Software Rejuvenation based Fault Tolerance Scheme for Cloud Applications

Jing Liu, Jiantao Zhou
College of Computer Science
Inner Mongolia University
Hohhot, China
{liujing, cszjtao}@imu.edu.cn

Rajkumar Buyya
CLOUDS Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
rbuyya@unimelb.edu.au

***Abstract*—Cloud applications are typically composed of multiple cloud service components communicating with each other through web service interfaces, where each component fulfills specified functionalities. Lack of effective fault tolerance scheme is one of major obstacles for enhancing availability and efficiency of complex and aging cloud application systems. In this paper, we propose a holistic software rejuvenation based fault tolerance scheme for cloud applications, which contains three indispensible parts: adaptive failure detection, aging degree evaluation, and checkpoint with trace replay based component rejuvenation. Through a preliminary and qualitative evaluation, it shows that our new fault tolerance scheme brings promising improvement on the availability of cloud applications.**

Keywords—software rejuvenation; failure prediction; live VM migration; checkpoint; cloud computing

I. INTRODUCTION

Cloud computing is a large-scale and complex distributed computing paradigm where the configurable resources (servers, storage, network, data and software applications) are provided as multi-level services via virtualization technologies. A cloud application is typically developed as a distributed system that utilizes multiple cloud services that communicate with each other through web service interface. Every cloud service is also composed of several service components that fulfill specified functionalities. The cloud application framework is shown in Figure 1. Most of cloud applications deployed in the cloud environments have lots of complicated and non-deterministic behaviors and interconnections. During their long time running, the accumulation of software internal errors will very likely lead to software aging problems, where unpredictable failures, performance degradation, or eventual crash occurs [1]. So, how to enhance the availability of cloud application execution becomes a significant and desiderated research issue.

Lack of effective fault tolerance scheme is one of major obstacles for enhancing availability of aging cloud application systems [2]. Recently, more studies use software rejuvenation, a proactive fault tolerance approach, to counteract the software aging problem in cloud computing scenarios. Most of them propose rejuvenation methods towards software components in cloud infrastructure platforms, such as Eucalyptus, or specific hypervisor software. They pay more attention on predicting the aging error-prone state and optimal time to start rejuvenation

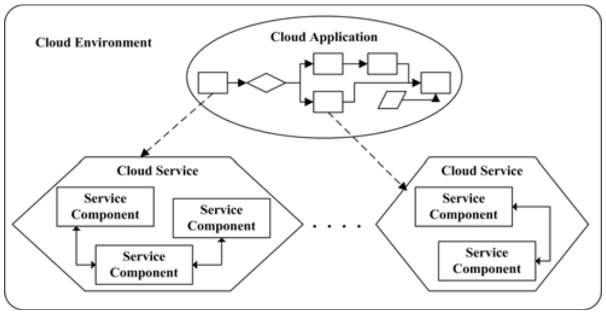

Fig. 1. Cloud Application Framework.

actions. Then they usually apply simple rejuvenation methods, such as rebooting, to bring the software from a failure-prone state to an aging-free state.

In this paper, we propose a novel and holistic software rejuvenation based fault tolerance scheme to counteract aging obstacles for cloud applications. Our major contribution comes from two inherently related aspects. First, an adaptive failure detection and aging degree evaluation approach is proposed to predict which cloud service components deserve foremost to be rejuvenated. Second, a component rejuvenation approach based on checkpoints with trace replay is proposed to guarantee the continuous running of cloud application systems. Through a preliminary and qualitative evaluation, it shows that our new fault tolerance scheme brings promising improvement on the availability of cloud applications.

II. SOFTWARE REJUVENATION IN CLOUDS

Software aging phenomenon, which results from the accumulation of internal errors occurring along with the software long time running, will likely lead to the performance degradation and progressive resource depletion, and eventually to the software crash [1]. As to cloud computing scenarios, cloud application systems require nearly uninterrupted running, but software aging is one of serious barriers to achieve such high availability. Software aging could occur in the hypervisor level or in the upper cloud application system level. To counteract this aging obstacle, software rejuvenation is utilized as a proactive fault tolerant method to reduce probability of future unpredictable cloud application outages [3].

