(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5819543号
(P5819543)

(45)発行日　平成27年11月24日(2015.11.24)　(24)登録日　平成27年10月9日(2015.10.9)

(51)Int.Cl.		F I		
H04L 12/70	(2013.01)	H04L	12/70	100Z
H04L 12/24	(2006.01)	H04L	12/24	
G06F 11/34	(2006.01)	G06F	11/34	S
H04L 12/717	(2013.01)	H04L	12/717	

請求項の数 20　(全 14 頁)

(21)出願番号	特願2014-550553 (P2014-550553)
(86)(22)出願日	平成25年1月9日 (2013.1.9)
(65)公表番号	特表2015-509316 (P2015-509316A)
(43)公表日	平成27年3月26日 (2015.3.26)
(86)国際出願番号	PCT/US2013/020795
(87)国際公開番号	WO2013/112288
(87)国際公開日	平成25年8月1日 (2013.8.1)
審査請求日	平成26年6月30日 (2014.6.30)
(31)優先権主張番号	61/590,042
(32)優先日	平成24年1月24日 (2012.1.24)
(33)優先権主張国	米国(US)
(31)優先権主張番号	13/736,158
(32)優先日	平成25年1月8日 (2013.1.8)
(33)優先権主張国	米国(US)

(73)特許権者　504080663
エヌイーシー　ラボラトリーズ　アメリカ　インク
ＮＥＣ　Ｌａｂｏｒａｔｏｒｉｅｓ　Ａｍｅｒｉｃａ，　Ｉｎｃ．
アメリカ合衆国　０８５４０　ニュージャージー州　プリンストン　スイート　２００　インディペンデンス　ウェイ　４
(74)代理人　100123788
弁理士　宮崎　昭夫
(74)代理人　100127454
弁理士　緒方　雅昭

最終頁に続く

(54)【発明の名称】ネットワークデバッギング

(57)【特許請求の範囲】

【請求項１】

ネットワーク内のデータセンターで使用されるデバッギングシステムであって、

ネットワークコントローラーからトラフィック情報を収集することによって、ネットワークトラフィックをモニターするモニタリングエンジンと、

モニターされたログを使用して、アプリケーションシグネチャー、インフラストラクチャーシグネチャー、およびタスクシグネチャーをモデリングするモデリングエンジンと、

参照ログおよび問題ログを使用して、動作状態と非動作状態との間での前記アプリケーションシグネチャーの変化を検出し、前記タスクシグネチャーを使用して、前記変化を検証するデバッギングエンジンと、

トラブルシューティング情報を提供する提供部と、を有し、

前記アプリケーションシグネチャーの未知の変化は、前記インフラストラクチャーシグネチャーの変化に対する依存性を考慮することによって、既知の問題クラスと相関付けられる、デバッギングシステム。

【請求項２】

請求項１に記載のデバッギングシステムであって、

前記ネットワークはＯｐｅｎＦｌｏｗネットワークを有し、前記トラフィック情報はＰａｃｋｅｔＩｎメッセージおよびＰａｃｋｅｔＯｕｔメッセージの少なくとも１つを通して収集される、デバッギングシステム。

【請求項３】

請求項２に記載のデバッギングシステムであって、
前記アプリケーションシグネチャーは、連結グラフ（ＣＧ）、遅延分布（ＤＤ）、要素相互作用（ＣＩ）、編相関（ＰＣ）、およびフロー統計（ＦＳ）の少なくとも１つによってモデリングされる、デバッギングシステム。
【請求項４】
請求項３に記載のデバッギングシステムであって、
前記デバッギングエンジンは、ＰａｃｋｅｔＩｎメッセージまたはＴＣＰ（Transmission Control Protocol）パケットのＩＰヘッダに含まれる、送信元および宛先ＩＰ（Internet Protocol）データを使用することによって、どのアプリケーションノードが伝達し、どのアプリケーションノードに伝達されたかを検出し、アプリケーショングループの各々について前記連結グラフ（ＣＧ）を生成する、デバッギングシステム。
【請求項５】
請求項３に記載のデバッギングシステムであって、
前記アプリケーションシグネチャーは、前記遅延分布（ＤＤ）のピークを有し、前記デバッギングエンジンは、前記ピークのシフトまたは消失を検出する、デバッギングシステム。
【請求項６】
請求項３に記載のデバッギングシステムであって、
前記デバッギングエンジンは、前記連結グラフ（ＣＧ）におけるノードでの前記要素相互作用（ＣＩ）を、前記ノードの入力および出力エッジの各々でのフロー量を使用することによって測定する、デバッギングシステム。
【請求項７】
請求項３に記載のデバッギングシステムであって、
ロギング間隔は、等間隔なエポック間隔に分割され、
前記デバッギングエンジンは、エポック間隔の各々でのＰａｃｋｅｔＩｎメッセージを使用することによって、前記連結グラフ（ＣＧ）における隣接する２つのエッジの各々についてのフロー数を測定し、ピアソンの係数を使用することによって、時系列データ上の前記編相関（ＰＣ）を計算する、デバッギングシステム。
【請求項８】
請求項３に記載のデバッギングシステムであって、
前記デバッギングエンジンは、アプリケーションノードのアプリケーションエッジでのアプリケーションフローの統計値を測定する、デバッギングシステム。
【請求項９】
請求項８に記載のデバッギングシステムであって、
前記統計値は、単位時間当たりのフロー数およびフローサイズの少なくとも１つを含む、デバッギングシステム。
【請求項１０】
請求項８に記載のデバッギングシステムであって、
前記デバッギングエンジンは、前記統計値の最大値、最小値、および平均値を保存する、デバッギングシステム。
【請求項１１】
請求項３に記載のデバッギングシステムであって、
前記デバッギングエンジンは、前記モニターされたログを複数の時間間隔で区切り、前記複数の時間間隔について前記アプリケーションシグネチャーを計算し、
前記デバッギングエンジンは、アプリケーショングループについての前記アプリケーションシグネチャーが、前記複数の時間間隔にわたって実質的に変化していない場合、前記アプリケーションシグネチャーが安定したシグネチャーであると決定する、デバッギングシステム。
【請求項１２】
請求項２に記載のデバッギングシステムであって、

