(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 August 2013 (01.08.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/112288 A1

(51) International Patent Classification:
G06F 11/36 (2006.01) *G06F 9/44* (2006.01)

(21) International Application Number:
PCT/US2013/020795

(22) International Filing Date:
9 January 2013 (09.01.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/590,042 24 January 2012 (24.01.2012) US
13/736,158 8 January 2013 (08.01.2013) US

(71) Applicant: **NEC LABORATORIES AMERICA, INC.** [US/US]; 4 Independence Way, Suite 200, Princeton, New Jersey 08540 (US).

(72) Inventors: **LUMEZANU, Cristian**; 25 Rock Run Road, East Windsor, New Jersey 08520 (US). **JIANG, Guofei**; 5 Danby Court, Princeton, New Jersey 08540 (US). **ZHANG, Yueping**; 95 Harvard Circle, Princeton, New Jersey 08540 (US). **SINGH, Vishal**; 602 Blue Spring Road, Princeton, New Jersey 08540 (US). **AREFIN, Ahsan**; 508 S. First Street, Apt. 402, Champaign, Illinois 61820 (US).

(74) Agent: **KOLODKA, Joseph**; NEC Laboratories America, Inc., 4 Independence Way, Suite 200, Princeton, New Jersey 08540 (US).

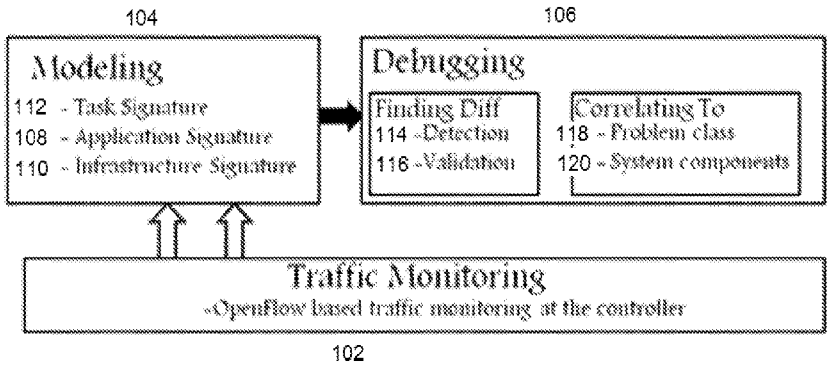
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) Title: NETWORK DEBUGGING



(57) Abstract: A debugging system used for a data center in a network is disclosed. The system includes a monitoring engine to monitor network traffic by collecting traffic information from a network controller, a modeling engine to model an application signature, an infrastructure signature, and a task signature using a monitored log, a debugging engine to detect a change in the application signature between a working status and a non-working status using a reference log and a problem log, and to validate the change using the task signature, and a providing unit to provide toubleshooting information, wherein an unknown change in the application signature is correlated to a known problem class by considering a dependency to a change in the infrastructure signature. Other methods and systems also are disclosed.

WO 2013/112288 A1

NETWORK DEBUGGING

[0001] This application claims the benefit of U.S. Provisional Application No. 61/590,042, entitled, "OFDiff: A Debugging Tool for OpenFlow Based Data Center," filed on January 24, 2012, the contents of which are incorporated herein by reference. This application is related to U.S. Patent Application No. 13/554,632 and U.S. Patent Application No. 13/556,930, the contents of both of which are incorporated herein by reference.

BACKGROUND OF THE INVENTION

[0002] The present invention relates to network debugging and, more particularly, to network debugging for a data center.

[0003] Modern data centers are very large in the order of thousands of servers and network components. They all claim on-demand resource provisioning and 24x7 system availability to the users though it is very hard to eradicate all hardware and software failures in such complex and large scale infrastructures. There have been many incidents recently where failures has lead to unavailability of services and the lack of efficient debugging has caused heavy commercial losses to the businesses. The fact is, efficient debugging of failures and misconfiguration in such large scale data centers is very difficult. Applications not only interact with each other but also interact with infrastructure services in diverse ways. Moreover, there is a strict line of visibility restriction between the application layer and the infrastructure layer, which limits the data center operators to look inside the applications for any debugging purposes. There is currently also no easy way for the operators to verify the success of management operations in the data center. Therefore, efficient data center wide debugging is still an open research area in computer science.

[0004] Existing commercial or academic [1, 2, 5, 6] solutions have taken a microscopic approach, where people try to diagnose issues on specific servers or processes using domain knowledge (like agents) or statistical techniques. Data center wide debugging using coarse-grained and light-weight monitoring remains a challenge. The previous techniques are focused on extracting per application dependency graph (in most cases, using network flow concurrency or delay properties) and use it for diagnosis purposes. Commercial solutions have been relying

on enterprise management solutions, which require agents to be installed with semantic knowledge of application protocols and applications configuration files. Efforts have been ongoing to apply model checking to distributed states. Furthermore, people have tried instrumentation for tracing [3] requests and use of record and replay using distributed system logging [4] and using network traffic [7]. The current approaches are far from practically deployable. Typically, the solutions require heavy instrumentation resulting in a lot of overhead. Also the commercial cloud is heterogeneous, which poses additional problems for instrumentation. To sum up, intrusive monitoring, scalability issues in deployment, network overhead and insufficient data availability are some of the challenges in data center debugging.