Applying software rejuvenation technologies to the cloud computing scenario is a new and promising research frontier. Two major research issues are studied. One is to predict aging error-prone state and optimal time to schedule rejuvenation actions via measurement-based analysis or model-based (i.e., time-based) analysis. The other one is to perform rejuvenation methods to bring the software from a failure-prone state to an aging-free state, where the most intuitive and widely-adopted rejuvenation method is to terminate software transiently, clean up their internal runtime states (e.g. reinitializing internal data structures or garbage collection), and finally restart it. Typical related works are listed as follows. Bruneo et al. [3] propose a time based optimal rejuvenation policy, utilizing symbolic algebraic techniques. It improves the Virtual Machine Monitor (VMM) availability in variable workload conditions. Melo et al. [4] propose a pre-checking and live migration mechanisms based rejuvenation scheduling approach toward the VMM software, which brings a significant improvement on system availability. Araujo et al. [5] investigate the aging faults in a private cloud infrastructure due to the accumulation of memory leaks in key functional components of Eucalyptus platform. Langner et al. [6] detect software aging issues by investigating anomalous behaviors among different development versions of the same software, according to runtime metrics, such as CPU usage or number of running threads.

Compared with these studies, we focus on dealing with the software aging problems occurring in cloud application system, not in VMM-level systems, and we propose a proactive error detection approach and a software rejuvenation approach to enhance the high availability of aging cloud applications.

III. System Architecture

In our study we consider one specific cloud application system deployed in a private cloud environment. It composes of several interconnected cloud services with several functional service components respectively. We assume that Tight-coupling components (TCC) execute in the same virtual machine (VM), while loose-coupling components (LCC) execute in different virtual machines. Service components often communicate with each other via RPC over local high-speed network, and cloud services always communicate with each other via web services protocols over Internet. However, network caused failures are not considered in our study.

The system architecture is presented in Figure 2. Besides the running cloud application, four fault tolerance entities work cooperatively to carry out the holistic cloud application aging failure detection and rejuvenation process. Specifically, *Aging Failure Detector* (AFD) inspects CPU and memory usage, and communication status of service components periodically for detecting their aging failures. When a specific probable failure is detected, *Aging Degree Evaluator* (ADE) is used to record contexts of this failure, and then assign a fatal degree number to this failure event through an evaluation process, and finally insert this event into a failure event queue. The event queue is ordered according to the fatal degree number of every failure event automatically. That is, the failure that tends to result in more fatal consequence, such as service crash, will be firstly rejuvenated. Detailed algorithm in the aging failure detection and fatal degree evaluation approach is shown in section IV.

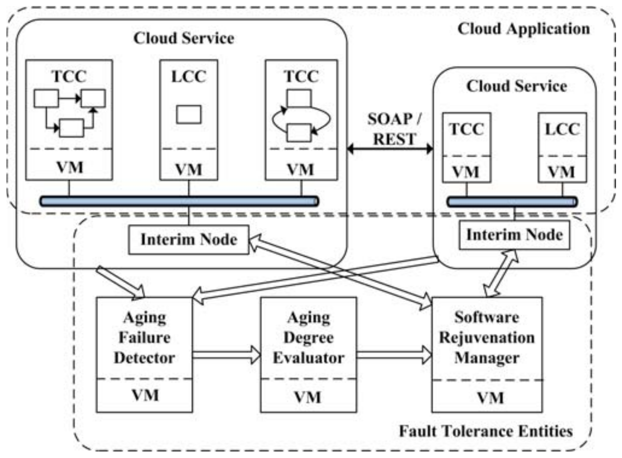

Fig. 2. System Architecture.

The practical execution of service component rejuvenation is controlled by the *Software Rejuvenation Manager* (SRM). In order to avoid re-executing aging service components from their initial states, we adopt the checkpoint and VM migration based service component rejuvenation to guarantee continuous running of cloud applications to a larger extent. That is, when a TCC or LCC is detected as most failure-prone, a checkpoint of its running state is made first and stored into the *Interim Node*, which locates in a separate physical machine in the same local network. Next, the VM that hosts failure-prone TCC or LCC is migrated to the Interim Node to keep service communications. When the migration is done, the original VM environment is rebooted to clear aging effects. Finally, on this new VM, the same TCC or LCC software re-executes based on the retrieved checkpoint data and subsequent behavior trace-log. After last trace-log is replayed, there is a consistent replica of that TCC or LCC both in this new VM and in the Interim Node, and then the migrated VM will terminate to run to indicate a successful rejuvenation procedure. Detailed algorithm in the service component rejuvenation approach is shown in section V.