前記インフラストラクチャーシグネチャーは、物理トポロジ、インタースイッチレイテンシー、およびＯｐｅｎＦｌｏｗコントローラー特性の少なくとも１つによってモデリングされる、デバッギングシステム。

【請求項１３】

請求項１２に記載のデバッギングシステムであって、

前記物理トポロジは、物理トポロジマップを有し、

前記デバッギングエンジンは、前記ＰａｃｋｅｔＩｎメッセージおよび前記ＰａｃｋｅｔＯｕｔメッセージを基に、前記物理トポロジマップを生成する、デバッギングシステム。

【請求項１４】

請求項１２に記載のデバッギングシステムであって、

前記インタースイッチレイテンシーは、２つのネットワークスイッチ間の遅延を有し、

前記デバッギングエンジンは、前記ＰａｃｋｅｔＩｎメッセージのタイムスタンプを使用することによって、前記インタースイッチレイテンシーを計算し、

前記ＰａｃｋｅｔＩｎメッセージは、エントリーに一致しないフローに対する応答として前記２つのネットワークスイッチの各々から送信され、前記ネットワークコントローラーにて前記タイムスタンプの時刻に受信される、デバッギングシステム。

【請求項１５】

請求項１２に記載のデバッギングシステムであって、

前記ＯｐｅｎＦｌｏｗコントローラー特性は、コントローラー応答時間を有し、

前記デバッギングエンジンは、前記ＰａｃｋｅｔＩｎメッセージのタイムスタンプと前記ＰａｃｋｅｔＯｕｔメッセージのタイムスタンプとの間の差分を使用することによって、前記コントローラー応答時間を測定し、

前記ＰａｃｋｅｔＯｕｔメッセージは、前記ＰａｃｋｅｔＩｎメッセージに対する応答として前記ネットワークコントローラーから送信される、デバッギングシステム。

【請求項１６】

請求項２に記載のデバッギングシステムであって、

前記タスクシグネチャーは、タスクオートマトンを有し、該タスクオートマトンは、

タスクについての複数のトレースを収集することと、

前記複数のトレースにわたって１つ以上の共通のフローパターンを見つけることと、

前記複数のトレースにおける前記１つ以上の共通のフローパターンを保存することと、

前記１つ以上の共通のフローパターンにおける１つ以上のシーケンシャルな頻出パターンを見つけることと、

前記１つ以上のシーケンシャルな頻出パターンを長さでソートすることと、

タスクオートマトンを構築することと、

によって得られる、デバッギングシステム。

【請求項１７】

請求項２に記載のデバッギングシステムであって、

前記変化は、前記タスクシグネチャーを使用することによって、アプリケーション層およびインフラストラクチャー層における、前記未知の変化または既知の変化に分類される、デバッギングシステム。

【請求項１８】

請求項２に記載のデバッギングシステムであって、

前記相関付けは、

前記インフラストラクチャーシグネチャーの変化を計算することと、

前記アプリケーションシグネチャーの未知の変化を、前記インフラストラクチャーシグネチャーの変化に相関付けることと、

予め定義されたマッピングテーブルを参照して、前記既知の問題クラスを得ることと、

によって実行される、デバッギングシステム。

【請求項１９】

請求項２に記載のデバッギングシステムであって、
前記デバッギングエンジンは、
アプリケーションレベルのフローを下層の物理リンクのシーケンスにマッピングするリンクテーブルと、
前記アプリケーションレベルのフローをスイッチのシーケンスにマッピングするスイッチテーブルと、
アプリケーションプロセスを物理ホストにマッピングする物理マシン（ＰＭ）テーブルと、
のうちの１つ以上を使用することによって、前記未知の変化をシステム要素に相関付ける、デバッギングシステム。

【請求項２０】
ネットワーク内のデータセンターで使用されるデバッギング方法であって、
ネットワークコントローラーからトラフィック情報を収集することによって、ネットワークトラフィックをモニターすることと、
モニターされたログを使用して、アプリケーションシグネチャー、インフラストラクチャーシグネチャー、およびタスクシグネチャーをモデリングすることと、
参照ログおよび問題ログを使用して、動作状態と非動作状態との間での前記アプリケーションシグネチャーの変化を検出することと、
前記タスクシグネチャーを使用して、前記変化を検証することと、
トラブルシューティング情報を提供することと、を有し、
前記アプリケーションシグネチャーの未知の変化は、前記インフラストラクチャーシグネチャーの変化に対する依存性を考慮することによって、既知の問題クラスと相関付けられる、デバッギング方法。

【発明の詳細な説明】
【技術分野】
【０００１】
本願は、２０１２年１月２４日付で提出された「OFDiff:A Debugging Tool for OpenFlow Based Data Center」という題目の米国仮出願第６１／５９００４２号の利益を主張し、その内容を参照によって本明細書に組み込む。本願は、米国特許出願第１３／５５４６３２号および米国特許出願第１３／５５６９３０号に関連し、それら両者の内容を参照によって本明細書に組み込む。

