Journal of Mobile Multimedia, Vol. 10, No. 1&2 (2014) 016-029


# FAULT RESOLUTION SYSTEM FOR INTER-CLOUD ENVIRONMENT

HA MANH TRAN SYNH VIET UYEN HA HUYNH TU DANG

*Computer Science and Engineering*

*International University-Vietnam National University*

Ho Chi Minh City, Vietnam

*{tmha, hvusynh, dhtu}@hcmiu.edu.vn*

KHOA VAN HUYNH

*Network and Services Management*

*VNPT Dong Thap-VNPT Group*

*Dong Thap, Vietnam*

*khoahv.dtp@vnpt.vn*

Fault resolution in communication networks and distributed systems is a complicated process that demands the involvement of system administrators and supporting systems in monitoring, diagnosing, resolving and recording faults. This process becomes more challenging in inter-cloud environment where multiple cloud systems coordinate in provisioning applications and services. In this context, we propose a fault resolution system that assists system administrators in resolving faults in inter-cloud environment. The proposed system is characterized by the capability of sharing and searching fault knowledge resources among cloud systems for fault resolution. It uses a peer-to-peer network of fault managers that provide facilities to monitor faults occurring in cloud systems and search similar faults with solutions occurring in other cloud systems. We have implemented several components of the proposed system including fault monitor, fault searcher and fault updater. We have also experimented and evaluated the prototyping system on fault databases obtained from several fault sources, such as bug tracking systems, online discussion forums and vendor knowledge bases.

*Key words*: Cloud Computing, Fault Resolution, Peer-to-Peer Network, Inter-Cloud Environment, Bug Tracking System

## 1. Introduction

Fault resolution in communication networks and distributed systems today becomes challenging due to the increasing degree of complexity, diversity and evolution of these systems. Resolving faults is a human driven process that demands the expertise of system administrators and the support of several systems. Monitoring and event correlation systems are usually used to produce fault reports, while trouble ticket systems (TTSs) are used to manage the fault reports for resolution.

Cloud computing is a new paradigm of provisioning infrastructure, platform, and software as services over the Internet. This paradigm combines distributed computing resources and virtualization technologies that outsource not only platform and software but also infrastructure to solve the demand of users. Cloud systems fostering the centralization of various services need a large number of system

administrators and supporting systems to manage faults occurring in the cloud systems and services. It is necessary to develop a supporting system that can exploit fault knowledge resources on cloud computing environment and assist system administrators in resolving faults.

We propose a fault resolution system for inter-cloud environment [1] that supports applications and services for running across multiple cloud systems. This system allows system administrators to monitor faults and search similar faults with solutions on cloud systems. The system recruits a peer-to-peer (P2P) network of fault managers running on cloud systems to facilitate sharing and searching fault knowledge resources among cloud systems for fault resolution. A fault manager acts as both a fault monitor of the cloud system and a fault searcher of the P2P network. It also maintains a fault updater to obtain fault reports frequently because software components and offered services change very dynamically. The contribution is thus threefold:

- Studying the existing fault monitoring, fault diagnosis, and failure prediction approaches on cloud systems, and proposing a fault resolution system based on P2P technology for federation of cloud computing environments
- Providing the design and implementation of several components of the proposed system including fault monitor, fault updater, and fault searcher with the support of open source tools
- Evaluating the feasibility and efficiency of the prototyping system using software bug reports crawled from bug tracking systems (BTSs) and Apache Hadoop clusters [2] on OpenStack cloud systems [3]

The rest of the paper is structured as follows: the next section presents the existing approaches related to fault monitoring, fault diagnosis, and failure prediction on cloud systems. This section also provides recent studies of fault resolution in communication networks and distributed systems using P2P technology. Section 3 describes the design and implementation of the fault resolution system that focuses on system architecture, component communication, and protocol support and fault data. Performance evaluation in Section 4 reports the feasibility and efficiency of the prototyping system with experiments and analyses before the paper is concluded in Section 5.

## 2. Background

This section explains the terminology and systems related to tracking faults. It also presents the studies of fault monitor and diagnosis on cloud systems and the recent studies of using P2P technology for fault resolution.