We assume that all TCCs or LCCs, together with four functional entities are running on a VM, and besides these parts, there is a remote storage volume which is accessed by such VMs for easy live migration.

IV. Failure Detection and Evaluation

As shown in Figure 2, cloud service components execute in separated VMs. Some VMs may be active, some may be busy or heavy-loaded, and the others may be offline or even crashed. Meanwhile, communications among service component are not always available and stable. Thus, to design effective fault tolerant methods for mitigating the aging problem for cloud applications, we need at first to detect both VM and network performance status as certain credible metrics for accurately identifying which service components need rejuvenation.

Based on the failure detection methods shown in [7, 8], we propose an adaptive failure detection method. The CPU and memory usage of certain VMs, and the transmission delay of monitoring packets between service components and the AFD unit are chosen as the basic runtime metrics for indicating more comprehensively whether aging problems tend to cause service failures. That is, if a TCC or LCC is suffering from an aging problem, its host VM may have intuitionally high CPU usage and/or small free memory available data, as well as it is unable to send monitoring packets to the AFD at every interval regularly, or even fail to send packets.

In our study, we assume that in every VM that hosts a TCC or a LCC, a monitoring agent is running for periodically collecting runtime metrics of that VM. Specifically, in every sampling interval Δt_i, the agent collects the CPU usage C_i and the free memory available data FM_i, encapsulates C_i and FM_i data into a packet and sends it to the AFD. From the AFD point of view, it should achieve two related tasks.

1) *Computing Expected Arrival Time (EAT)*. EAT indicates the expected time of next packet arrival in regular case. If next metrics data packet arrives after EAT or totally does not arrive, it indicates some kinds of failures tend to appear. To improve computing accuracy, the computation of EAT utilizes actual arrival time of recent packets as the historical information in a sliding window, i.e., recent n packet $p_1, p_2, ..., p_n$, and $T_1, T_2, ..., T_n$ are their actual arrival time according to AFD's local clock. EAT_{k+1} ($k>n$) is the theoretically expected arrival time of next packet, and it is estimated as follows based on studies in [7].

$$EAT_{k+1} = \frac{1}{n}\left(\sum_{i=k-n+1}^{k}(T_i - \Delta t_i \times i)\right) + (k+1)\Delta t_{k+1}$$

Intuitively, it computes the average of packet transmission time of latest n packets and shifts forward the average for next p_{k+1}. Then, EAT for next packet could be added an offset to EAT_{k+1} to mitigate effects of possible normal packet delay.

2) *Software Aging Degree Evaluation*. Evaluation process in ADE utilizes the recorded running metrics from two aspects. One is whether next data packet arrives before EAT, after EAT, or totally lost. The other one is the degree of C_i and FM_i data from the data packet. According to the aging severity and when to perform rejuvenations, the aging degree of TCC or LCC is roughly divided into four levels. *Level 1* indicates the crucial case which is very likely to result in fatal consequence, such as service crash, and needs rejuvenation immediately. *Level 2* indicates the serious case which tends to result in serious consequence, e.g., long transmission delay or high performance loss, and needs rejuvenation as soon as possible. *Level 3* indicates the suspectable case which may cause unsatisfied consequence or false detection. It needs on-going monitoring and now rejuvenation is just a suggestion. *Level 4* indicates the normal case without any rejuvenation. It is assumed that for a cloud application which is not computing-intensive, the metric significance follows that EAT > M_i > C_i. Based on the Pareto principle (also known as the 80-20 rule), we present an exemplified aging degree decision in Table I according to different levels of three metrics.

In every interval of metric packet arrival, AFD will trigger ADE to detect proactively whether an aging failure is present. ADE records the context of the failure with a corresponding aging degree number, as a new failure event, and insert it into a centralized failure event queue. This queue is automatically ordered based on their aging degree number in ascending order. So, top events in the queue should be rejuvenated inevitably using the approaches discussed in the next section.

