Aging-related bugs in cloud computing software

Fumio Machida, Jianwen Xiang, Kumiko Tadano, Yoshiharu Maeno
NEC Knowledge Discovery Research Laboratories
{f-machida@ab, j-xiang@ah, k-tadano@bq, y-maeno@aj}.jp.nec.com

***Abstract*— Cloud computing is established on advanced software technologies intended to enhance the scalability of computing infrastructure by making full use of commodity servers. The more cloud computing relies on software technologies, the more software bugs have significant impacts on the system availability. Aging-related bugs, which cause the accumulation of errors in long time software execution, often remain even after the release of a stable version of the software. This paper investigates the bug reports of five major open-source software projects related to cloud computing and confirms the issues about the existence of aging-related bugs. From the investigation how the developers work around such aging-related bugs, the importance of the tool assistance for reproducing the aging problem in replicated site is discussed.**

Keywords-component; aging-related bugs; cloud computing; Key-value store; MapReduce; Open source software.

I. INTRODUCTION

Aging-related bugs have been observed in a number of software products and considered as the possible causes of system failures due to resource depletion like memory leak [1]. It is difficult to remove such aging-related bugs completely in the software development and testing phases. Therefore, residual aging-related bugs are often detected in the runtime after the software release. Depending on the execution environment where the software is deployed, system failures caused by the aging-related bugs may not be observed. The limited observability of the failure makes it further difficult to locate and fix such aging-related bugs.

Cloud-oriented software such as Hadoop MapReduce[1], Cassandra[2] and Eucalyptus[3] etc. might also suffer from aging-related bugs. Many open source software (OSS) projects for cloud computing has been launched in recent years and they are used worldwide for commercial services even though some of them are still in beta release. These software products are often designed to be scalable. Hadoop MapReduce enables distributed parallel computing by the combination of map tasks and reduce tasks, Cassandra provides a scalable distributed key-value store, and Eucalyptus manages a number of virtual machines on virtualized infrastructure. However, dealing with scalability in the software design and development is not so easy. Since they need to handle hundreds or thousands of objects simultaneously, resource management in such software tends to be complicated. Immature design and implementation can easily lead to software faults in resource management such as aging-related bugs.

In this paper, we characterize the actual aging-related bugs found in cloud-oriented software by investigating the bug reports of five OSS projects related to cloud computing. By reading the descriptions of the bug reports filtered by key word search, we identify the unique sets of the issues originated from aging-related bugs. The obtained bugs are then analyzed in terms of the ratio of the aging-related bugs, the resource types affected by aging, and the time to fix the issues. The target projects are Hadoop MapReduce, Cassandra, Eucalyptus, memcached[4] and Xen[5]. The basic functionalities of those OSS projects are introduced in the next section.

II. CLOUD-ORIENTED OSS PROJECTS

Five characteristic OSS projects related to cloud computing are chosen for the investigation. Although there are several similar projects, the selected OSSs are widely used for their special functionalities such as MapReduce framework, NoSQL, virtual machine manager, cache server and hypervisor.

A. Hadoop MapReduce

Hadoop MapReduce is an open source implementation of MapReduce framework that processes vast amounts of data in parallel on clustered computers. Hadoop MapReduce was originally inspired by Google's MapReduce and has been developed by worldwide contributors under Apache software foundation. Hadoop is widely adopted by many organizations including Yahoo!, Facebook, Amazon.com, Twitter, IBM, and Microsoft. The code is written in Java and the current latest version is 2.0.0-alpha which was released in May 2012. Software bugs reported on Hadoop MapReduce are managed by JILA, a software project tracking tool, hosted on apache software foundation. All of statuses, priorities, descriptions and discussions of bugs from April 2006 are available on the web site.

B. Cassandra

Cassandra was developed by Facebook as a scalable database management system and was released as an Apache open source project through Google code and Apache Incubator. Cassandra provides a structured key-value store and supports rich data model such as column

[1] Hadoop MapReduce, http://hadoop.apache.org/mapreduce/
[2] Apache Cassandra, http://cassandra.apache.org/
[3] Eucalyptus, http://open.eucalyptus.com/
[4] Memcached, http://memcached.org/
[5] Xen, http://www.xen.org/

families in Google BigTable [2]. It is designed to handle large amounts of data spread out across clustered node while providing a highly available service with no single point of failure. Cassandra is used in Facebook, Rackspace, Twitter, Netflix, Cisco, etc. The code is written in Java and the current latest release is version 1.1.4, which was released in August 2012. Software bugs in Cassandra are tracked by JILA in Apache software foundation. The bug reports have been posted from March 2009.

