アプリケーションの細粒度原因分析に基づくクラウドサービス障害対処手法の検討

名倉 正剛[1,a)]　平島 陽子[2]

概要：サーバ仮想化機構の普及により，仮想サーバを利用したクラウドサービスの提供が容易になっている．サービスを提供するインフラに障害が発生した場合，サービス停止やサービスの利用品質低下が発生するため，障害対処には特に迅速な対応が求められる．オートスケール技術のような自動化技術では，サーバのパフォーマンスに起因した障害への自動的な対処が可能であるが，実際の運用管理現場では，サーバ上で動作するアプリケーションのボトルネックに応じた対処作業が必要になる場合が多い．そこで本研究では，障害発生時にアプリケーション実行を細粒度で分析することで原因となる処理を特定し，あらかじめ定義しておいた回避策を自動的に実行する手法を検討する．

1. はじめに

IT システムの規模は年々拡大し，クラウド化により構成要素となる機器の種類も増大し，構成も複雑化している．豊富な知識や経験を持つシステム運用管理者を十分に確保できないことも多く，IT システムの自律的な運用管理の実現が求められている．そこで本研究では，クラウドサービスを構成するアプリケーションの障害発生時に，その実行過程を細粒度で分析することで，原因となる処理の特定と対応する回避策を自動的に実行する手法を検討する．

2. 障害対処作業自動化の既存技術と課題

IT システムの障害発生時の障害対処作業を支援する研究のうち，障害原因箇所の特定のための研究として，IT システムで障害を検知した場合に障害原因を解析する障害原因解析技術 [1][2] がある．また，あらかじめ定義された障害事象と回避策の関連に基づき，回避するための対処プランを生成する回避策推論システムも提案されている [3][4]．Amazon などのクラウドインフラを提供する事業者は，パフォーマンス障害が発生した場合に備え，自動でスケールを調整するオートスケール技術を用意している [5]．

これらの技術を利用することで，サーバやネットワーク機器などのシステムを構成する機器や，仮想サーバなどの仮想化インフラに障害が発生した場合に，自動で障害を回避可能である．しかし実際の運用管理現場での障害対処はこれだけでは十分ではなく，サーバ上で動作するアプリケーションのボトルネック箇所を特定し，特定された箇所に応じた対処作業が必要になることが多い．

例えば，Web 3 層システムにおいて，DB 層の許容できる接続数を超えて，Web サーバ等のプレゼンテーション層がクライアントからのリクエストを受け付け，結果としてアプリケーションロジックの実行に遅延が起きた場合を考える．既存手法では，CPU 利用率や，アプリケーション応答時間等の，アプリケーション実行に対して外部から測定できるメトリクスに基づき原因を特定し，回避策を実行する．先述のオートスケール技術 [5] では，アプリケーション応答時間が遅延している場合に，アプリケーションサーバに対しスケールアウトを実施する．しかしこれによりさらに多くのアプリケーションが DB へ接続するようになり，接続数超過がさらに発生することになりかねず，適切な障害対処にはならない．

このように，アプリケーション実行に発生した障害に対する対処作業の自動化を考えた場合，既存技術では次の 2 点の課題が存在すると考える．

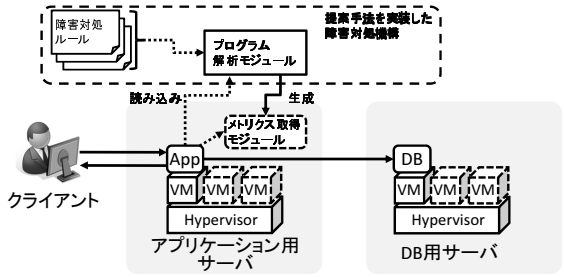
【課題 1】　アプリケーション粒度での原因分析（本研究では細粒度原因分析と呼ぶ）を実施できる必要がある．