[0005] OFDiff approaches the problem from a unique angle and takes advantage of OpenFlow's monitoring capabilities built upon message exchange in its control plane. Basically OFDiff captures network traffic from the OpenFlow controller for debugging purpose. The debugging is done by using logs of working and non-working states. To compare them, OFDiff models application and infrastructure level behavior of the data center of the corresponding logging period. Any changes in the application signatures (e.g., a different connectivity graph, or change in application response time) captured from those logs are considered to explain using operational tasks. The operational tasks are also identified from the traffic logs using a pattern matching algorithm to the previously known tasks' patterns (learned offline also from OpenFlow traffic logs). Application layer changes (detected by OFDiff), which cannot be attributed to well known operational tasks are further correlated to the infrastructure level changes to identify problem class in the data center. Finally we correlate the problem class to the system components for further troubleshooting purposes.

[0006] [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] [2] X. Chen, M. Zhang, Z. Morley, and M. P. Bahl,"Automating Network Application Dependency Discovery:Experiences, Limitations, and New Solutions," in Proc. of OSDI, 2008.

[0008] [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] [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] [5] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks." in Proc. SIGCOMM, 2009.

[0011] [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] [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.

BRIEF SUMMARY OF THE INVENTION

[0013] OFDiff offers very powerful data center debugging capabilities by using available logs from the OpenFlow's unique messaging mechanism.

[0014] An objective of the present invention is to contribute to achieving system situation awareness in a network and to troubleshooting (or debugging) for a system such as a data center.

[0015] An aspect of the present invention includes, in a debugging system used for a data center in a network, a monitoring engine to monitor network traffic by collecting traffic information from a network controller, a modeling engine to model an application signature, an infrastructure signature, and a task signature using a monitored log, and a debugging engine to detect a change in the application signature between a working status and a non-working status using a reference log and a problem log, and to validate the change using the task signature, and a providing unit to provide toubleshooting information, wherein an unknown change in the application signature is correlated to a known problem class by considering a dependency to a change in the infrastructure signature.

[0016] Another aspect of the present invention includes, in a debugging method used for a data center in a network, monitoring network traffic by collecting traffic information from a network controller, modeling an application signature, an infrastructure signature, and a task signature using a monitored log, detecting a change in the application signature between a working status and a non-working status using a reference log and a problem log, validating the change using

the task signature, and providing toubleshooting information, wherein an unknown change in the application signature is correlated to a known problem class by considering a dependency to a change in the infrastructure signature.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] FIG. 1 depicts system architecture.

[0018] FIG. 2 depicts a flow chart which system architecture is mapped to.

[0019] FIG. 3 depicts component interaction (CI) in Procedure 1.3.

[0020] FIG. 4 depicts a flow chart for Procedure 1.6.

[0021] FIG. 5 depicts a flow chart for Procedure 2.

[0022] FIG. 6 depicts a flow chart for Procedure 3.

[0023] FIG. 7 depicts a flow chart for Procedure 4.

[0024] FIG. 8 depicts a flow chart for Procedure 5.1.

[0025] FIG. 9 depicts a flow chart for Procedure 5.2.

DETAILED DESCRIPTION

[0026] System architecture:

[0027] As illustrated in Fig. 1, OFDiff collects traffic information from the OpenFlow controller PacketIn and PacketOut messages in step 102. Other active traffic monitoring protocol at the switches can also be used. The system has three major components: Monitoring 102, Modeling 104 and Debugging 106. As explained before, the monitoring engine collects the traffic information in step 102. The modeling engine models data center application (108), infrastructure (110) and task (112) signatures using monitored logs in step 104. Regarding step 106, the debugging engine first finds the differences in application signatures between the working and non-working states using the reference and problem logs, respectively, in step 114. Then it validates the changes by using task signatures in step 116. The unknown changes are correlated to the known problem classes considering their dependencies to the changes in infrastructure signatures in step 118. We use class specific correlation enquiry to finally find the problematic or faulty system components in step 120.

[0028] The above system architecture can be mapped to Fig. 2. OFDiff collects reference log 204 and problem log 206 from the OpenFlow controller (not shown). It parses reference log 204 to model application signatures under normal behavior (for example, connectivity graph, delay distribution, component interaction, partial correlation, and flow statistics) in block 208. OFDiff also parses problem log 206 to model application signatures under suspicious behavior in block 210. It then compares the signatures in the two logs in block 216. Using task sequence of known operator tasks 214, it identifies the changes between the reference and problem logs as either known changes 218 (e.g., due to a known operator task) or unknown changes 220. Further, OFDiff analyzes the unknown changes using infrastructure signatures 212 extracted from the flows and provides troubleshooting information 222 to operator.

[0029] Procedure 1: Modeling Application Signatures

[0030] The goal of modeling application signatures is to capture the application behavior running inside the data center. To cover the spatial, temporal and volume dimension of applications, OFDiff models application signatures, for example, in terms of Procedure 1.1 Connectivity Graph (CG), Procedure 1.2 Delay Distribution (DD), Procedure 1.3 Component Interaction (CI), Procedure 1.4 Partial Correlation (PC) and Procedure 1.5 Flow Statistics (FS). We next elaborate each procedure.

[0031] Procedure 1.1: Connectivity Graph (CG)

[0032] Using source and destination IP (Internet Protocol) metadata that comes in the PacketIn message or the IP header in TCP (Transmission Control Protocol) packets, OFDiff detects who (which application node) is talking to whom. A connectivity graph or CG is built out of it for each application group.

[0033] Procedure 1.2: Delay Distribution (DD)

[0034] OFDiff uses peaks of the delay distribution as the application signature with the goal to detect the shift in peaks or their disappearance due to the anomalies in the data center (such as node overload, link congestion, etc.).

[0035] Procedure 1.3: Component Interaction (CI)

[0036] As shown in Fig. 3, to measure component interaction or CI at each node 302, 304, 306, 308, or 310 in CG, OFDiff uses flow volumes at each of input and output edges 312 and 314 of

node 306. The value is normalized with the total number of communication to and from the node and represented using histogram 316.

