会特別研究員．1997 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．同大学情報科学センター助手，准教授を経て，現在情報社会基盤研究センター教授．また，2002 年から 2006 年まで科学技術振興機構さきがけ研究 21（機能と構成）に参加し研究に従事．2008 年〜2009 年米国南フロリダ大学上級客員研究員．この間 FPGA などを用いた並列処理や並列システムのハードウェアやソフトウェアに関する研究を行う．IEEE，日本バーチャルリアリティ学会各会員，情報処理学会，電子情報通信学会各シニア会員．