【背景技術】
【０００２】
本発明は、ネットワークデバッギングに関連し、特に、データセンター用のネットワークデバッギングに関連する。

【０００３】
現代のデータセンターは、非常に大きく、数千台のサーバやネットワーク要素（network components）が並んでいる。それらは全て、オンデマンドリソースのプロビジョニングおよび２４×７システムの利用可能性（availability）をユーザに要求するが、複雑で大規模なインフラストラクチャーにおいては、全てのハードウェアおよびソフトウェアの障害をなくすことは非常に困難である。最近は、障害によってサービスが利用不可になることや、効率的なデバッギングの不足によってビジネスが重大な商業的損失を受けることも多くなっている。実際、大規模なデータセンターにおいては、障害およびミスコンフィグレーションに対する効率的なデバッギングを行うことは非常に困難である。アプリケーションは、アプリケーション同士で相互に作用するだけではなく、様々な形でインフラストラクチャーサービスにも作用する。さらに、アプリケーション層とインフラストラクチャー層との間には、視認性を制限する制限ライン（strict line）が存在し、それによって、データセンターのオペレーターは、いかなるデバッギング目的であっても、アプリケーションの内部を見ることが制限される。現在のところ、データセンターにおいて、オペレーターが、管理オペレーションの成功を検証する簡単な方法は存在しない。そのため、デ

ータセンターに広められる効果的なデバッギングは、コンピュータサイエンスにおいて、未だオープンな研究分野（open research area）になっている。

【０００４】

既存の商業的または専門的な解決法（非特許文献１，２，５，６）は、人々が、特定のサーバ上の事象（issues）、または、（エージェントのような）領域知識（domain knowledge）あるいは統計技術を使用してプロセスの診断を試行するという、マイクロスコピック的なアプローチを取っている。データセンターに広まっている、粗野（coarse-grained）でライトウェイト（light-weight）なモニタリングを使用するデバッギングには、課題（challenge）が残っている。先の技術は、アプリケーション依存性グラフ（application dependency graph）ごとに抽出（extracting）を行うことにフォーカスし（多くの場合、ネットワークフローの同時並列性（concurrency）または遅延特性（delay properties）を使用する）、それを診断目的で使用している。商業的解決法は、エージェントにアプリケーションプロトコルおよびアプリケーションコンフィグレーションファイルの意味的知識（semantic knowledge）をインストールすることを要求する、企業管理の解決法に依存している。成果は継続されており、モデルチェック（model checking）は分散状態（distributed states）に適用されている。さらに、人々は、リクエストをトレース（tracing）する（非特許文献３）ための計測（instrumentation）、記録の使用、分散システムロギング（非特許文献４）やネットワークトラフィック（非特許文献７）を使用する応答を試行する。現在のアプローチは、実際に配置可能なもの（deployable）からは懸け離れている。典型的には、それらの解決法は、大量のオーバーヘッドを引き起こす、多くの計測（heavy instrumentation）を要求する。また、商業的なクラウドは、計測についての追加の問題を引き起こすヘテロジーニアスである。要約すると、煩わしいモニタリング、配置（deployment）時におけるスケーラビリティな事象、ネットワークオーバーヘッド、および非効率的なデータ利用可能性は、データセンターのデバッギングにおける課題の一部となっている。

【０００５】

ＯＦＤｉｆｆは、ユニークなアングルから問題にアプローチし、コントロールプレーンでのメッセージ交換によって作られるＯｐｅｎＦｌｏｗのモニタリング能力の効果を得るものである。基本的には、ＯＦＤｉｆｆは、デバッギング目的で、ＯｐｅｎＦｌｏｗコントローラーからネットワークトラフィックをキャプチャーする。デバッギングは、動作状態および非動作状態（working and non-working states）のログを使用することによって行われる。それらを比較するため、ＯＦＤｉｆｆは、対応するロギング期間のデータセンターのアプリケーションレベルおよびインフラストラクチャーレベルの挙動（behavior）をモデリングする。オペレーショナルタスクを使用するに際しては、それらのログからキャプチャーされるアプリケーションシグネチャーの如何なる変化（例えば、連結グラフ（connectivity graph）の差分、または、アプリケーション応答時間の変化）も考慮される。また、オペレーショナルタスクは、先の既知のタスクのパターン（このパターンはＯｐｅｎＦｌｏｗトラフィックログからオフラインで学習される）に対するパターンマッチングアルゴリズムを使用するトラフィックログから識別される。アプリケーション層の変化（この変化はＯＦＤｉｆｆで検出される）は、既知のオペレーショナルタスクに起因すると考えることはできず、データセンターでの問題クラス（problem class）を識別するインフラストラクチャーレベルの変化に相関付けられる。最終的に、問題クラスを、さらなるトラブルシューティングの目的で、システム要素（system component）と相関付ける。

【先行技術文献】

【非特許文献】

【０００６】

【非特許文献１】P. Bahl, R. Chandra, A. Greenberg, S. Kandual, D. MaltZ, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in Proc. SIGCOMM' 07, Aug. 2007, pp. 13-24.