[0037] Procedure 1.4: Partial Correlation (PC)

[0038] Though the delay distribution captures the dependent flows and their delay bound, the strength of the dependency is still unknown. To measure this, we use the partial correlation between adjacent edges in the Connectivity Graph using flow volume statistics. The logging interval is divided into equal spaced epoch intervals. Using the PacketIn message at each epoch interval OFDiff measures the flow count for each edge in the CG. We then use Pearson's coefficient for computing correlation over these time-series data.

[0039] Procedure 1.5: Flow Statistics (FS)

[0040] Using the controller log, we measure statistical values of application flows at each application edge as well as application node. The statistical measurement includes flow counts per unit time and flow size. As the statistical value can be biased depending on time (e.g., some applications are used heavily during the day time compared to the night time), we keep or save max, min and average values for each case.

[0041] Procedure 1.6: Stability of Signatures

[0042] However, in some application groups, some of the signatures may not be stable e.g., if an application node does not use any linear decision logic across its outgoing flows, the component interaction signature becomes unstable, similar instability is found when we measure the delay distribution or partial correlation between two independent flows. Unstable signatures may not be used in the application signatures for the comparison purpose to avoid the false positive in raising the debugging flag. Therefore, the flow chart in Fig. 4 is executed for each application signature. Given a reference log in step 402 to compute the application behavior, OFDiff only considers the stable application signatures. To detect signatures stability, OFDiff partitions the log into several time intervals in step 404 and computes the application signatures for all intervals in step 406. If for an application group, a signature does not change significantly across all partitions in step 408, it is considered as a stable signature for that application group in step 410. If the signature changes substantially in step 408, OFDiff discards the signatures as being unstable in step 412.

[0043] Procedure 2: Modeling Infrastructure Signatures

[0044] In reference to Fig. 5, the goal of infrastructure signature is to characterize the physical infrastructure in step 504 using the captured log in step 502. The physical infrastructure comprises different physical components (switches, routers and physical hosts). The signature of physical topology captures how physical hosts, switches and routers are physically connected and their link characteristics such as latency and utilization.

[0045] In particular, we can use Procedure2.1 Physical topology, Procedure 2.2 Inter-switch latency, and Procedure 2.3 OpenFlow controller characterization to model the infrastructure signature.

[0046] Procedure 2.1: Physical topology

[0047] We build a physical topology map based on PacketIn and PacketOut messages. The PacketIn message contains information about the switch, the ingress port and other packet fields. The PacketOut contains information about the output port it needs to be sent. Subsequently, another PacketIn is received from the next switch. This gives enough information to stitch the physical connectivity of switches and construct the physical topology.

[0048] Procedure 2.2: Inter-switch latency

[0049] The Inter-switch latency measures the delay between any two switches in the data center. When a flow arrives at a switch that has no matching entry in the flow table, it sends a PacketIn message to the controller. For a new flow, such reporting is done by all the switches along the path. Using the timestamp of the receiving message at the PFC (OpenFlow controller), OFDiff computes the inter-switch latency.

[0050] Procedure 2.3: OpenFlow controller characterization

[0051] OpenFlow controller can be characterized in terms of its response time and throughput. Controller Response Time defines the time the controller takes to reply a PacketOut message in response to a PacketIn. The controller response time is measured by taking the difference in timestamp between a PacketIn message and its corresponding PacketOut message. For an efficient switching protocol and data center communication, it is desirable that this time difference be fairly small. Similar to the previous signature, we use a statistical mean of the response time along with the standard deviation as the Controller Response (CRT) Time signature.

[0052] Procedure 3: Modeling Tasks Signatures –> Operator Task Sequence

[0053] The goal of the task signature is to capture the stable sequence of flows for each operational task (e.g., common tasks in data centers like VM (virtual machine) migration, attaching a device, rebooting a server) and to represent them by sequence automata. A flow chart for computing a task signature for a specific task is given in Fig. 6. First, OFDiff collects multiple traces T for a task in step 602 and finds common flow patterns across the traces in step 604. Next, OFDiff keeps or stores only the common flows in the traces T in step 606, meaning that $T \geq T'$, where T' denotes the common flows. OFDiff further finds sequential frequent patterns in T' in step 608 and sorts them in terms of length in step 610. For the same length, OFDiff sorts them in terms of the support value or the minimum support (min_sup) value in step 610. Then, OFDiff constructs task automata or task signatures in step 612.

[0054] It should be noted that the operator task sequence increases the accuracy of debugging, but is not compulsory for debugging method. Also, it could be used stand alone to validate network operations initiated by the operator or operator's agent.

[0055] Procedure 4: Validating Changes

[0056] Referring to Fig. 7, changes identified by comparing the models in the previous sections are checked for known (714) and unknown (712) changes in both application and infrastructure layers. Known changes 714 are the changes that are done intentionally and can be explained by valid operational tasks in the systems. To differentiate between known (714) and unknown (712) changes, we use task signature. The task signature builds a task sequence from the given log in step 708. When an application change is detected in step 706, OFDiff checks the task-time series to find related operational tasks around the same time when the change happened in step 710.

[0057] Here, **n** in Fig. 7 is the list of application nodes related to the unknown changes, **e** in Fig. 7 is list of application flows related to the unknown changes.

[0058] Procedure 5: Debugging

[0059] The debugging comprises two procedures: Procedure 5.1 includes correlating to problem class, which finds the possible problem class in the data center. Procedure 5.2 including correlating to system contexts, which finds the problematic or faulty data center components causing the problem in the data center.