### *2.1 Terminology*

The terminology *fault* can be understood as *trouble*, *bug*, *issue*, *defect*, *incident* depending on the context. The X.790 recommendation from ITU-T [30] defines *trouble* as “any cause that may lead to or contribute to a manager perceiving degradation in the quality of service of one or more network services or one or more network resources being managed.” The RFC 1297 from the IETF [31] defines *trouble* as “a single malfunctioning piece of hardware or software that breaks at some time, has various efforts to fix it, and eventually is fixed at some given time.” System administrators usually use a trouble ticket system to manage troubles for the quality of communication services. Similarly, an

incident tracking system tracks incidents that need certain actions to be taken to resolve them. An issue tracking system is similar to incident tracking systems; the terms are often used interchangeably. A defect tracking system is used to track defects of engineering products. A bug tracking system is used to keep track of software bugs. These systems in general aim to improve the quality of products. They keep track of current problems, and maintain historical records of previously experienced problems. They use the same data format that contains pre-defined fields used to keep track of the status of the problem while textual descriptions are used to describe the problem. They also establish a reasoning engine that allows searching for similar past problems, and providing reports and statistics for performance evaluation of the services.

### *2.2 Fault Monitor and Diagnosis for Cloud Systems*

The study of Armbrust et al. has emphasized 10 obstacles for cloud computing [4]. Several obstacles are related to fault management, such as service availability, performance unpredictability, and bugs in large-scale distributed systems. Fault management for cloud computing has attracted several research activities. The authors of the study [5] have proposed an approach for realizing generic fault tolerance mechanisms as independent modules, validating fault tolerance properties of each mechanism, and matching user's requirements with available fault tolerance modules to obtain a comprehensive solution with desired properties. The study of Dudko et al. [6] has described the shortcomings of failure monitoring and prediction models in the context of Hadoop, and proposes a novel approach to predict performance failures in Hadoop clusters. The approach exploits the high-level data provided by Hadoop logs and the low-level data provided by hardware logs to predict failures more accurately. Thanamani [7] has studied various failure prediction methods in distributed systems. The study focuses on the information of hardware errors from individual nodes to predict failures for high performance computing clusters, Linux computing clusters or single servers. Faults can be observed at three stages: monitor of symptoms, detection of errors, or observation of failures. The authors of the study [8] have proposed an auto-recovery system for the job tracker that can be applied to Hadoop applications. The auto-recovery mechanism is based on a checkpoint method, which the snapshots of the job tracker are stored on a distributed file system periodically, and used later when the system detects any failure. The advantage of this system is the capability of continuing job execution during the recovery phase. The study of Garduno et al. [9] has proposed the Theia visualization tool that analyzes application-level logs in a Hadoop clusters, and generates visual signatures of job performance. Theia recruits heuristics to identify visual signatures of problems that allow users to distinguish application-level problems, e.g., software bugs or workload imbalance from infrastructural problems, e.g., contention problems, hardware problems. Tan et al. [10] have described a non-intrusive log-analysis approach that traces the causality of execution in a MapReduce system based on its behavior. The approach also proposes a novel way to characterize the behavior by decomposing performance along the dimensions of time, space, and volume. The approach can be useful for system administrators to debug performance problems in MapReduce systems. These studies deal with the same challenge: fault monitoring and diagnosis, or failure prediction.

### *2.3 Fault Resolution Using Peer-to-Peer Technology*

A P2P network is a collection of networked computers referred as peers. Peers play both roles of client and server that share computing resources including file, storage, bandwidth and processor power through consuming and provisioning services, respectively. The P2P network is constructed by a special communication mechanism that allows peers to join and leave dynamically with some degree of self-administration, fault-tolerance and scalability. P2P networks can be divided into three categories based on the mechanism of searching and sharing resources: centralized, structured and unstructured networks, as shown in Figure 1.