TABLE I. DECISION RELATION

Metrics	**Level**	**Decision Scenarios**								
EAT	***Lost***	Y								
	After		Y	Y	Y	Y				
	Before						Y	Y	Y	Y
M_i	***≤ 20%***		Y	Y			Y	Y		
	> 20%				Y	Y			Y	Y
C_i	***≥ 80%***		Y		Y		Y		Y	
	< 80%			Y		Y		Y		Y
Aging Level		**3***	**1**	**2**	**2**	**3**	**2**	**3**	**3**	**4**

* If packets are all lost in three sequential EAT interval, value **1** is set instead.

V. MIGRATION BASED SOFTWARE REJUVENATION

To avoid re-executing of a TCC or a LCC from its initial states, which costs more due to high downtime and complex synchronization behaviors, we integrate checkpoint technique with trace record and replay and live VM migration technique to develop a novel and holistic service rejuvenation based fault tolerant method to guarantee the continuous running of cloud applications to a larger extent. The intuitive idea is mentioned in section III with Figure 2. In this section, we present in detail three significant procedures during service rejuvenation. Figure 3 present the framework of rejuvenation workflows.

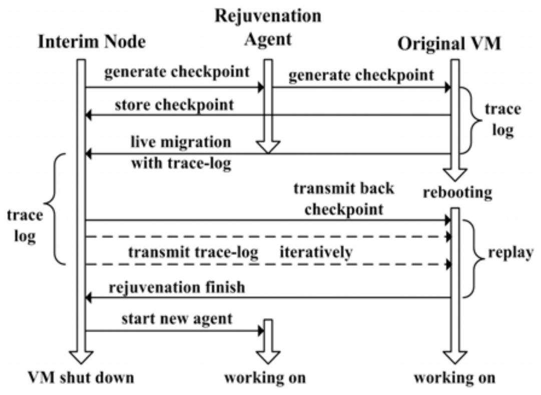

Fig. 3. Rejuvenation Workflows.

1) *Checkpoint Generation and Remote Storing*. When a TCC or a LCC is detected with aging failure-prone in level 1 or 2, the ADE will trigger the SRM to generate a rejuvenation starting message and send it to the interim node, which locates in the same LAN with the VM hosting aging TCC or LCC. It is also assumed that in every working VM, a rejuvenation agent is running for executing rejuvenation related actions. Thus, the interim node will establish a connection with the agent, and ask it to generate a checkpoint of the runtime status of that aging cloud service component, and finally transmit the checkpoint image to interim node for temporarily storing. The checkpoint data records variable values of running process, register values, runtime environment control information, and so on. It makes the aging TCC or LCC re-execute easily and reliably. From the checkpoint time on, all their subsequent external behaviors are recorded into a local trace-log file, containing functional input and communicating interactions. This trace-log file is used to guide the replay path of recorded behaviors accurately from the checkpoint time. Such replay executions using trace-log file do not need adding additional interface into original components.

2) Live Migration of the Original VM. After generating the checkpoint, the VM hosting those failure-prone TCC or LCC is migrated in pre-copy scheme [9] to the interim node, while the continuous communication of upper cloud service components are kept on as the migration occurs in the same LAN. It should be noted that during this live migration process, the trace-log file for recording subsequent behaviors of the aging TCC or LCC is still written by the rejuvenation agent.

3) Rejuvenation from Checkpoint. When migration is done, the original VM environment could be rebooted automatically controlled by the rejuvenation agent. That means a totally new VM environment (VM_s) for the cloud service running are well available. Next, the checkpoint image is transmitted from the interim node to VM_s to restore the TCC or LCC running from that checkpoint time. Then following iterations copy the trace-log file from the VM in interim node (VM_{in}) to VM_s, to replay the subsequent behaviors accurately. This iterative process is executed until trace-log files generated during the previous transfer round is reduced to a pre-defined threshold value. Till now, VM_{in} is suspended for running and the remaining trace-log file is transferred to the VM_s. Based on [10, 11], after last trace-log file is replayed, there is a consistent replica of cloud service component both in the VM_s and the VM_{in}. Together, all network traffic is redirected to VM_s destination, and VM_{in} is shut down. Until then, the rejuvenation of aging component is successful finally. Subsequently, the interim node will become available for next rejuvenation requests by clearing temporary checkpoint and trace-log data and delete the migrated VM.