【０００７】
【非特許文献２】X. Chen, M. Zhang, Z. Morley, and M. P. Bahl, "Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions," inProc. of OSDI, 2008.
【０００８】
【非特許文献３】R. Fonseca, G. Porter, R. H. KatZ, S. Shenker, and I. Stoica, "X-Trace: A Pervasive Network Tracing Framework," in Proc. USENIX NSDI, Cambridge, MA, USA, 2007.
【０００９】
【非特許文献４】D. Geels, G. Altekar, S. Shenker, and I. Stoica, "Replay debugging for distributed applications," in Proc. Proceedings of the annual conference on USENIX' 06 Annual Technical Conference.
【００１０】
【非特許文献５】S. Kandula, R. Mahajan, P. Verkaik, S.Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks." in Proc. SIGCOMM, 2009.
【００１１】
【非特許文献６】L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft, "Macroscope: End-Point Approach to Networked Application Dependency Discovery," in Proc. ACM CoNEXT, 2009.
【００１２】
【非特許文献７】A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling Record and Replay Troubleshooting for Networks," in Proc. Proceedings of Usenix ATC 2011.
【発明の概要】
【発明が解決しようとする課題】
【００１３】
ＯＦＤｉｆｆは、ＯｐｅｎＦｌｏｗのユニークなメッセージメカニズムからの利用可能なログを使用することによって、データセンターが非常に高いデバッギング能力を備えることを提案（offer）する。
【００１４】
本発明の目的は、ネットワーク内のシステム状況認識（system situation awareness）の達成およびデータセンターのようなシステムにおけるトラブルシューティング（またはデバッギング）に貢献することにある。
【課題を解決するための手段】
【００１５】
本発明の一態様は、ネットワーク内のデータセンターで使用されるデバッギングシステムにおいて、ネットワークコントローラーからトラフィック情報を収集（collect）することによって、ネットワークトラフィックをモニターするモニタリングエンジンと、モニターされたログを使用して、アプリケーションシグネチャー（application signature）、インフラストラクチャーシグネチャー（infrastructure signature）、およびタスクシグネチャー（task signature）をモデリングするモデリングエンジンと、参照ログ（reference log）および問題ログ（problem log）を使用して、動作状態（working status）と非動作状態（non-working status）との間での前記アプリケーションシグネチャーの変化を検出し、前記タスクシグネチャーを使用して、前記変化を検証するデバッギングエンジンと、トラブルシューティング情報を提供する提供部と、を有し、前記アプリケーションシグネチャーの未知の変化は、前記インフラストラクチャーシグネチャーの変化に対する依存性（dependency）を考慮することによって、既知の問題クラス（problem class）と相関付けられる。
【００１６】
本発明の他の態様は、ネットワーク内のデータセンターで使用されるデバッギング方法

において、ネットワークコントローラーからトラフィック情報を収集（collect）することによって、ネットワークトラフィックをモニターすることと、モニターされたログを使用して、アプリケーションシグネチャー（application signature）、インフラストラクチャーシグネチャー（infrastructure signature）、およびタスクシグネチャー（task signature）をモデリングすることと、参照ログ（reference log）および問題ログ（problem log）を使用して、動作状態（working status）と非動作状態（non-working status）との間での前記アプリケーションシグネチャーの変化を検出することと、前記タスクシグネチャーを使用して、前記変化を検証することと、トラブルシューティング情報を提供することと、を有し、前記アプリケーションシグネチャーの未知の変化は、前記インフラストラクチャーシグネチャーの変化に対する依存性（dependency）を考慮することによって、既知の問題クラス（problem class）と相関付けられる。

【図面の簡単な説明】

【００１７】

【図１】図１は、システム構成を示している。

【００１８】

【図２】図２は、システム構成がマッピング（mapped）されるフローチャートを示している。

【００１９】

【図３】図３は、手順１．３における要素相互作用（ＣＩ（component interaction））を示している。

【００２０】

【図４】図４は、手順１．６のフローチャートを示している。

【００２１】

【図５】図５は、手順２のフローチャートを示している。

【００２２】

【図６】図６は、手順３のフローチャートを示している。

【００２３】

【図７】図７は、手順４のフローチャートを示している。

【００２４】

【図８】図８は、手順５．１のフローチャートを示している。

【００２５】

【図９】図９は、手順５．２のフローチャートを示している。

【発明を実施するための形態】

【００２６】

システム構成：

【００２７】

図１に示されるように、ＯＦＤｉｆｆは、ステップ１０２において、ＯｐｅｎＦｌｏｗコントローラーのＰａｃｋｅｔＩｎメッセージおよびＰａｃｋｅｔＯｕｔメッセージから、トラフィック情報を収集（collect）する。スイッチでは、その他のアクティブトラフィックモニタリングプロトコルを使用することもできる。このシステムは、モニタリング１０２、モデリング１０４、およびデバッギング１０６という３つの主要な要素を有している。上述したように、モニタリングエンジンは、ステップ１０２において、トラフィック情報を収集する。モデリングエンジンは、ステップ１０４において、モニターされたログを使用して、データセンターのアプリケーションシグネチャー（１０８）、インフラストラクチャーシグネチャー（１１０）、およびタスクシグネチャー（１１２）をモデリングする。ステップ１０６に関して、デバッギングエンジンは、最初に、ステップ１１４において、参照ログ（reference log）および問題ログ（problem log）を使用して、動作状態と非動作状態（working and non-working states）との間でのアプリケーションシグネチャーの差分をそれぞれ見つける。それから、ステップ１１６において、タスクシグネチャーを使用することによって、その変化を検証する。ステップ１１８において、未知の変

化は、インフラストラクチャーシグネチャーの変化に対する依存性（dependencies）を考慮することによって、既知の問題クラス（problem class）と相関付けられる。ステップ１２０において、クラス特有の相関性調査（class specific correlation enquiry）を使用して、問題または欠陥があるシステム要素を最終的に見つける。