[0060] Procedure 5.1: Correlating to problem class

[0061] Referring to Fig. 8, once we detect a change in the application signatures for the given logs in step 802, we compute related changes in their infrastructure signatures in step 804 and correlate the dependency of application and infrastructure changes in step 806. Using the correlation output, we classify the current data center problems to one of these known classes in step 808.

[0062] Before this comparison, we initially build a correlation mapping table (MT) that includes the mapping of the application and related infrastructure signature changes to different problem classes such as host failure, host performance problem, network disconnectivity and so on. For example, if the delay distribution (DD), partial correlation (PC) values and flow statistics (FS) of some applications change along with the related inter-switch latency, there is a possibility of the network bottleneck problem class.

[0063] Procedure 5.2: Correlating to system components

[0064] Referring to Fig. 9, to detect the faulty component in step 908 from problem class 902, OFDiff first generates three tables using infrastructure signature: a link table, a switch table and a PM (physical machine) table in step 904. The link table maps the application level flows to the sequence of underlying physical links that the flows traverse. The switch table maps the application level flows to the sequence of switches the flows traverse and finally the PM table maps the application processes to the physical hosts. Depending on problem class 902, specific tables are selected in step 904 to correlate the changes to specific components in the data center in step 906.

[0065] Procedures 5.1 and 5.2 scale the debugging techniques.

[0066] OFDiff contributes to achieving system situation awareness and troubleshooting using a combination of multiple novel signature modeling techniques. Although, these goals are achieved by leveraging the unique light-weight sensing mechanism of OpenFlow technologies, many of the proposed techniques are general and can be applied to other systems. In particular, Procedure 3 modeling task signature is a supervised learning based mechanism to create fingerprints of common data center tasks using network flows, and can be used standalone to validate these tasks at later time and also as a part of OFDiff in debugging.

[0067] The foregoing is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the invention disclosed herein is not to be determined from

the Detailed Description, but rather from the claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that the embodiments shown and described herein are only illustrative of the principles of the present invention and that those skilled in the art may implement various modifications without departing from the scope and spirit of the invention. Those skilled in the art could implement various other feature combinations without departing from the scope and spirit of the invention.

What is claimed is:

1. A debugging system used for a data center in a network, the system comprising:

a monitoring engine to monitor network traffic by collecting traffic information from a network controller;

a modeling engine to model an application signature, an infrastructure signature, and a task signature using a monitored log;

a debugging engine to detect a change in the application signature between a working status and a non-working status using a reference log and a problem log, and to validate the change using the task signature; and

a providing unit to provide toubleshooting information,

wherein an unknown change in the application signature is correlated to a known problem class by considering a dependency to a change in the infrastructure signature.

2. The debugging system as in claim 1,

wherein the network comprises an OpenFlow network and the traffic information is collected through at least one of PacketIn and PacketOut messages.

3. The debugging system as in claim 2,

wherein the application signature is modeled by at least one of connectivity graph (CG), delay distribution (DD), component interaction (CI), partial correlation (PC), and flow statistics (FS).

4. The debugging system as in claim 3,

wherein the debugging engine detects which application node talks and which application node is talked to by using source and destination IP (Internet Protocol) data included in the PacketIn message or in an IP header in a TCP (Transmission Control Protocol) packet, and generates the connectivity graph (CG) for each application group.

5. The debugging system as in claim 3,

wherein the application signature comprises a peak of the delay distribution (DD), and

wherein the debugging engine detects a shift in the peak or disappearance.

6. The debugging system as in claim 3,

wherein the debugging engine measures the component interaction (CI) at a node in the connectivity graph (CG) by using a flow volume at each of input and output edges of the node.

7. The debugging system as in claim 3,

wherein a logging interval is divided into epoch intervals that are equal spaced, and

wherein the debugging engine measures a flow count for each of two adjacent edges in the connectivity graph (CG) by using a PacketIn message at each epoch interval, and computes the partial correlation (PC) over time-series data by using Pearson's coefficient

8. The debugging system as in claim 3,

wherein the debugging engine measures a statistical value of an application flow at an application edge of an application node,

9. The debugging system as in claim 8,

wherein the statistical value includes at least one of a flow count per unit time and a flow size.

10. The debugging system as in claim 8,

wherein the debugging engine stores max, min, and average values of the statistical value.

11. The debugging system as in claim 3,

wherein the debugging engine partitions the monitored log into a plurality of time intervals, and computes the application signature for the plurality of time intervals, and

wherein the debugging engine determines that the application signature is a stable signature if the application signature for an application group does not change substantially across the plurality of time intervals.

12. The debugging system as in claim 2,

wherein the infrastructure signature is modeled by at least one of physical topology, inter-switch latency, and OpenFlow controller characterization.

13. The debugging system as in claim 12,

wherein the physical topology comprises a physical topology map, and

wherein the debugging engine generates the physical topology map based on the PacketIn and PacketOut messages.

14. The debugging system as in claim 12,

wherein the inter-switch latency comprises a delay between two network switches,

wherein the debugging engine computes the inter-switch latency by using a timestamp of the PacketIn message, and

wherein the PacketIn message is transmitted from each of the two network switch in response to a flow that has no matching entry and received at the network controller at the timestamp.

15. The debugging system as in claim 12,

wherein OpenFlow controller characterization comprises a controller response time, and

wherein the debugging engine measures the controller response time by using a difference between timestamps of the PacketIn message and the PacketOut message, and

wherein the PacketOut message is transmitted from the network controller in response to the PacketIn message.