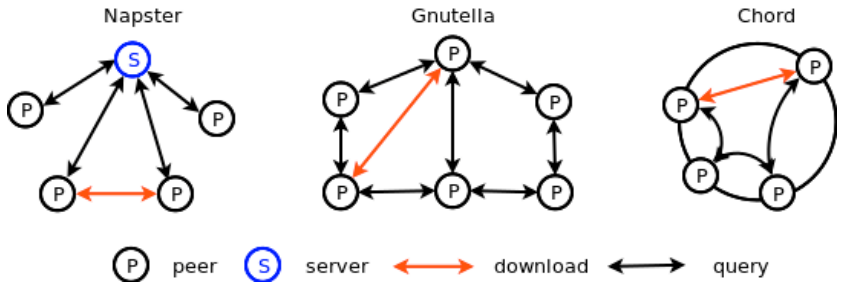



Figure 1 Searching and sharing resources on Napster, Gnutella and Chord

*A centralized P2P network* uses a centralized server to index resources from peers. A peer searches the resource indexes and corresponding peers from the server and directly downloads the resources from the corresponding peers. The typical network includes Napster. *A structured P2P network* uses Distributed Hash Table (DHT) to generate uniquely consistent identifiers for peers and resources such that the peers hold the resource indexes if their identifiers are in the same identifier space. Peers forward queries to other peers closer to the resource indexes in the identifier space to figure out the corresponding peers. The typical structured networks include CAN [24], Chord [25], and Kademlia [26]. *An unstructured P2P network* maintains resource indexes on peers. Peers search resources by flooding queries to other peers on the network, and the peers holding the resources send back their information for the download process. The typical unstructured P2P systems include Gnutella [27], Freenet [28], and BitTorrent [29]. In addition, a super peer P2P network is a hybrid network that combines the characteristics of the P2P network with the client-server network to address the problem of heterogeneous peers, i.e., peers possess various capabilities of storage, bandwidth and processing power. The study of Yang et al. [34] has presented guidelines for designing the super peer network to take advantage of peer capabilities. The super peer network comprises many clusters connected to each other to form either structured or unstructured P2P networks, in which each cluster contains a super peer and a set of clients. The clients submit queries to, and also obtain queryhits from, their super peer while the super peers forward the queries and receive the queryhits on the super peer network.

There are a huge number of studies in fault resolution for communication networks and distributed systems. Recent studies consider using P2P technology for fault resolution. The study of Tran et al. [11, 12, 13, 14, 15] has proposed the DisCaRia distributed case-based reasoning system that combines P2P technology with the conventional case-based reasoning (CBR) system [16] to explore problem solving knowledge resources in distributed environments. The prototyping system has been deployed

and currently measured on the EMANICSLAB distributed computing testbed [17]. The authors of the study [18] have recently proposed a network search system for network management based on distributed network search algorithms as a primitive. The system runs on an overlay network that provides distributed query processing functions to retrieve management and monitoring data from network elements. The approach shares some similarity with the ideas behind the DisCaRia system. However, the system primarily aims to provide a generic search mechanism for management and monitoring data. The study in this article continues these research activities by proposing a fault resolution system for inter-cloud environment using P2P technology. This system not only works as a TTS for cloud systems but also features the capability of exploiting fault knowledge resources on cloud computing environment for fault resolution.

## 3. Fault Resolution System

The fault resolution system features the capability of sharing and searching fault knowledge resources for fault resolution on inter-cloud systems. This section describes appropriate system architecture, component communication, and protocol support and fault data.

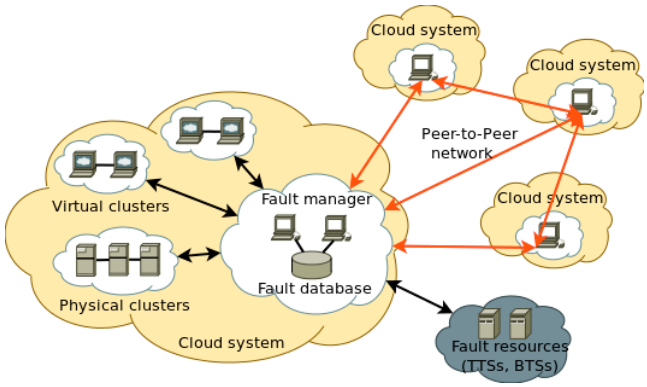



Figure 2 An architecture of the fault resolution system

### *3.1 System Architecture*

The system architecture relies on a P2P network that contains remarkable characteristics including autonomous management, scalable architecture and efficient content distribution in distributed environment. Searching on a P2P network can be more effective because queries can be processed by groups of workstations on large domain-specific databases, thus reducing high computing cost on centralized servers. The heart of the system architecture is a fault manager that contains the components of fault monitor, fault resolver and fault database. The fault manager is responsible for monitoring faults occurring on cloud systems, diagnosing faults, searching similar faults on a local database and sharing faults with other fault managers.

Each cloud system configures a fault manager as shown in Figure 2. The fault manager works as a trouble ticket system for the cloud system. It also updates fault data from various fault resources such as BTSs, online discussion forums and vendor knowledge bases. The connection of fault managers forms a fault resolution system using a P2P network that facilitates searching and sharing fault

knowledge resources among cloud systems. This system architecture is more suitable for inter-cloud environment where cloud systems integrate several common packages for applications and services.