【００２８】

上記のシステム構成は、図２にマッピングすることができる。ＯＦＤｉｆｆは、ＯｐｅｎＦｌｏｗコントローラー（不図示）から、参照ログ２０４および問題ログ２０６を収集する。ＯＦＤｉｆｆは、ブロック２０８において、通常の挙動（normal behavior。例えば、連結グラフ（connectivity graph）、遅延分布（delay distribution）、要素相互作用（component interaction）、編相関（partial correlation）、およびフロー統計（flow statistics））の下でのアプリケーションシグネチャーをモデリングするために、参照ログ２０４を構文解析する。また、ＯＦＤｉｆｆは、ブロック２１０において、疑わしい挙動（suspicious behavior）の下でのアプリケーションシグネチャーをモデリングするために、問題ログ２０６を構文解析する。それから、ＯＦＤｉｆｆは、ブロック２１６において、２つのログのシグネチャーを比較する。既知のオペレータータスク２１４のタスクシーケンスを使用して、ＯＦＤｉｆｆは、参照ログと問題ログとの間の変化を、既知の変化２１８（この変化は、例えば、既知のオペレータータスクに起因する）または未知の変化２２０として識別する。さらに、ＯＦＤｉｆｆは、フローから抽出されたインフラストラクチャーシグネチャー２１２を使用して、未知の変化を分析し、トラブルシューティング情報２２２をオペレーターに提供する。

【００２９】

手順１：アプリケーションシグネチャーのモデリング

【００３０】

アプリケーションシグネチャーのモデリングのゴールは、データセンター内部で起動されているアプリケーションの挙動をキャプチャーすることである。空間的、一時的、および大規模なアプリケーション（the spatial, temporal, volume dimension of applications）をカバーするために、ＯＦＤｉｆｆは、例えば、手順１．１の連結グラフ（ＣＧ：Connectivity Graph）、手順１．２の遅延分布（ＤＤ：Delay Distribution）、手順１．３の要素相互作用（ＣＩ：component interaction）、手順１．４の編相関（ＰＣ：Partial Correlation）、および手順１．５のフロー統計（ＦＳ：Flow Statistics）の観点で（in terms of）、アプリケーションシグネチャーをモデリングする。以下、各手順について詳細に述べる。

【００３１】

手順１．１：連結グラフ（ＣＧ）

【００３２】

ＰａｃｋｅｔＩｎメッセージまたはＴＣＰ（Transmission Control Protocol）パケット内のＩＰ（Internet Protocol）ヘッダに現れる（comes）、送信元および宛先ＩＰメタデータを使用して、ＯＦＤｉｆｆは、誰（どのアプリケーションノード）が誰に伝達（talking）したかを検出する。連結グラフまたはＣＧは、各アプリケーショングループについて作られる。

【００３３】

手順１．２：遅延分布（ＤＤ）

【００３４】

ＯＦＤｉｆｆは、遅延分布のピークをアプリケーションシグネチャーとして使用して、データセンターの異常（例えば、ノードの過負荷、リンクの輻輳など）に起因する、ピークのシフトまたはそれらピークの消失を検出してゴールに向かう。

【００３５】

手順１．３：要素相互作用（ＣＩ）

【００３６】

図３に示されるように、ＣＧにおいて、各ノード３０２，３０４，３０６，３０８，ま

たは３１０での要素の相互作用またはＣＩを測定するために、ＯＦＤｉｆｆは、ノード３０６の入力エッジ３１２および出力エッジ３１４の各々でのフロー量（flow volume）を使用する。その値は、そのノードへのおよびそのノードからの通信の総数（total number）で正規化（nomalized）され、ヒストグラム３１６を使用して表される。

【００３７】

手順１．４：編相関（ＰＣ）

【００３８】

遅延分布は、依存的なフローおよびそれらフローの遅延限度（delay bound）をキャプチャーするが、その依存性の強さはまだ未知である。それを測定するために、フロー量の統計を使用して、連結グラフにおける隣接エッジ間の編相関を使用する。ロギング間隔（interval）は、等間隔（equal spaced）なエポック間隔に分割される。各エポック間隔でのＰａｃｋｅｔＩｎメッセージを使用して、ＯＦＤｉｆｆは、ＣＧにおいて、各エッジについて、フロー数（flow count）を測定する。それから、時系列データ上での相関を計算するために、ピアソンの係数（Pearson's coefficient）を使用する。

【００３９】

手順１．５：フロー統計（ＦＳ）

【００４０】

コントローラーのログを使用して、アプリケーションノードと同様に、各アプリケーションエッジでのアプリケーションフローの統計値を測定する。この統計測定は、単位時間当たりのフロー数およびフローサイズ（flow size）を含む。統計値は、時間に依存するため（例えば、幾つかのアプリケーションは夜間よりも昼間の方が多く使用される）、各ケースについて、最大値、最小値、および平均値を、保持または保存する。