C. Eucalyptus

Eucalyptus is a middleware for constructing a private Infrastructure as a Service (IaaS) cloud that is accessible via APIs compatible with Amazon EC2[6] and Amazon S3[7]. Eucalyptus was originally developed by MAYHEM labs in UC Santa Barbara [3]. Currently Eucalyptus fully supports KVM[8] and Xen. In addition, the Enterprise Edition Eucalyptus supports the proprietary VMware hypervisor. Several programming languages are used for the implementation including Java, C, python and perl. The current stable release of Eucalyptus is version 3.1, which was released in June 2012. Software bugs in Eucalyptus were originally tracked on Launchpad[9], but later they are migrated to JILA-based system. The bugs are reported from the version 1.5.x series in 2009.

D. memcached

Memcahed is a widely adopted general-purpose distributed memory caching system. It maintains a key-value associative array in memory. Clients populate this array and query it through simple API. Many organizations use memcached to speed up web application systems using data bases by caching query results. The software is used in Wikipedia, YouTube, Twitter, Mixi, etc. It is implemented by C and the latest version is 1.4.14 released in July 2012. The project is managed in Google code and bug reports are posted on the issued list. On the issue list, bugs are tracked from the version 1.2.6 in October, 2008.

E. Xen

Xen is an open source implementation of virtual machine monitor (VMM) that allows multiple operating systems execute on a single physical server concurrently. Xen was originally developed in University of Cambridge Computer Lab. and the first public release of Xen is occurred in 2003. Currently, non-commercial XenServer is fully open-source and available to the public. Xen is implemented by C except some scripts which are written by python. The latest release of Xen hypervisor is version 4.1.3 which was released in August 2012. Software bugs in Xen projects are managed in Bugzilla hosted by xen.org. The bugs are tracked from an unstable version in April 2005.

[6] Amazon EC2, http://aws.amazon.com/ec2/
[7] Amazon S3, http://aws.amazon.com/s3/
[8] Kernel based Virtual Machine, http://www.linux-kvm.org/
[9] Launchpad, https://launchpad.net/

III. BUG REPORTS

For the five OSS projects, we investigate the bug reports on the bug tracking systems from the initial reports to the last reports until January 31st, 2012. In the reported bugs, there are duplicated reports originated from the same software bug, invalid reports which are not exactly associated with the faults in the software, and unsolved issues whose causes are not identified so far. First, we remove those duplicated reports and invalid reports based on the resolution status assigned by the developers. Unsolved issues are counted in the results because we analyze the amount of unresolved issues as well. Next, we filter the reports by keyword search using the key words: “leak”, “increas”, “descreas”, “deplet”, “exhaust”, “exceed”, “aging”, “Out of memory” and “OOM”. The verb keywords are written in a special form (e.g., “increas” instead of “increase”) to match both present and progressive tenses. By reading the descriptions of filtered bug reports carefully, we identify the unique set of reports caused by aging-related bugs following the definition in [1]. The statistics of aging-related bugs are summarized and discussed in the following sections.

A. Amount of aging-related bugs

First, we count the total number of the valid bug reports and the number of the reports caused by aging-related bugs as in TABLE I. Aging-related bugs are confirmed in all of the five projects. The results reveal that the recent OSS products for cloud computing contain aging-related bugs regardless of the functionalities provided by the software. The related statistics are also summarized in TABLE I. Bug rates are computed by dividing the total number of valid bug reports by the bug tracking period in months. The ratio of aging-related bugs represents the ratio of the number of reports for aging-related bug to the total number of valid reports. The bug rates may be affected by the amount of activities (updates in a certain time period) in the project. Both Hadoop MapReduce and Cassandra are extensively active projects and their bug rates are higher than the others. The ratios of aging-related bugs in the all five projects are observed around at 0.01 (from 0.4% to 1.4%).