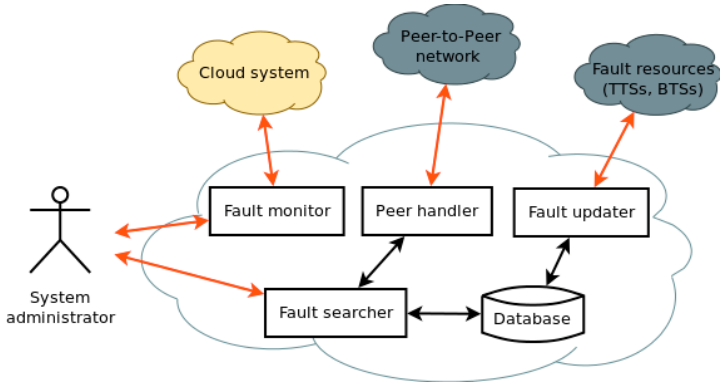



Figure 3 Component communications of the fault resolution system

### *3.2 Component Communication*

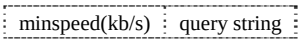
The *fault manager* communicates with several inside and outside systems as shown in Figure 3. The *fault monitor* component is used to monitor the cloud system. This component contains monitoring tools, such as Ganglia [19], Nagios [20], Flume [21]. These tools check several parameters of services, networks, servers and clusters, and send reports to the system administrators. They also collect trace and log data for fault reports. A challenge of this component is to analyze fault reports to notify and assist the system administrators in dealing with faults on the cloud system. This study does not focus on this challenge, instead the component helps on providing fault reports that can be used to query the proposed system. The *fault searcher* component is a search engine that allows system administrators to query fault reports and to obtain similar fault reports with solutions. This search engine seeks fault reports and solutions either in the local database or from the other fault managers through a peer handler component. It supports both keyword and semantic search methods. The *peer handler* component allows the fault manager to communicate with other fault managers. This component uses a P2P protocol to maintain the connection of peers and forward queries to and receive queryhits from peers. The protocol contains a mechanism to check the neighboring peers alive regularly. Finally, the *fault updater* component is used to update the local fault database. It contains crawlers that retrieve fault reports from multiple fault sources such as BTSs, online discussion forums and vendor knowledge bases.

### *3.3 Protocol Support*

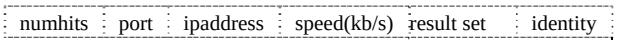
We investigate several structured and unstructured P2P networks. A structured P2P network is tightly controlled in topology where a peer or a resource is fixed in a logical location using DHT. This network offers efficient search mechanisms with high success search rate, but suffers from fault tolerance when a large number of peers leave the network. An unstructured P2P network is loosely controlled in topology where a peer connects to other peers in a random fashion and maintains a list of resources in the local repository. This network offers complex search mechanisms with high resource

availability, but suffers from severe scalability due to high traffic generation. The heterogeneity problem of most P2P networks is the existence of both weak and strong peers in the networks.

*Original Gnutella query message*




*Original Gnutella queryhit message*




We have chosen to implement the Gnutella P2P protocol version 0.6 for the proposed system. This protocol includes the concept of super peers in the Gnutella network. A super peer uses sufficient bandwidth, uptime, and processing power to undertake heavy operations. The super peers act as proxies to the Gnutella network for peers, and queries are only forwarded among the super peers. The super peers increase the scalability of the Gnutella network by reducing a large number of peers with limited capability incorporating in the query routing process and thus reducing a large amount of network traffic generated on the network. The Gnutella protocol basically supports a simple keyword based search mechanism, while the proposed system aims at providing a complex search mechanism. The protocol is therefore modified to enable complex queries and queryhits. The modification of two query and queryhit messages is rather straightforward. The query string of the query message contains several parts of fault reports, while the queryhit message contains a set of results that is usually large in size and complex in structure. The previous studies [11, 12] have extended the protocol similarly.

### *3.4 Fault Data*

Table 1 Popular bug tracking sites (as of Jan. 2013). A plus indicates that we were unable to get precise numbers and our numbers present a lower bound

| Site | System | Bugs |
|---|---|---|
| bugs.kde.org | Bugzilla | 306489 |
| bugs.eclipse.org | Bugzilla | 396232 |
| bugs.gentoo.org | Bugzilla | 352772 |
| bugzilla.mozilla.org | Bugzilla | 819715 |
| bugzilla.redhat.com | Bugzilla | 879989 |
| qa.netbeans.org | Bugzilla | 218589 |
| issues.asterisk.org | Mantis | 21772 |
| bugs.scribus.net | Mantis | 11247 |
| bugtrack.alse-project.org | Mantis | 5586 |
| dev.rubyonrails.org | Trac | 11446 |
| trac.edgewall.org | Trac | 7326 |

| | | |
|---|---|---|
| bugs.icu-project.org | Trac | 4324 |
| bugs.launchpad.net/ubuntu | Launchpad | 648407+ |
| bugs.launchpad.net/launchpad | Launchpad | 27514+ |