【００４１】

手順１．６：シグネチャーの安定性（stability）

【００４２】

しかし、幾つかのアプリケーショングループでは、シグネチャーの幾つかは安定（stable）なものではなく、例えば、アプリケーションノードが、出力フローにわたる、如何なる線形決定ロジック（any linear decision logic across its outgoing flows）も使用しない場合、要素の相互作用のシグネチャーが不安定（unstable）になり、２つの独立したフロー間の遅延分布または編相関を測定する際には同様の不安定性（instability）が見つかることになる。デバッギングフラグを立てる時に、誤判定を回避するための比較目的で使用されるアプリケーションシグネチャーには、不安定なシグネチャーは使用されない。そのため、図４のフローチャートは、各アプリケーションシグネチャーについて実行される。ステップ４０２において、アプリケーションの挙動を計算するために参照ログが与えられると、ＯＦＤｉｆｆは、安定なアプリケーションシグネチャーのみを考慮する。シグネチャーの安定性を検出するために、ＯＦＤｉｆｆは、ステップ４０４において、ログを数個の時間間隔（time intervals）に区切り、ステップ４０６において、全間隔についてのアプリケーションシグネチャーを計算する。もし、ステップ４０８において、あるアプリケーショングループについてのシグネチャーが、全パーティションにわたって、大きく（significantly）変化していない場合、ステップ４１０において、そのアプリケーショングループについてのシグネチャーは安定したシグネチャーであると考慮される。もし、ステップ４０８において、そのシグネチャーが実質的（substantially）に変化している場合、ＯＦＤｉｆｆは、ステップ４１２において、そのシグネチャーは不安定なものとして廃棄する。

【００４３】

手順２：インフラストラクチャーシグネチャーのモデリング

【００４４】

図５を参照すると、インフラストラクチャーシグネチャーのゴールは、ステップ５０２でキャプチャーされたログを使用して、ステップ５０４において、物理インフラストラクチャーに特徴付けることである。物理インフラストラクチャーは、異なる物理要素（スイ

ッチ、ルータ、および物理ホスト）を有している。物理トポロジのシグネチャーは、物理ホスト、スイッチ、およびルータの物理的な接続方法や、それらのレイテンシーおよびユーティライゼイションといったリンク特性をキャプチャーする。

【００４５】

特に、インフラストラクチャーシグネチャーをモデリングするために、手順２．１の物理トポロジ、手順２．２のインタースイッチレイテンシー（Inter-switch latency）、手順２．３のＯｐｅｎＦｌｏｗコントローラー特性（OpenFlow controller characterization）を使用することができる。

【００４６】

手順２．１：物理トポロジ

【００４７】

ＰａｃｋｅｔＩｎメッセージおよびＰａｃｋｅｔＯｕｔメッセージを基に、物理トポロジマップを作る。ＰａｃｋｅｔＩｎメッセージは、スイッチ、イングレスポート、および他のパケットフィールドの情報を含んでいる。ＰａｃｋｅｔＯｕｔメッセージは、送信に必要な出力ポートの情報を含んでいる。その後、その他のＰａｃｋｅｔＩｎメッセージが次のスイッチから受信される。それによって、スイッチの物理的接続性をまとめ（stitch）、物理トポロジを構築するのに十分な情報が与えられる。

【００４８】

手順２．２：インタースイッチレイテンシー

【００４９】

インタースイッチレイテンシーは、データセンターにおける任意の２つのスイッチ間の遅延を測定する。フローテーブルにおけるエントリーに一致しないフローがスイッチに到着した場合、スイッチはＰａｃｋｅｔＩｎメッセージをコントローラーに送信する。新たなフローについては、こうしたレポートが全てのスイッチによってパスに沿って（along）行われる。ＰＦＣ（ＯｐｅｎＦｌｏｗコントローラー）での受信メッセージのタイムスタンプを使用して、ＯＦＤｉｆｆは、インタースイッチレイテンシーを計算する。

【００５０】

手順２．３：ＯｐｅｎＦｌｏｗコントローラー特性

【００５１】

ＯｐｅｎＦｌｏｗコントローラーは、応答時間およびスループットの観点で特徴付けることができる。コントローラー応答時間を、コントローラーがＰａｃｋｅｔＩｎメッセージに応答してＰａｃｋｅｔＯｕｔメッセージを返信するのに要する時間と定義する。コントローラー応答時間は、ＰａｃｋｅｔＩｎメッセージとそれに対応するＰａｃｋｅｔＯｕｔメッセージとの間のタイムスタンプの差分を取ることで測定される。効率的なスイッチングプロトコルおよびデータセンター通信については、この時間差を非常に小さくすることが望まれている。先のシグネチャーと同様に、コントローラー応答時間（ＣＲＴ）シグネチャーとして、標準偏差と共に、応答時間の統計的な手段を使用する。

【００５２】

手順３：タスクシグネチャーのモデリング→オペレータータスクシーケンス

【００５３】

タスクシグネチャーのゴールは、各オペレーションタスク（このタスクは、例えば、デバイスに取り付けられてサーバをリブートするＶＭ（virtual machine）マイグレーションのような、データセンターにおける共通のタスクである）についてのフローの安定したシーケンスをキャプチャーし、それらをシーケンスオートマトンで表すことである。特定のタスクについてのタスクシグネチャーを計算するためのフローチャートを図６に示す。最初に、ＯＦＤｉｆｆは、ステップ６０２において、タスクについての複数のトレースＴを収集し、ステップ６０４において、複数のトレースにわたって共通のフローパターンを見つける。次に、ＯＦＤｉｆｆは、ステップ６０６において、複数のトレースＴにおける共通のフローのみを保持または保存する。ここで、Ｔ≧Ｔ’で、Ｔ’は共通フローを意味している。さらに、ＯＦＤｉｆｆは、ステップ６０８において、Ｔ’におけるシーケンシ

ャルな頻出パターン（sequential frequent patterns）を見つけ、ステップ６１０において、それらを長さの観点でソートする。同じ長さの場合は、ＯＦＤｉｆｆは、ステップ６１０において、サポート値または最小サポート（min#sup）値の観点でそれらをソートする。それから、ＯＦＤｉｆｆは、ステップ６１２において、タスクオートマトンまたはタスクシグネチャーを構築する。