16. The debugging system as in claim 2,

wherein the task signature comprises a task automaton obtained by:

collecting multiple traces for a task;

finding one or more common flow patterns across the multiple traces;

storing said one or more common flow patterns in the multiple traces;

finding one or more sequential frequent patterns in said one or more common flow patterns;

sorting said one or more sequential frequent patterns in length; and

constructing a task automaton.

17. The debugging system as in claim 2,

wherein the change is categorized into the unknown change or a known change in an application layer and an infrastructure layer by using the task signature.

18. The debugging system as in claim 2,

wherein the correlation is performed by:

computing a change in the infrastructure signature;

correlating the unknown change in the application signature to the change in the infrastructure signature; and

obtaining the known problem class with reference to a predefined mapping table.

19. The debugging system as in claim 2,

wherein the debugging engine correlates the unknown change to a system component by using one or more of:

a link table to map an application level flow to a sequence of underlying physical links,

a switch table to map the application level flows to a sequence of switches, and

a physical machine (PM) table to map an application process to a physical host.

20. A debugging method used for a data center in a network, the method comprising:

monitoring network traffic by collecting traffic information from a network controller;

modeling an application signature, an infrastructure signature, and a task signature using a monitored log;

detecting a change in the application signature between a working status and a non-working status using a reference log and a problem log;

validating the change using the task signature; and

providing toubleshooting information,

wherein an unknown change in the application signature is correlated to a known problem class by considering a dependency to a change in the infrastructure signature.

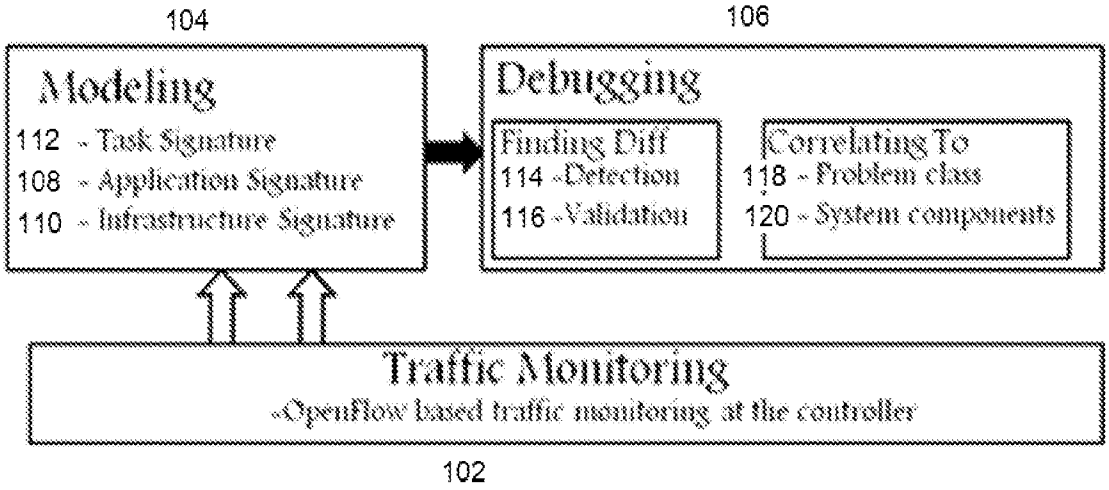

104
106
Modeling
112 - Task Signature
108 - Application Signature
110 - Infrastructure Signature
Debugging
Finding Diff
114 -Detection
116 -Validation
Correlating To
118 - Problem class
120 - System components
Traffic Monitoring
-Openflow based traffic monitoring at the controller
102

FIG. 1

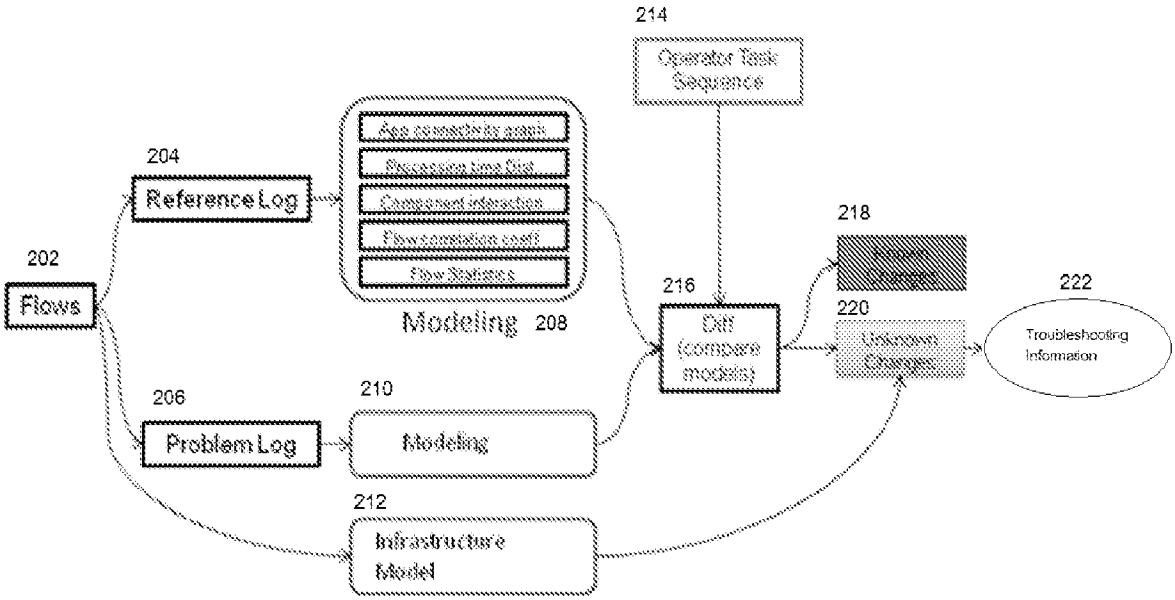

214
Operator Task Sequence
204
Reference Log
202
Flows
Modeling 208
218
216
Diff (compare models)
220
222
Troubleshooting Information
206
Problem Log
210
Modeling
212
Infrastructure Model