We have developed bug crawlers to retrieve fault data from BTSs, online discussion forums and vendor knowledge bases. A bug crawler obtains bug reports from BTSs that support XML-RPC web service interfaces, such as Bugzilla. This crawler is very efficient in updating bug reports because it can get a large number of bug reports in a single request. However, it limits the attachments of bug reports. Another bug crawler uses HTML web interfaces to obtain bug reports from various sources, such as Launchpad, networking forums. This crawler is rather slow and complex in obtaining bug reports due to the inconsistency of HTML pages and the variety of HTML structure presentations. The crawler also contains several HTML parsers corresponding to tracking sites. Table 1 reports four popular bug tracking sites with bug numbers: Bugzilla, Mantis, Trac and Launchpad. BTSs using the same sites usually present the similar structures of bug reports. We have also used a unified bug schema [33] to store bug reports with different formats. Open source software packages, such as Ubuntu, Redhat, Gentoo, Mozilla contain large numbers of bug reports that can be useful fault data sources for fault resolution because several cloud computing software toolkits, such as Eucalyptus [22], OpenNebula [23], OpenStack [3] support these platforms.

## 4. Performance Evaluation

We have evaluated the feasibility and efficiency of the fault resolution system on the inter-cloud environment of multiple cloud systems. For feasibility, we measure the capability of the system to contribute a large amount of fault data while minimizing data transfer, retrieve a large number of queryhits with short responding time. We also assess the traffic generation and failure issues of the P2P network. For efficiency, we measure the capability of the system to provide relevant fault reports. To configure experiments, we extend the implementation of the Gnutella P2P protocol [11, 12] to including fault monitor, update and search operations. We have crawled software bug reports from several BTSs to build the fault databases as shown in Table 1. The system contains several peers (or workstations) with different platforms; each peer maintains a fault database updated by BTSs and comprises a fault manager that connects to fault managers on other cloud systems. The fault manager uses Ganglia and Flume to monitor Apache Hadoop clusters and obtain log data on OpenStack cloud systems. The search mechanism used in this study exploits error symptoms, log messages, bug relationship and dependency to retrieve similar bug reports [33].

The first experiment measures the data contribution and data transfer of peers. Data contribution increases quickly as the number of peers increases, as shown in Figure 4. Each peer contributes 50.000 bug reports on average to the network due to the limitation of computing capability. The whole network therefore contributes a huge number of bug reports that increase the possibility of finding solutions. Data transfer is the amount of queryhit data responded by peers to a set of 50 queries. It is also traffic generated on the network when 50 queries are sent concurrently. Data transfer increases slowly as the number of peers increases, depending on the size of queryhit data. We need to balance between a large number of queryhits obtained and a small amount of traffic generated.

The second experiment compares time consumption to obtain queryhits for different numbers of peers. Each peer responds 30 queryhits on average for a query due to the problem of traffic generation. Figure 5 shows that at least 3 or 4 peers respond after 5 seconds, and 6 or 7 peers respond after 8 seconds. Time consumption increases as the number of peers increases. However, the number of peers involved in a query is limited by the protocol, thus reducing the possibility of finding solutions. Moreover, time consumption can increase significantly if peers run on the Internet. We need to balance between a large number of queryhits and fast response time.

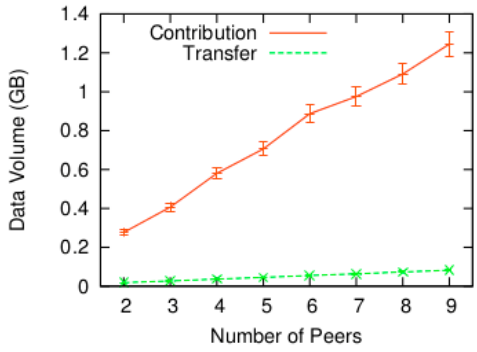



Figure 4 Data contribution and data transfer over the number of peers

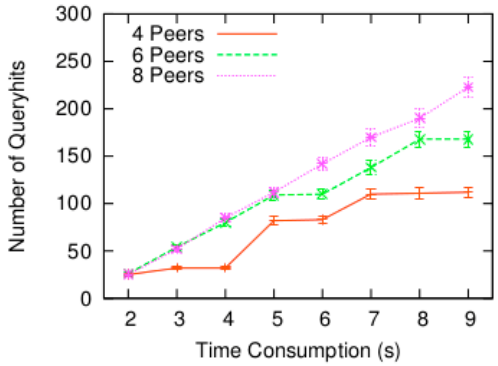



Figure 5 Comparison of time consumption and number of queryhits for different numbers of peers