【００５４】

オペレータータスクシーケンスは、デバッギングの精度を高めるが、デバッギング方法には必須（compulsory）ではない点に注意すべきである。また、オペレータータスクシーケンスは、オペレーターまたはオペレーターのエージェントによって初期化されたネットワークオペレーションを検証するために、独立して使用することができる。

【００５５】

手順４：検証変化（Validating Changes）

【００５６】

図７を参照すると、先のセクションでモデルとの比較によって識別される変化は、アプリケーション層およびインフラストラクチャー層の双方において、既知の変化（７１４）および未知の変化（７１２）かどうかがチェックされる。既知の変化７１４は、意図的に（intentionally）終了した変化であり、システムにおける正当な（valid）オペレーションタスクによって説明することができる。既知の変化（７１４）と未知の変化（７１２）とを区別するために、タスクシグネチャーを使用する。タスクシグネチャーは、ステップ７０８において、与えられたログからタスクシーケンスを作る。ステップ７０６において、アプリケーションの変化が検出された場合、ＯＦＤｉｆｆは、ステップ７１０において、タスクの時系列をチェックして、その変化が起きた同時刻付近の関連するオペレーショナルタスクを見つける。

【００５７】

ここで、図７のｎは、未知の変化に関連するアプリケーションノードのリストであり、図７のｅは、未知の変化に関連するアプリケーションフローのリストである。

【００５８】

手順５：デバッギング

【００５９】

デバッギングは２つの手順を有する。手順５．１は、データセンターで発生し得る（possible）問題クラスを見つけるという、問題クラスへの相関付けを含む。手順５．２は、データセンターで問題を生じさせる、問題または欠陥があるデータセンターの要素を見つけるという、システムコンテキストへの相関付けを含む。

【００６０】

手順５．１：問題クラスへの相関付け

【００６１】

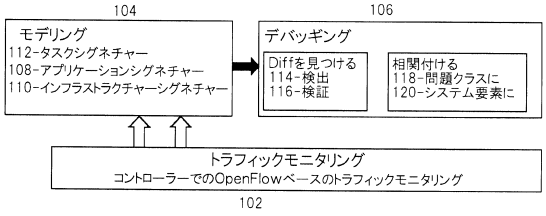
図８を参照すると、ステップ８０２において、与えられたログについて、アプリケーションシグネチャーの変化を検出すると、ステップ８０４において、インフラストラクチャーシグネチャーにおける関連する変化を計算し、ステップ８０６において、アプリケーションの変化とインフラストラクチャーの変化との依存性を相関付ける。その相関付けの出力を使用して、ステップ８０８において、現在のデータセンターの問題を、既知のクラスの１つに分類する。

【００６２】

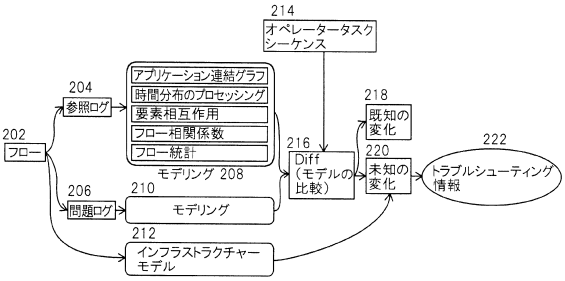
この比較の前に、初期的に、相関マッピングテーブル（ＭＴ）を作る。ＭＴは、ホスト障害、ホストパフォーマンス問題、ネットワーク切断などの異なる問題クラスに変化した、アプリケーションシグネチャーおよびそれに関連するインフラストラクチャーシグネチャーのマッピングを含む。例えば、幾つかのアプリケーションの遅延分布（ＤＤ）、編相関（ＰＣ）値、およびフロー統計（ＦＳ）が、関連するインタースイッチレイテンシーと共に変化した場合、ネットワークボトルネック問題クラス（network bottleneck problem class）の可能性がある。

【００６３】
手順５．２：システム要素への相関付け
【００６４】
図９を参照すると、ステップ９０８で問題クラス９０２から欠陥要素を検出するために、ＯＦＤｉｆｆは、最初に、ステップ９０４において、インフラストラクチャーシグネチャーを使用して、リンクテーブル、スイッチテーブル、およびＰＭ（physical machine）テーブルという３つのテーブルを生成する。リンクテーブルは、アプリケーションレベルのフローを、そのフローが妨害する下層物理リンクのシーケンス（the sequence of underlying physical links that the flows traverse）にマッピングする。スイッチテーブルは、アプリケーションレベルのフローを、そのフローが妨害するスイッチのシーケンス（the sequence of switches the flows traverse）にマッピングし、最後に、ＰＭテーブルは、アプリケーションプロセスを物理ホストにマッピングする。問題クラス９０２に依存して、ステップ９０４において、特定のテーブルが選択され、ステップ９０６において、その変化が、データセンターにおける特定の要素に相関付けられる。
【００６５】
手順５．１および５．２はデバッギング技術を上げる（scale）。
【００６６】
ＯＦＤｉｆｆは、複数の新規なシグネチャーモデリング技術の組み合わせを使用して、システム状況認識の達成およびトラブルシューティングに貢献する。それらのゴールは、ＯｐｅｎＦｌｏｗ技術のユニークなライトウェイトセンシング（light-weight sensing）メカニズムを利用することによって達成されるが、提案技術の多くは、一般的であり、他のシステムにも適用可能である。特に、手順３のタスクシグネチャーのモデリングは、ネットワークフローを使用して、共通のデータセンターのタスクの特徴を作るためのメカニズムベースの教師あり学習（supervised learning）であり、それらタスクをその後に検証するために、またデバッギングにおけるＯＦＤｉｆｆの一部として、独立して使用することができる。
【００６７】
上記は、全ての点で具体的および典型的なものであって限定的なものではないと理解されるべきであり、ここで開示された発明の範囲は、「発明の詳細な説明」からではなく、むしろ特許法によって認められた十分な範囲によって解釈されるような請求項から判断されるべきである。ここで示され記載された実施形態は、本発明の原理の例示にすぎず、発明の範囲および精神から逸脱することなく、当業者が種々の変更を実行してもよいことを理解されるべきである。当業者は、本発明の範囲および精神から逸脱することなく、種々の他の特徴の組み合わせを実現することができる。