B. Types of aging-related bugs

Next, the aging-related bugs are categorized by the resources affected by aging in long time execution. Software objects such as file descriptors, database connections, data objects and threads are also considered as software resources which should have a finite limit of their amounts. The violation of such limitation often causes a memory leak problem in its execution environment. As summarized in TABLE II, amongst all the resource categories, file descriptors are the most affected resources by aging-related bugs. Many bug reports of Hadoop MapReduce and Cassandra alert possible memory leaks due to the faults in closing file descriptors. Similarly, connections used in software such as database connections or network connections are likely to be unreleased even after the completion of their missions. Such unreleased connections often cause a memory leak. Although the aging-related bugs largely belong to these two categories, fixing such bugs are

TABLE I. AGING-RELATED BUGS FOUND IN THE FIVE OSS PROJECTS

	Project period (months)	The total number of valid bug reports	The number of aging-related bugs	Bug rate (per month)	Ratio of aging-related bugs to the total reports
Hadoop MapReduce	70	1802	21	25.74286	0.011653718
Cassandra	35	1605	17	45.85714	0.0105919
memcached	39	179	1	4.589744	0.005586592
Eucalyptus	37	540	8	14.59459	0.014814815
Xen	79	1612	8	20.43038	0.004956629

TABLE II. THE NUMBER OF AGING-RELATED BUGS IN RESOURCE CATEGORIES

	File descriptors	Connections/ Pipes	Data objects/ Variables	Threads	External resources	Unknown	Others
Hadoop MapReduce	5	1	8		3	2	2
Cassandra	10	4	2			1	
memcached			1				
Eucalyptus		1	3	3			1
Xen	2	1	1			3	1

TABLE III. RESOLUTION STATUSES OF THE ALL REPORTS AND AGING-RELATED BUGS IN THE FIVE OSS PROJECTS

	All bugs				Aging-related bugs	
Resolution Status	Fixed or resolved	Unresolved	Won't Fix	Cannot reproduce	Fixed or resolved	Unresolved
Hadoop MapReduce	957	779	41	25	13	8
Cassandra	1442	48	37	78	17	0
memcached	146	9	24	–	1	0
Eucalyptus	–	–	–	–	8	0
Xen	858	641	115	–	5	3

relatively easy once the unreleased object is identified. In fact, most of the bug reports in these categories are resolved in a short period. Besides the file descriptors and connections, data objects and thread handlers also confront the risk of leakage due to the missing operations to release.

In some aging-related bugs in Hadoop MapReduce affect the external resources such as the number of files in a directory, the number of zombie processes, the objects in external software components. We categorize these resource types as external resources as they persist independently of the software execution. Compared to the first four categories, this type of aging-related bugs is difficult to find by verification of single software component. Since the aging phenomena caused by this type of aging-related bugs depend on the configuration of the execution environment, the limited reproducibility of the aging becomes a problem.

C. Resolutions of aging-related bugs

Once the root-cause of software aging is identified, the developers work around the bug to create a patch. When the patch is tested and the removal of the problem is confirmed, the bug report is closed as resolved status. The resolution statuses of aging-related bugs at 31st January 2012 are summarized in TABLE III. The data for Eucalyptus is not presented due to the inaccessibility to the Launchpad which is discarded after employing the new bug tracking system (in June 2012). The status "Cannot reproduce" is used only in JILA-based system (i.e, Hadoop MapReduce and Cassandra). This status maybe categorized into "Unresolved" or "Won't Fix" in the other projects.

Hadoop MapReduce and Xen have surprising numbers of unresolved issues. Although such undesirable numbers of unresolved problems can be accounted by the length of the projects in a sense, the immaturities of the software should be alerted to the users. In contrast, the number of unresolved bugs in Cassandra is considerably small instead of its high bug rate as seen in TABLE I. In particular, all of the aging-related bugs in Cassandra are in the fixed status at this time. Although the high bug rate indicates the immaturity of the source code, the small number of unresolved bugs implies the well-organized bug tracking community. All of the aging-related bugs in memcached and Eucalyptus are also in the fixed status.

For Hadoop MapReduce, periodic clean-up of unnecessary objects are adopted as a tentative or a permanent solution to aging-related bug instead of removing the bug. Periodic clean-up is also used for a tentative measure in the other projects until the corresponding bug-fix is available. The solution relying on the garbage collection in Java virtual machine (JVM) can also be considered as an approach using

periodic clean-up. Some aging-related bugs in Java based projects are fixed by replacing an object reference with a weak reference to that object. Since Java objects having only weak references become the target of garbage collection of JVM, unnecessary objects are pushed to be cleared by garbage collection.