VI. Discussion

Our rejuvenation based fault tolerance scheme effectively counteracts aging obstacles of cloud application systems. Its effectiveness could be qualitatively evaluated from both spatial and temporal aspects. First, it need one interim node for spatial redundancy in every cloud service group. Compared with the replication-based fault tolerance scheme which needs enough replication nodes for every service component, our method definitely has less spatial cost. Second, cooperation of the VM migration and trace-log based checkpoint replay techniques guarantee continuous running of cloud applications to a larger extent. Compared with the VM-rebooting based rejuvenation methods, it highly reduce the downtime of cloud application executions because the cloud applications do not need to re-execute from their initial states. Besides, our method can be directly used to handle multiple rejuvenations of independent VMs that host respective aging components at the same time.

Furthermore, our aging failure detection tends to be more comprehensive and more accurate because we consider both runtime metrics of VMs and communication liveness of cloud services as credible guidance to identify which cloud service components need rejuvenation imperatively.

VII. Conclusions and Future Work

In order to counteract software aging obstacles for service components in cloud applications, a novel and holistic software rejuvenation based fault tolerance scheme is well presented to improve their running availability. By a preliminary evaluation, we discuss that the adaptive failure detection and aging degree evaluation approach can accurately predict which cloud service components deserve foremost to be rejuvenated, and the checkpoint with trace-log replay based rejuvenation approach is a very promising fault tolerance mechanism choice to guarantee continuous running of cloud applications.

As to future works, we intend to improve the accuracy of the failure detection metrics and propose method to rule out the false positive scenarios. More importantly, we will conduct a more quantitative evaluation based on stochastic models to validate availability and performance of our method.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (No.61262017, No.61262082) and Inner Mongolia Key Engineering Lab of Cloud Computing and Service Software. The first author is currently visiting Melbourne CLOUDS Laboratory.

References

[1] D. Cotroneo, R. Natella, R. Pietrantuono and S. Russo, "Software Aging and Rejuvenation: Where we are and where we are going", in Proc. of the 3rd International Workshop on Software Aging and Rejuvenation (WoSAR 2011), pp. 1–6, Nov. 2011.

[2] R. Jhawar and V. Piuri, "Fault Tolerance and Resilience in Cloud Computing Environments", in Cyber Security and IT Infrastructure Protection, J. R. Vacca, Eds. Elesvier: USA, 2014, pp. 1–28.

[3] D. Bruneo, S. Distefano, F. Longo, A. Puliafito and M. Scarpa, "Workload-Based Softwar Rejuvenation in Cloud Systems", IEEE Transactions on Computers, vol. 62, no. 6, pp. 1072–1085, Jun. 2013.

[4] M. Melo, J. Araujo, R. Matos, J. Menezes and P. Maciel, " Comparative Analysis of Migration-Based Rejuvenation Schedules on Cloud Availability", in Proc. of the 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2013), pp. 4110–4115, Oct. 2013.

[5] J. Araujo, R. Matos, P. Maciel and R. Matias, "Software Aging Issues on the Eucalyptus Cloud Computing Infrastructure," in Proc. of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), pp. 1411–1416, Oct. 2011.

[6] F. Langner, A. Andrzejak, "Detecting Software Aging in a Cloud Computing Framework by Comparing Development Versions", in Proc. of the 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 896–899, May 2013.

[7] N. Xiong, A. V. Vasilakos, J. Wu, etc., "A Self-tuning Failure Detection Scheme for Cloud Computing Service", in Proc. of the IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS 2012), pp. 668–679, May 2012.

[8] I. P. Egwutuoha, S. Chen, D. Levy, B. Selic and R. Calvo, "Energy Efficient Fault Tolerance for High Performance Computing (HPC) in the Cloud", in Proc. of the IEEE 6th International Conference on Cloud Computing (CLOUD 2013), pp. 762–769, Jun. 2013.

[9] V. Medina and J. M. GarcIa, "A Survey of Migration Mechanisms of Virtual Machines", ACM Computing Surveys, Vol. 46, No. 3, pp. 30.1–30.33, Jan. 2014.

[10] H. Liu, H. Jin, X. Liao, C. Yu, and C. Xu, "Live Virtual Machine Migration via Asynchronous Replication and State Synchronization", IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No. 12, pp. 1986–1999, Dec. 2011.

[11] S. Di, Y. Robert, F. Vivien, etc., "Optimization of Cloud Task Processing with Checkpoint-Restart Mechanism", in Proc. of the 25th International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2013), pp.1–12, Nov. 2013.