FIG. 2

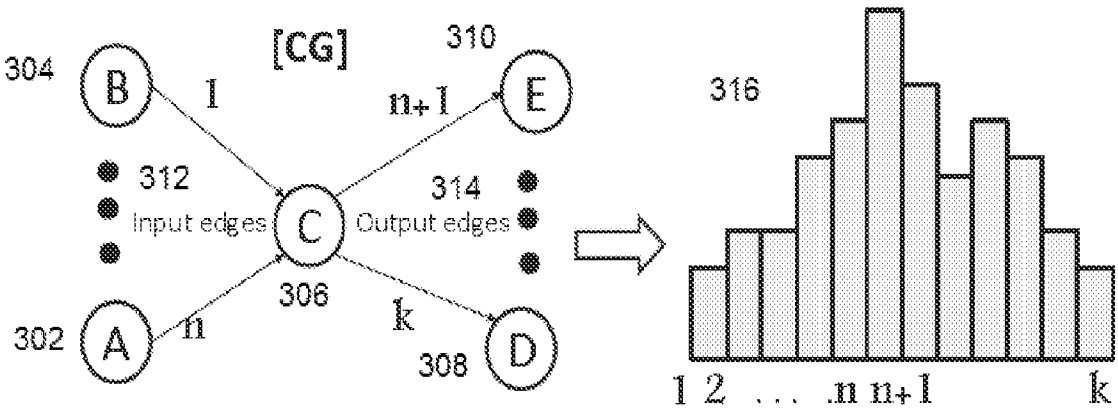

[CG]
304
B
1
310
n+1
E
312
Input edges
C
314
Output edges
306
302
A
n
k
308
D
316
1 2 . . . n n+1 k

FIG. 3

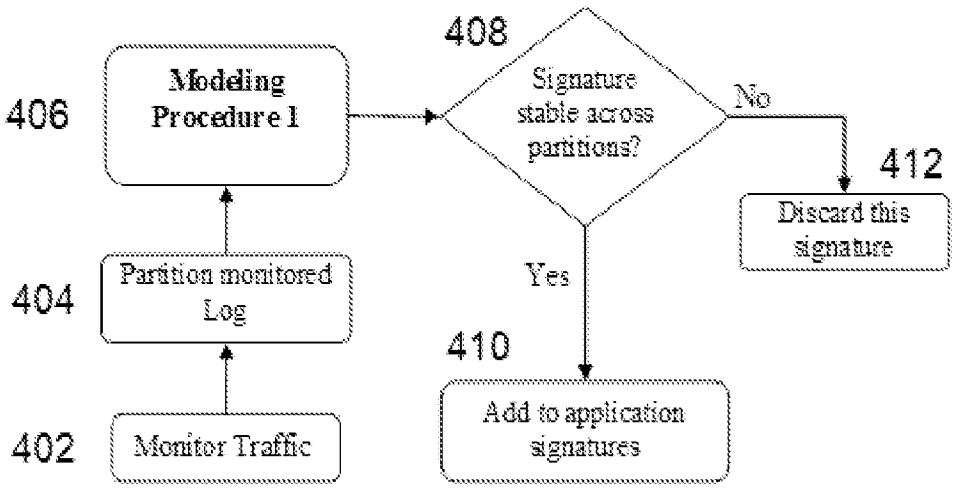

408
406
Modeling Procedure 1
Signature stable across partitions?
No
412
Discard this signature
Yes
404
Partition monitored Log
410
Add to application signatures
402
Monitor Traffic