【課題 2】　アプリケーション設定等のアプリケーション粒度での回避策を実行できる必要がある．

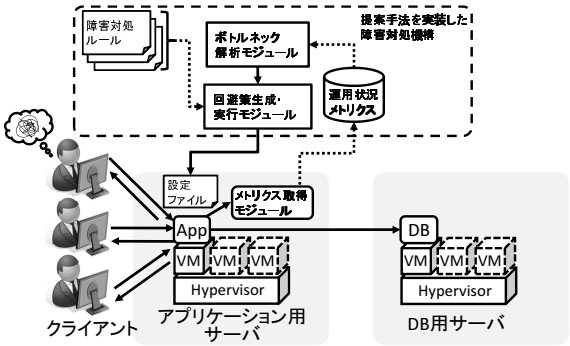
3. 提案手法

全体像を図 1 に示す．以降では障害対処のために記述

[1] 日本大学工学部 情報工学科
Department of Computer Science, College of Engineering, Nihon Univ., Koriyama, Fukushima, 963–8642, Japan
[2] 株式会社 日立製作所 研究開発グループ
Research & Development Group, Hitachi, Ltd., Yokohama, Kanagawa, 244–0817, Japan
[a)] nag@cs.ce.nihon-u.ac.jp



(a) プログラム解析に基づくメトリクス取得モジュールの生成



(b) 障害発生時の原因解析と回避策実行

図 1 提案方式の概要

する障害対処ルールと，それを利用したメトリクス取得モジュールの生成，障害発生時の回避策実行について述べる．

3.1 障害対処ルール

提案手法では，管理者はあらかじめ障害対処ルールとして，アプリケーション実行のボトルネック箇所と，それを回避する方法の対応を定義する．この記述例を，図 2 に示す．DB 接続プール不足時に観測される事象を，対象アプリケーションとボトルネック箇所のプログラム記述のパターンを示す名前の組として *bottleneck* 節に記述する．そして，それに対する回避策を *solution* 節に記述する．この例では DB サーバの実装により回避策が異なることを想定し，アプリケーション設定変更方法に対する複数の回避策を，*solution* 節に記述している．*solution* 節には観測事象に応じたさまざまな回避策を記述できる．例えば，リソース構成を変更することで観測されたボトルネックを解消できる場合には，ハイパーバイザに処理を指示するような回避策を，*solution* 節の記述によって指定する．

3.2 メトリクス取得モジュールの生成

プログラム解析モジュールは，3.1 節で定義した障害対処ルールと，監視対象のアプリケーションプログラムを読み込み，メトリクス取得モジュールを生成する（図 1(a)）．

障害対処ルールにボトルネック箇所として指定したプログラム記述パターンがプログラム内に存在する場合，ボトルネックが実際に発生しているかどうかを検査するため，

```
1  case db_pool_starvation: // DB 接続プール不足
2    bottleneck:   // ボトルネックの観測事象
3      webapp waiting_for_db_pool;
4    solution:     // 回避策
5      if %db_server% == "Oracle12":
6        db_server shorten_connectiontimeout;
7          // DB ドライバのタイムアウト値を短くする
8      else:
9        webapp reduce_maxconnections;
10         // クライアントからの接続数を絞る
```

図 2 障害対処ルールの記述例

生成したメトリクス取得モジュールを，該当部分の処理実行の前後に挿入する．サーバサイドで動作するアプリケーションは，変更できなかったり，メトリクス取得のために変更することが望ましくなかったりする場合が多い．このため，メトリクス取得モジュールをアスペクトとして生成する．これにより，アプリケーションを変更せずにメトリクス取得モジュールを呼び出すことができるようにする．

3.3 障害発生時の回避策実行

アプリケーション実行中は，メトリクス取得モジュールによって取得されたメトリクスをボトルネック解析モジュールが定期的に解析する．そしてベースラインに対して一定の割合を超過した時にボトルネック発生を検知する．そして，障害対処ルールから実際のシステム構成に合わせて具体化した回避策を生成し，実行する（図 1(b)）．

4. まとめ

クラウドサービスの障害発生時に，アプリケーションの細粒度原因分析に基づき障害対処を自動化する手法を検討した．現在は研究の途中段階であるが，障害対処ルールとして定義した管理ノウハウを利用することで，アプリケーションレベルでの障害対処が可能になると考える．

参考文献

[1] 永井崇之，名倉正剛：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，情報処理学会論文誌，Vol.54 No.3，pp.1109-1119 (2013).

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

[3] 加藤裕，敷田幹文：障害予測における最適な障害回避手段の提示法，情報処理学会 インターネットと運用技術シンポジウム 2012 論文集，pp.110-116 (2012).

[4] 永井崇之，名倉正剛，中島淳，平島陽子，森村知弘：IT システム向け障害対処プラン自動生成システムの検討，電子情報通信学会 技術研究報告，Vol.112，No.492，pp.125-130 (2013).

[5] Amazon Web Services, Inc.: AWS CloudFormation: User Guide (API Version 2010-05-15) (online), available from <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-ug.pdf> (accessed 2016-08-05).