The third experiment computes traffic generation for processing queries and queryhits on peers, super peers and Gnutella peers. The appearance of Gnutella peers is used for comparison because the Gnutella protocol uses the message flooding mechanism that causes large amount of traffic on the network. When receiving a message, both Gnutella peers and super peers react similarly by flooding the message to the neighboring peers, and peers only forward the message to some super peers.

Figure 6 reports that peers generate less traffic to the network comparing to Gnutella peers and super peers; i.e., 18MB comparing to 44MB over 10 thousand queries. Gnutella peers generate the

same amount of traffic as super peers. This experiment also considers the uncertainty of the Gnutella and super peer networks. Heterogeneous peers on the Gnutella network are less stable than super peers that possess sufficient capability of performing complex operations.

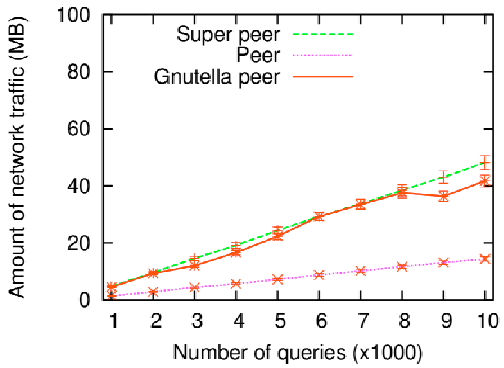



Figure 6 Traffic generation for query and queryhit processing

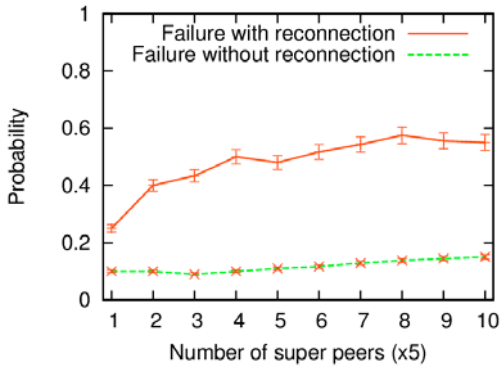



Figure 7 Failure probability of super peers

The fourth experiment reports the failure probability of super peers in two situations: failure with reconnection and failure without reconnection. Super peers in the former situation can disconnect to some super peers and connect to other super peers for network performance enhancement, while super peers in the later situation can cause data unavailability for peers due to unexpected reasons. Since super peers have been selected by high uptime, the possibility of super peers to go offline is thus low except for unexpected software or network problems, i.e., 10% to 15% of super peers can fail without reconnection. However, the possibility of super peers to go offline and online is relatively high, i.e., 45% to 50% of super peers can reset connections during the experiment period, as shown in Figure 7.

The fifth experiment evaluates the capability of peers to obtain identical bug reports. We have created query sets whose queries are selected from three bug databases: Bugzilla Redhat, Launchpad Ubuntu and Mantis Asterisk. Peers contain overlapping sets of bug reports intentionally, i.e., two peers

contain Redhat bug reports ranging from id=50.000 to id=100.000 and from id=75.000 to id=125.000, respectively.

Figure 8 presents precision rate over recall rate using the query sets of 50 queries. Peers with the Redhat and Asterisk databases perform better than peers with the Ubuntu database. Precision rate reaches 0.65 as recall rate reaches 0.5 for three databases. Bug reports in the Ubuntu database are large in size, but contain much unrelated information, thus affecting precision rate.