FIG. 4

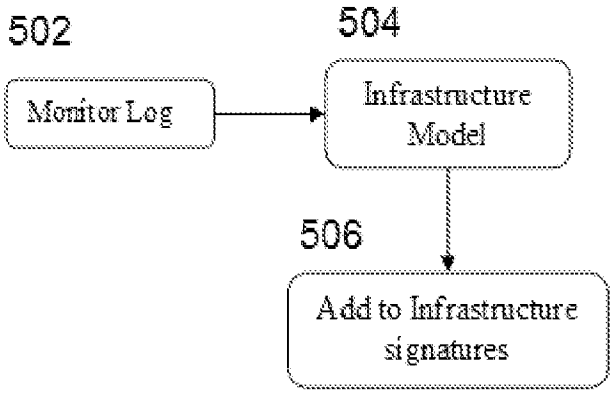

502
Monitor Log
504
Infrastructure Model
506
Add to Infrastructure signatures

FIG. 5

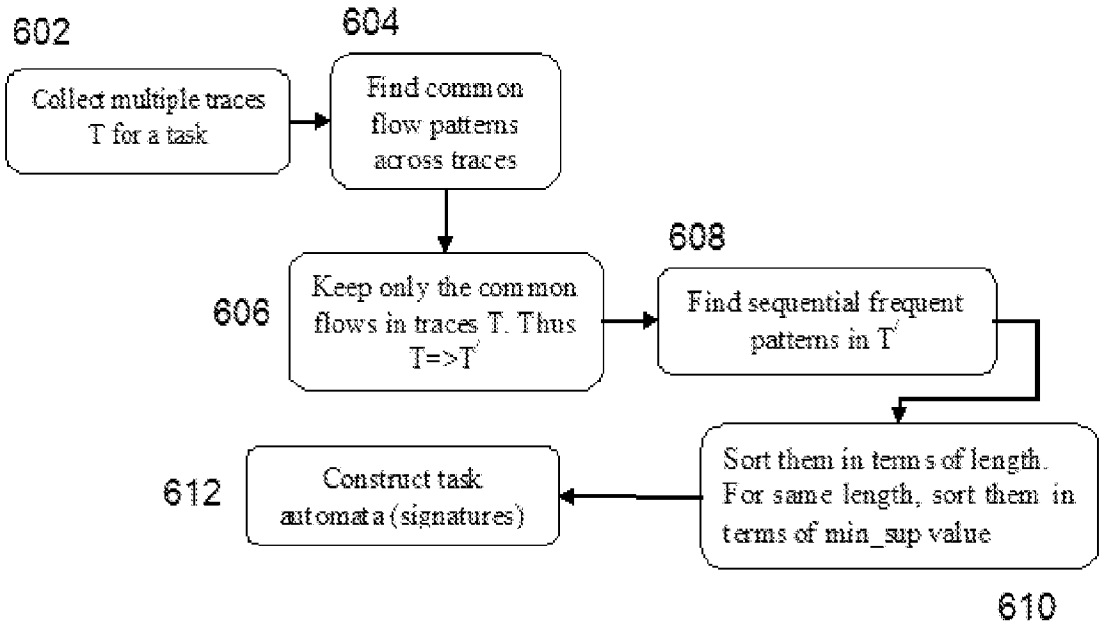

602
Collect multiple traces T for a task
604
Find common flow patterns across traces
606
Keep only the common flows in traces T. Thus T=>T'
608
Find sequential frequent patterns in T'
610
Sort them in terms of length. For same length, sort them in terms of min_sup value
612
Construct task automata (signatures)

FIG. 6

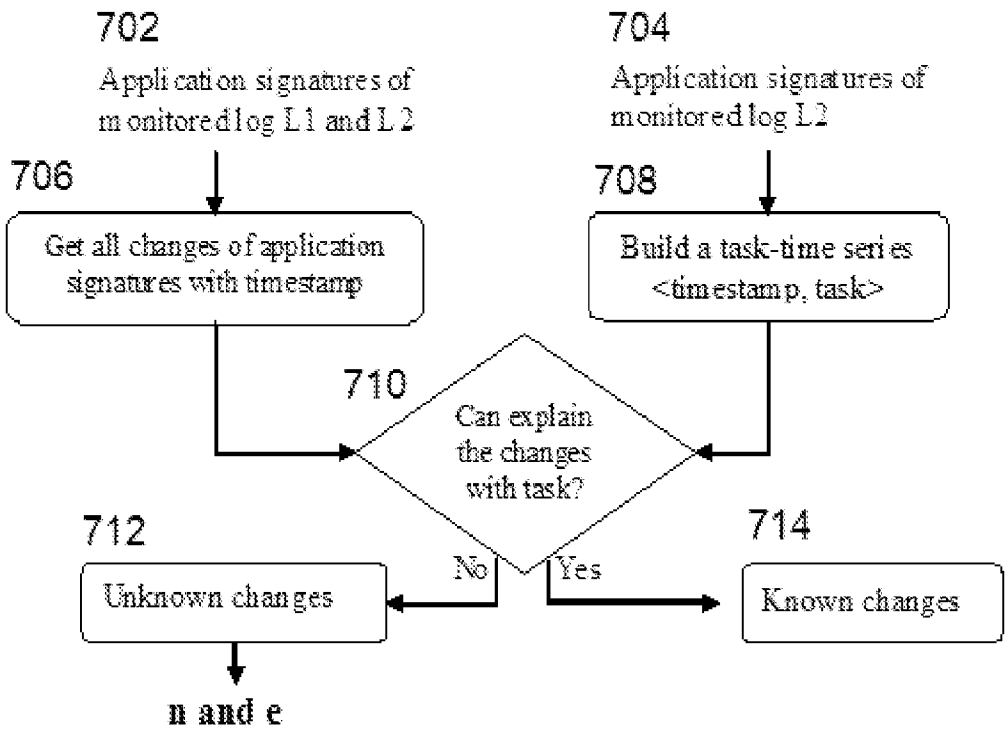

702
Application signatures of monitored log L1 and L2
704
Application signatures of monitored log L2
706
Get all changes of application signatures with timestamp
708
Build a task-time series <timestamp, task>
710
Can explain the changes with task?
712
Unknown changes
714
Known changes
No
Yes
n and e

FIG. 7

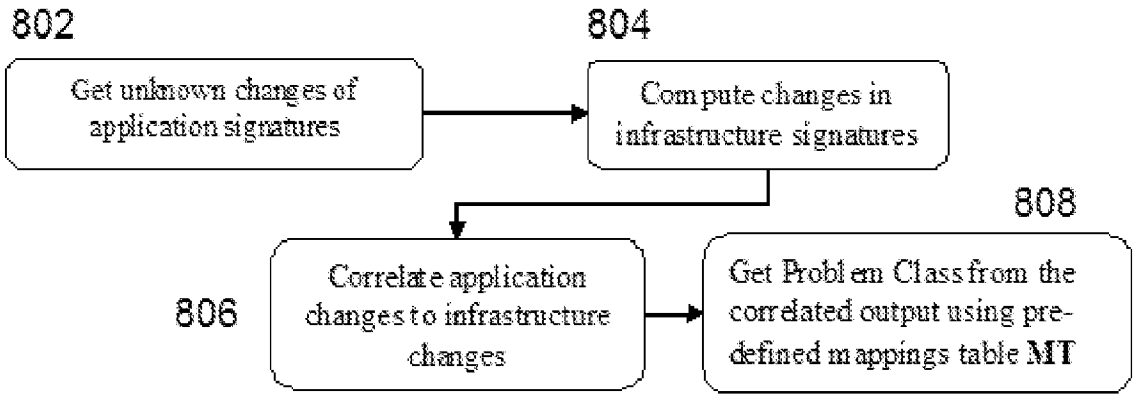

802
Get unknown changes of application signatures
804
Compute changes in infrastructure signatures
806
Correlate application changes to infrastructure changes
808
Get Problem Class from the correlated output using pre-defined mappings table MT