Periodic clean-up approaches usually require the configuration parameter which specifies the limitation to trigger clean-up operation. However, such configuration parameter faces the risk of user's mis-configuration. If the limitation is not set within the available resource capacity, the software suffers from software aging and eventually fails due to the depletion of resources. Careful configuration of the limitation is particularly important in the execution environments that allow flexible resource allocation such as virtual machines as discussed in [5].

A question related to the resolution of aging-related bug is whether the time to fix the aging-related bug is longer than that for all the bug reports. We investigate the time to fix the bugs (TTFx) from the time when the bugs are initially reported to the time when the problems are resolved. Figure 1 shows the distributions of TTFx for all the bugs and the aging-related bugs in Hadoop MapReduce. The TTFx distribution for all the bugs is obtained from the set of bug reports in resolved status. As can been seen, most of the resolved bugs are fixed within 1000 days. The distribution of TTFx for aging-related bugs is obtained from the 13 aging-related bugs in resolved status (including one applying periodic clean-up as a permanent solution). Contrary to our expectation, most of aging-related bugs are fixed in a shorter time than the TTFx for all the bugs. Since there are many unresolved issues in Hadoop MapReduce, it may contain more complex type of bugs which take longer time to fix than the aging-related bugs. In addition, we cannot neglect the impact of the time to identify the root-cause of the software aging which may not be counted in the TTFx. Some short TTFx for aging-related bug reports are accounted for by the quality of the reports which describes the suspicious root-cause and further provides a suggested software patch.

The TTFx distributions in Cassandra are computed as shown in Figure 2. Compared to Hadoop MapReduce, reported bugs are fixed promptly as 65% of the bugs are resolved in ten days and more than 99% of the bugs are resolved in 300 days. In particular, all of the aging-related bugs are completely removed within 250 days. Looking at the distribution of TTFx for aging-related bugs, we can see the silent period from 20 days to 140 days where the bug-fix event is not frequent as for all the bugs. Basically the delays of the bug-fixes in this period are caused by missing status updates rather than the difficulties of the bug-fix. These bugs could have been updated as resolved status earlier. Therefore, with only from this observation, we can hardly conclude that the aging-related bugs have longer TTFx than the others.

TTFx distributions for the other three projects are not computed because the number of aging-related bugs is small. We focus on Hadoop MapReduce and Cassandra in the following subsection as well.