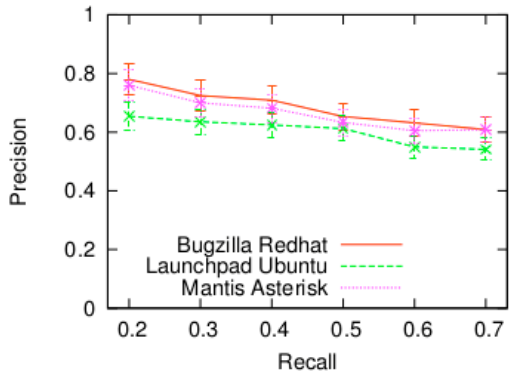



Figure 8 Precision rate over recall rate for the identical bug reports

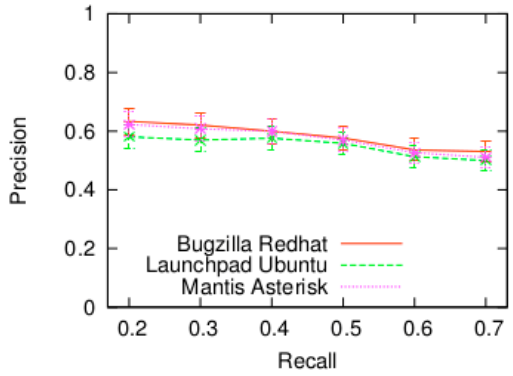



Figure 9 Precision rate over recall rate for the similar bug reports

The last experiment evaluates the capability of peers to obtain similar bug reports. Estimating similar bug reports is difficult because bug databases remain largely unexplored. We have changed the query sets in the third experiment by removing some keywords and symptoms from queries and considered the resulting sets of the identical bug reports as the sets of similar bug reports. Figure 9 reports precision rate over recall rate using the query sets of 50 queries. Peers with three databases perform similarly. Precision rate in this experiment is lower than precision rate in the third experiment. The number of similar bug reports retrieved increases as queries possess fewer keywords and symptoms. However, precision rate is reduced because several similar bug reports are unrelated to the queries.

## 5. Conclusions

We have proposed and implemented a fault resolution system for inter-cloud environment that supports applications and services for running across multiple cloud systems. This system allows system administrators to monitor faults, collect fault data and search similar faults with solutions on other cloud systems. The system uses a P2P network of fault managers located on cloud systems to facilitate sharing and searching fault knowledge resources for fault resolution. A fault manager acts as both a fault tracker of the cloud system and a fault searcher of the P2P network. We have studied several P2P protocols and extended the Gnutella P2P protocol to including the fault monitor, update and search operations. While the fault monitor recruits the existing monitoring tools, such as Ganglia, Nagios and Flume, the fault searcher (or peer) requires not only protocol support but also search mechanisms and fault databases. We have built fault databases obtained from software bug reports of BTSs and Apache Hadoop clusters on OpenStack cloud systems. Performance evaluation focuses on the feasibility and efficiency of the prototyping system on the inter-cloud environment of multiple cloud systems. The first two experiments report the high volume of fault data contributed by the proposed system and the low network traffic generated by peers when performing search operations. The experiments also compare time response between different numbers of peers. The next two experiments investigate the traffic generation and failure issues of the P2P network that can influence the scalability of the system. The last two experiments evaluate the capability of peers to obtain relevant bug reports. Note that finding similar faults with solutions in a short period of time can be significant for cloud systems in maintaining the normal operation of applications and services. The precision rates of three bug datasets are fairly similar given the same recall rates. Future work focuses on analyzing faults from the fault monitor to perform the fault searcher without the interference of the system administrators.

## Acknowledgements

This research work is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2011.01.

## References

1. R. Buyya, R. Ranjan, and R. N. Calheiros (2010), *Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services*, In Proc. 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10), pp 13-31, Heidelberg, Germany, Springer-Verlag.
2. Apache Hadoop Project (2005), http://hadoop.apache.org/, last access in July 2013.
3. OpenStack Cloud Software (2010), http://www.openstack.org/, last access in July 2013.
4. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia (2010), *A View of Cloud Computing*, ACM Communications, Vol. 53, No. 4, pp 50-58.
5. R. Jhawar, V. Piuri, and M. Santambrogio (2012), *Fault Tolerance Management in Cloud Computing: A System-Level Perspective*, Systems Journal, Vol. 7, No. 2.
6. R. Dudko, A. Sharma, and J. Tedesco (2012), *Effective Failure Prediction in Hadoop Clusters*, Technical Report, University of Illinois.