FIG. 8

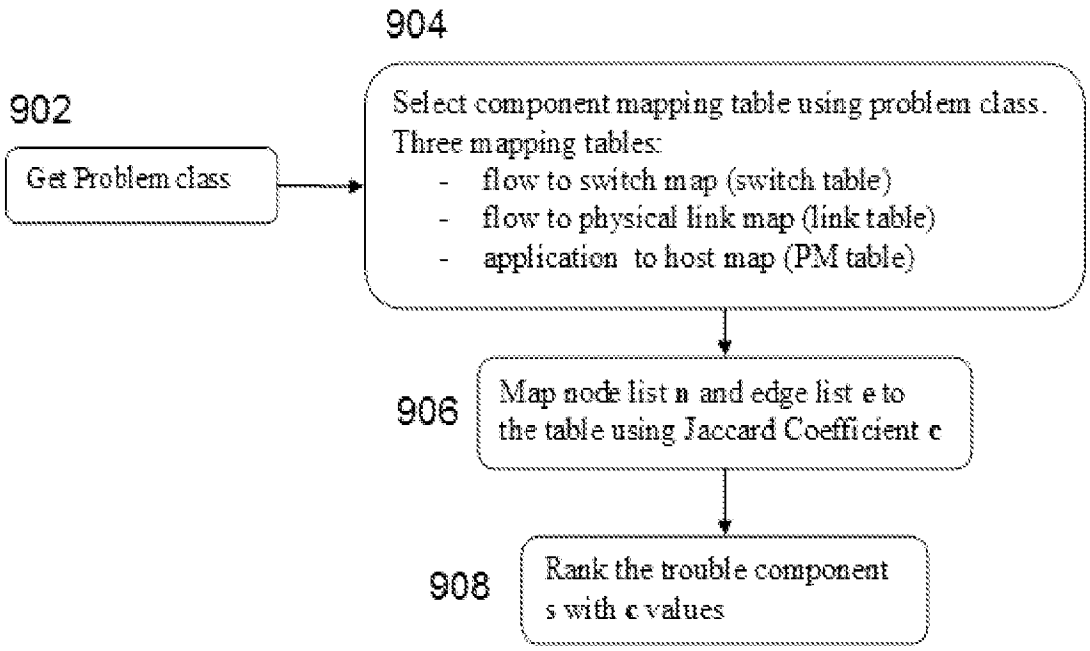

902
Get Problem class
904
Select component mapping table using problem class.
Three mapping tables:
- flow to switch map (switch table)
- flow to physical link map (link table)
- application to host map (PM table)
906
Map node list n and edge list e to the table using Jaccard Coefficient c
908
Rank the trouble components with c values

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2013/020795

A. CLASSIFICATION OF SUBJECT MATTER

G06F 11/36(2006.01)i, G06F 9/44(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 11/36; G06F 11/07; H04Q 7/00; G06F 15/173; G06F 17/30; H04L 12/26

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: debugging, network, application, modeling, traffic and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2006-0029016 A1 (AMIR PELES) 09 February 2006 See paragraphs 21-29, 61-90; figures 2a-2b; and claim 13.	1-3,8-10,12,14-15,17-18,20
A		4-7,11,13,16,19
Y	US 2009-0182794 A1 (ATSUJI SEKIHUCHI) 16 July 2009 See paragraphs 30, 52, 89-90; figure 10A; and claim 1.	1-3,8-10,12,14-15,17-18,20
A	US 7894357 B2 (WILLIAM J. PURPURA) 22 February 2011 See column 3, line 39 - column 6, line 10; and figures 1-2.	1-20
A	US 7916652 B1 (PAULO LIMA et al.) 29 March 2011 See column 6, line 4 - column 8, line 10; and figures 3-4.	1-20
A	US 7769851 B1 (KOWSIK GURUSWAMY et al.) 03 August 2010 See column 6, line 61 - column 7, line 40; and figure 5.	1-20

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
25 April 2013 (25.04.2013)	**25 April 2013 (25.04.2013)**
Name and mailing address of the ISA/KR Korean Intellectual Property Office 189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea Facsimile No. 82-42-472-7140	Authorized officer BYUN, Sung Cheal Telephone No. 82-42-481-8262

Form PCT/ISA/210 (second sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2013/020795

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2006-0029016 A1	09.02.2006	None	
US 2009-0182794 A1	16.07.2009	GB 0822370 D0	14.01.2009
		GB 2456619 A	22.07.2009
		JP 05119935 B2	02.11.2012
		JP 2009-169609 A	30.07.2009
US 7894357 B2	22.02.2011	US 2007-0206506 A1	06.09.2007
		US 2007-0206511 A1	06.09.2007
		US 2007-0274337 A1	29.11.2007
		US 2007-0297447 A1	27.12.2007
		US 2011-0004686 A1	06.01.2011
		US 7817536 B2	19.10.2010
		US 7929542 B2	19.04.2011
		US 7969879 B2	28.06.2011
US 7916652 B1	29.03.2011	None	
US 7769851 B1	03.08.2010	None	