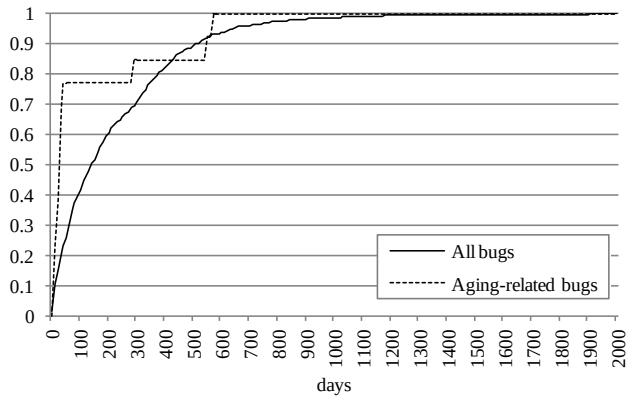

Figure 1. Distributions of time to fix the bugs in Hadoop MapReduce

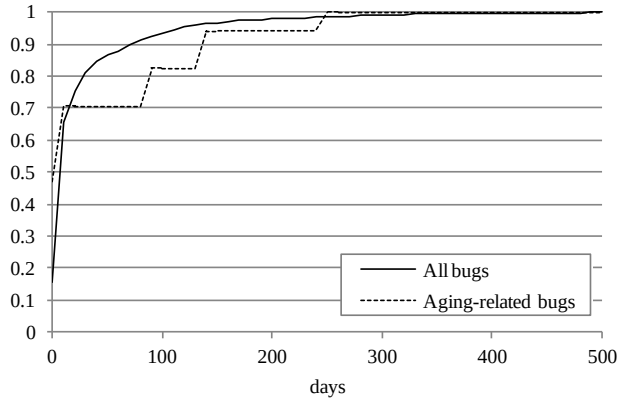

Figure 2. Distributions of time to fix the bugs in Cassandra

D. Trends of aging-related bugs

Finally, we investigate the trend of aging-related bugs by counting the number of bugs per month from the initial bug report. Figure 3 shows the bug curve for Hadoop MapReduce by plotting the accumulated number of bugs from April 2006 to January 2012. Although many users download and install the current version of Hadoop MapReduce, the software is not matured in terms of software reliability characterized by the bug curve. The recent sharp increasing trend of the number of bugs may be caused by the rapid increase of the number of users. As the software becomes popular, the number of users increases and hence the detection of software bugs is accelerated.

According to the trend of the number of bugs, the ratio of aging-related bugs is changed as shown in Figure 4. As can be seen, aging-related bugs constantly exist around at 1% (0.01) of the total number of bugs during the observed period. It is not appropriate to conclude that the aging-related bugs have an original trend over the trend of the total number of bugs. However, we confirm that aging-related bugs are contained in Hadoop MapReduce at a certain ratio of the total number of bugs in most of the development lifecycle.

The increasing trend of the number of bugs is also observed in Cassandra from March 2009 to January 2012 as shown in Figure 5.

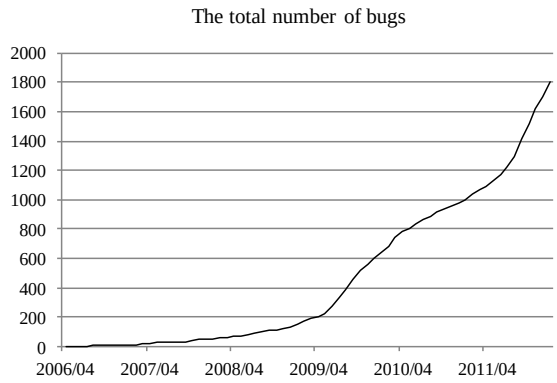

Figure 3. The bug curve for Hadoop MapReduce

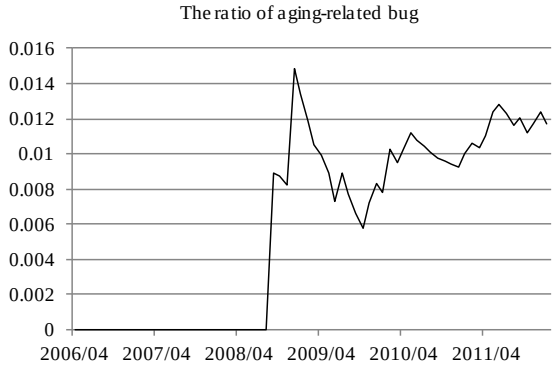

Figure 4. The ratio of aging-related bugs in Hadoop MapReduce

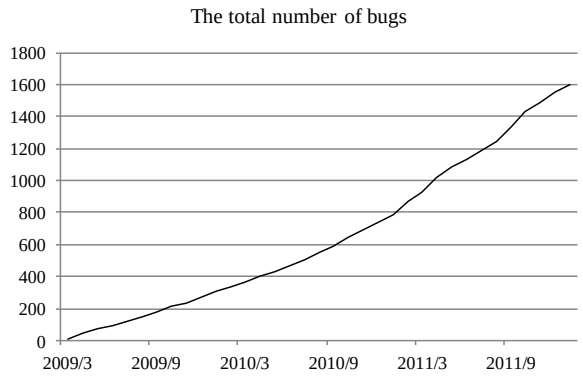

Figure 5. The bug curve for Cassandra

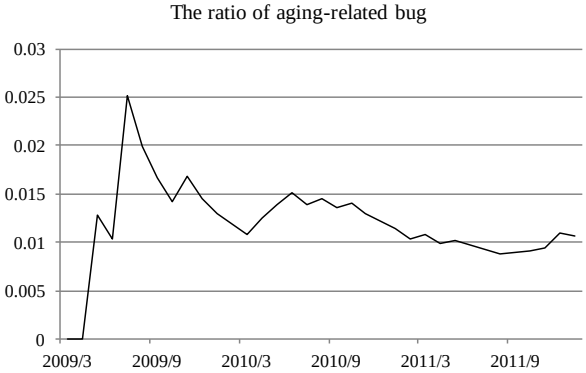

Figure 6. The ratio of aging-related bugs in Cassandra

Although the official release of Cassandra 1.0 was announced in October 2011, the number of bug reports increases constantly. The ratio of aging-related bugs is observed as shown in Figure 6. Similar to Hadoop MapReduce, aging-related bugs are found in the most ranges of the period by about 1 % of the total bugs. As some aging-related bugs are reported after the release of Cassandra 1.0, a number of users can confront the issues caused by aging-related bugs.