7. A. S. Thanamani (2011), *A Survey on Failure Prediction Methods*, International Journal of Engineering Science and Technology (IJEST), Vol. 3, No. 2.
8. N. Kuromatsu, M. Okita, and K. Hagihara (2013), *Evolving Fault-Tolerance in Hadoop with Robust Auto-Recovering JobTracker*, Bulletin of Networking, Computing, Systems, and Software, Vol. 2, No. 1.
9. E. Garduno, S. P. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan (2012), *Theia: Visual Signatures for Problem Diagnosis in Large Hadoop Clusters*, In Proc. 26th International Conference on Large Installation System Administration: Strategies, Tools, and Techniques (LISA'12), pp 33-42, Berkeley, USA, USENIX Association.
10. J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan (2010), *Visual, Log-based Causal Tracing for Performance Debugging of MapReduce Systems*, In Proc. 2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS'10), pp 795-806, Washington, USA, IEEE Computer Society.
11. H. M. Tran and J. Schönwälder (2007), *Fault Representation in Case-Based Reasoning*, In Proc. 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, pp 50-61, Springer-Verlag.
12. H. M. Tran and J. Schönwälder (2007), *Heuristic Search using a Feedback Scheme in Unstructured Peer-to-Peer Networks*, In Proc. 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing, Springer-Verlag.
13. H. M. Tran and J. Schönwälder (2008), *Fault Resolution in Case-Based Reasoning*, In Proc. 10th Pacific Rim International Conference on Artificial Intelligence (PRICAI '08), pp 417-429, Springer-Verlag.
14. H. M. Tran, G. Chulkov, and J. Schönwälder (2008), *Crawling Bug Tracker for Semantic Bug Search*, In Proc. 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '08), pp 55-66, Springer-Verlag.
15. H. M. Tran and J. Schönwälder (2011), *Evaluation of the Distributed Case-Based Reasoning System on a Distributed Computing Platform*, In Proc. 7th International Symposium on Frontiers of Information Systems and Network Applications (FINA 2011), pp 53-58.
16. A. Aamodt and E. Plaza (1994), *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*, AI Communications, Vol. 7, No. 1, pp 39-59.
17. D. Hausheer and C. Morariu (2008), *Distributed Test-Lab: EMANICSLab*, The 2nd International Summer School on Network and Service Management (ISSNSM '08), University of Zurich, Switzerland.
18. M. Uddin, R. Stadler, and A. Clemm (2013), *A Query Language for Network Search*, In Proc. 13th IFIP/IEEE International Symposium on Integrated Network Management (IM '13), IEEE Computer Society.
19. Ganglia Monitoring System (2000), http://ganglia.info/, last access in July 2013.
20. The Industry Standard In IT Infrastructure Monitoring (1999), http://www.nagios.org/, last access in July 2013.
21. Apache Flume (2009), http://flume.apache.org/, last access in Jan. 2014.
22. Eucalyptus Open Source AWS Compatible Private Clouds (2008), http://www.eucalyptus.com/, last access in Jan. 2014.
23. OpenNebula Flexible Enterprise Cloud Made Simple (2008), http://opennebula.org/, last access in Jan. 2014.
24. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker (2001), *A Scalable Content Addressable Network*, In Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01), pp 161-172, New York, USA, ACM Press.

25. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan (2001), *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*, In Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01), pp 149-160, New York, USA, ACM Press.
26. P. Maymounkov and D. Mazières (2002), *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*, In Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS '01), pp 53-65, London, UK, Springer-Verlag.
27. Gnutella Protocol Specification version 0.4 (2001), http://rfc-gnutella.sourceforge.net/developer/stable/index.html, last access in Mar. 2013.
28. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong (2000), *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, In Proc. International Workshop on Design Issues in Anonymity and Unobservability, pp 46-66, Heidelberg, Germany, Springer-Verlag.
29. B. Cohen (2003), *Incentives Build Robustness in Bittorrent*, In Proc. 1st Workshop on Economics of Peer-to-Peer Systems.
30. ITU-T (1995), *Trouble Management Function for ITU-T Applications*, X.790 Recommendation.
31. D. Johnson (1992), *NOC Internal Integrated Trouble Ticket System Functional Specification Wishlist*, RFC 1297.
32. D. Bloom (1994), *Selection Criterion and Implementation of a Trouble Tracking System: What's in a Paradigm?*, In Proc. 22nd Annual ACM SIGUCCS Conference on User Services (SIGUCCS '94), pp 201-203, New York, USA, ACM Press.
33. H. M. Tran, S. T. Le, S. V. U. Ha, and T. K. Huynh (2013), *Software bug ontology supporting bug search on peer-to-peer networks*, In Proc. 6th International KES Conference on Agents and Multi-agent Systems Technologies and Applications (AMSTA '13), IOS Press.
34. B. Yang and H. Garcia-Molina (2003), *Designing a super-peer network*, In Proc. 19th International Conference on Data Engineering (ICDE'03), pp 49, Los Alamitos, USA, IEEE Computer Society