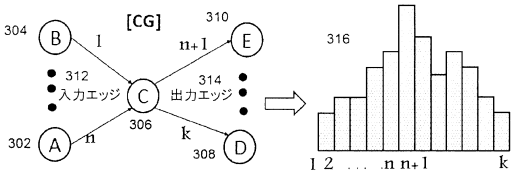
【図１】



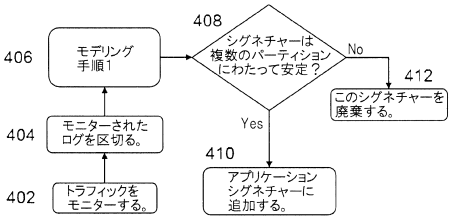
【図２】



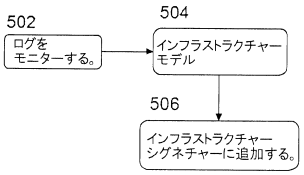
【図３】



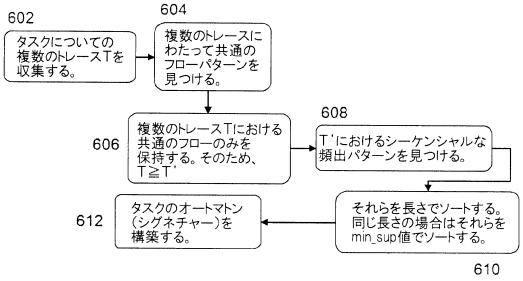
【図４】



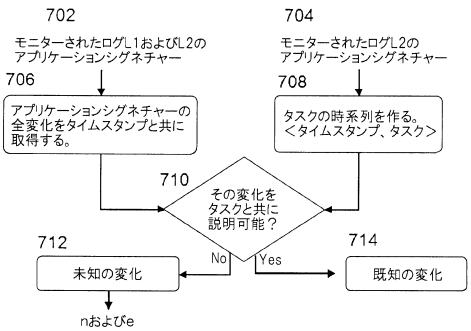
【図５】



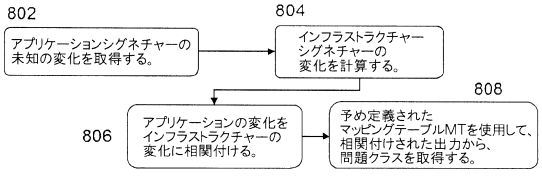
【図６】



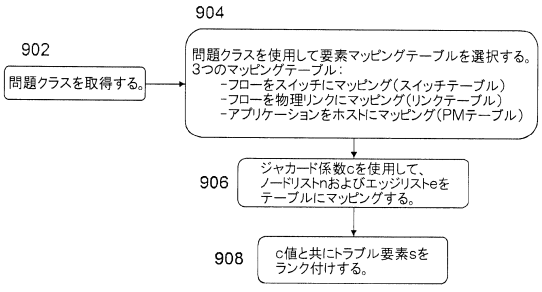
【図７】



【図８】



【図９】



フロントページの続き

(72)発明者 ルメザヌ、 クリスティアン
アメリカ合衆国 ０８５２０ ニュージャージー州 イースト ウインザ ロック ラン ロード ２５
(72)発明者 ジアング、 グオフェイ
アメリカ合衆国 ０８５４０ ニュージャージー州 プリンストン ダンビー コート ５
(72)発明者 ジャング、 ユエピング
アメリカ合衆国 ０８５４０ ニュージャージー州 プリンストン ハーヴァード サークル ９５
(72)発明者 シング、 ヴィシャル
アメリカ合衆国 ０８５４０ ニュージャージー州 プリンストン ブルー スプリング ロード ６０２
(72)発明者 アルフィン、 アフサン
アメリカ合衆国 ６１８２０ イリノイ州 シャンペイン エス. ファースト ストリート ５０８ アパートメント ４０２

審査官 安藤 一道

(56)参考文献 特開２００８－０２２４９８（ＪＰ，Ａ）
国際公開第２０１１／１１８５７４（WO，Ａ１）
特開２００５－３４６３３１（ＪＰ，Ａ）
米国特許出願公開第２０１３／０１８５４１９（ＵＳ，Ａ１）
米国特許出願公開第２００６／００２９０１６（ＵＳ，Ａ１）
特開２００９－１６９６０９（ＪＰ，Ａ）
米国特許第７８９４３５７（ＵＳ，Ｂ２）
米国特許第７９１６６５２（ＵＳ，Ｂ１）
米国特許第７７６９８５１（ＵＳ，Ｂ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｈ０４Ｌ １２／７０
Ｇ０６Ｆ １１／３４
Ｈ０４Ｌ １２／２４
Ｈ０４Ｌ １２／７１７