IV. DISCUSSION

The bug classification method used in our investigation may not perfectly cover all the latent aging-related bugs and it will bias the analysis. First, we intentionally include the unresolved reports in the scope of the investigation because remaining unresolved issues are quite common in OSS projects and are considered as the risk of the users as seen in Section III-C. However, the uncertainty of the classification of the unresolved issues might bring a misreading of the amount of aging-related bugs. Second, the chosen key-words for filtering the bug reports may not be enough to filter the aging-related reports. Our classification method relies on the description of the reports and hence we may overlook some aging-related bug reports due to the incomplete descriptions and/or the insufficient set of key words. We should remark that this is the first attempt to extract the aging-related bug reports from the OSS's bug tracking system and the method can be improved considering the potential biases discussed above.

Reproducibility of bugs is an important aspect of bug tracking process especially in OSS projects. We observe that the bug reports are broadly separated into two cases; i) the case where the fault in the source code is specified or ii) the case where the actual fault is unknown. In the former case, developers can easily fix the bug and commit the corresponding patch promptly. The reporter of the bug sometime prepares the patch at the same time as posting the report. In the latter case, however, the developers who try to fix the reported bug need to reproduce the bug in their own environment. The task to reproduce the problem is sometime cumbersome because it is not easy to create the exact same execution environment as that of the bug reporters. We find many bug reports which remain unresolved status or closed without fix due to the difficulty of reproducing the problem. The difficulty sometime is caused by the insufficient description about the reporter's system configuration. Fixing aging-related bugs also suffer from the limited reproducibility as they highly depend on the execution environment and require long time to confirm the problem.

To improve the reproducibility of aging-related bugs, it is important to characterize the aging phenomena associated with workloads. The framework to characterize workload-aging relationship [8], the techniques to understand the aging trends in a short period of time such as accelerated degradation tests [9] and accelerated life tests [10] are potentially useful for this purpose. These techniques are not designed for the tools available in OSS projects where a number of developers worldwide share the source code and

run the software in the individual environments. However, leveraging these techniques to provide a common testing tool and introducing the process to share the configuration and test results of the tool, the reproducibility of aging-related bugs might be greatly improved.

Although the terminology such as "software aging", "aging-related bug" and "software rejuvenation" have been used in many research literature, such terminology are seldom used in the software development community as we did not find them in the reports of aging-related bugs. This implies that there is a split among research works and software development in practice. Due to the gap in the terminology, the outcomes from research works may not be utilized effectively in practical software development projects. Developing a common tool for testing aging-related bugs in OSS development might help to overcome the gap.

V. Related Work

The most closely related work to the presented study is the bug investigation report of JPL/NASA projects [4]. The report categorizes 520 of unique software faults found in 18 missions into Bohrbug, non-aging-related Mandelbug, or aging-related bug and the proportions of the aging-related bugs over the software mission periods are investigated. The categorization is based on the textual description, while we resort to key word search for filtering only aging-related bugs and reading the description subsequently. The observed proportions of aging-related bug in cloud-oriented OSS projects (around at 0.1) are relatively smaller than 0.044 which is resulted from the JPL/NASA projects (23 software bugs are categorized into aging-related bugs from 520 bugs [4]). While their study focused on the reports only in the operational phase, the bug reports investigated in our study include the reports in the development phase as well. This might be a cause of the difference in the proportion of aging-related bugs. Besides the analysis of the proportion, we also study the distribution of the time to fix the aging-related bugs.

Studies on the bug reports of OSS projects have been also carried out for mobile OSes [6] and for several applications [7]. According to [7], 5-14% of the faults are only triggered by transient conditions, while they are not further categorized into aging-related bugs or others. The bug studies for mobile OSes also counts the number of transient or intermittent bugs by 2-10% [6] which might include aging-related bugs. In contrast to these works, our investigation focuses on the cloud-oriented OSS projects and shows the characteristics of aging-related bugs in detail.

Some of the aging-related bugs discussed in this paper have been already presented in the literature [11][12][13]. Kourai et al. reported the aging bugs in Xen hypervisor and proposed a fast rejuvenation technique to mitigate the software aging in VMM [11]. Aging behavior caused by aging-related bugs in Xen is also presented in [12]. Araujo et al. pointed out the aging issue in Eucalyptus and apply periodic rejuvenation to counter the software aging [13]. Although our investigation somewhat overlaps these reports, we cover more wide-range of aging-related bugs and show the statistics and the trends.

VI. Summury

Aging-related bugs in five OSS projects for cloud computing system are investigated from the public bug reports on the bug tracking systems. Aging-related bugs are found in all the five OSS projects and the ratios of aging-related bugs range from 0.4% to 1.4%. The ratio tends to be constant in the software development lifecycle. From the observations of the resource categories affected by aging, file descriptors are the major causes of the software aging. Some aging-related bugs affect the external resources of the software and thus they are difficult to be detected in the software unit test. The tool assistance for reproducing the aging phenomena in a replicated site might be helpful to locate and fix such complex type of aging-related bugs in OSS projects.

References

[1] M. Grottke, R. Matias Jr., and K. S. Trivedi, The fundamentals of software aging, In Proc. of the 1st Int'l Workshop on Software Aging and Rejuvenation (WoSAR2008), pp.1-6, 2008.

[2] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, Bigtable: a distributed storage system for structured data. In Proc. of Symposium on Operating System Design and Implementation (OSDI 2006), pp. 205-218, 2006.

[3] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus open-source cloud-computing system, UCSD Tech. Rep. 2008-10.

[4] M. Grottke, A. P. Nikora, and K. S. Trivedi, An empirical investigation of fault types in space mission system software, In Proc. of Int'l Conf. on Dependable Systems and Networks (DSN 2010), pp. 447-456, 2010.

[5] F. Machida, J. Xiang, K. Tadano and Y. Maeno, Software life-extension: a new countermeasure to software aging, accepted in the 23rd Int'l Symp. on Software Reliability Engineering (ISSRE2012).

[6] A. K. Maji, K. Hao, S. Sultana, and S. Bagchi, Characterizing failures in mobile OSes: a case study with Android and Symbian, In Proc. of Int'l Symp. Software Reliability Engineering (ISSRE2010), pp. 249-258, 2010.

[7] S. Chandra and P. M. Chen, Whither generic recovery from application faults? A fault study using open-source software, In Proc. of Int'l Conf. on Dependable Systems and Networks (DSN2000), pp. 97-106, 2000.

[8] A. Bovenzi, D. Cotroneo, R. Pietrantuono, S. Russo, Workload characterization for software aging analysis, In Proc. of Int'l Symp. on Software Reliability Engineering (ISSRE 2011), pp. 240-249, 2011.

[9] R. Matias Jr., Pedro A. Barbetta, K. S. Trivedi, and P. J. Freitas Filho, Accerelated degradation tests applied to software aging experiments, IEEE Transactions on Reliability, vol. 59, no. 1, pp. 102-114, 2010.

[10] R. Matias Jr., K. S. Trivedi, and P. R. M. Maciel, Using accelerated life tests to estimate time to software aging failure, In Proc. of Int'l Symp. Software Reliability Engineering (ISSRE2010), pp. 211-219, 2010.

[11] K. Kourai and S. Chiba, Fast software rejuvenation of virtual machine monitor, In IEEE Trans. on Dependable Sec. Comput. vol. 8, no. 6, pp. 839-851, 2011.

[12] F. Machida, J. Xiang, K. Tadano, and Y. Maeno, Combined server rejuvenation in a virtualized data center, In Proc. of Int'l Conf. on Autonomic Trusted Computing, 2012.

[13] J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias Jr., and K. S. Trivedi, Software rejuvenation in Eucalyptus cloud computing infrastructure: a method based on time series forecasting and multiple thresholds, In Proc. of the 3rd Int'l Workshop on Software Aging and Rejuvenation (WoSAR2011), 2011.