International Journal of Database Theory and Application
Vol.9, No.6 (2016), pp. 21-32
http://dx.doi.org/10.14257/ijdta.2016.9.6.03

Group Rejuvenation: A New Software Rejuvenation Framework

[1]Li Su, Pengfei Chen, [2]Yong Qi and [3]Xinyi Li

[1,2,3] School of Electronics and Information Engineering, Xi'an Jiaotong University Xi'an,China
School of Mathematics and Statistics, Hainan National University HaiKou, China
suli2007@foxmail.com, qiy@xjtu.edu.cn

Abstract

With cloud computing continues to mature and widely used, the availability guarantee of cloud system becomes one of the key issues. In the cloud system based on virtualization, reliability of virtual machine is an important premise to guarantee the quality of service of cloud computing. Recent research shows, applications deployed inside the virtual machine will appear performance degradation even failure after long-term operation, namely "the aging phenomenon". Prediction, diagnosis and rejuvenation of the virtual machine is a relatively complete technical framework to guarantee reliability of the cloud computing system. In this paper, we propose a software rejuvenation framework based on group migration of virtual machine to guarantee reliability of the distributed system. First construct the dependency relationship of the virtual machines; then diagnosis the software aging of virtual machine; decide the optimal virtual machine set which need to migration. Compared with the traditional single computing nodes restart/ recovery mode, downtime of group rejuvenation can be 61.53% of downtime of traditional mode. In the cloud computing environment, group rejuvenation method can ensure the availability of cloud systems more effectively than the conventional rejuvenation method.

Keywords*: Software aging, Software rejuvenation, Group rejuvenation, dependency relationship, NARX*

1. Introduction

In order to meet the various needs of human beings or organizations in modern society, data center should greatly increasing its scale, and provide uninterrupted and high reliability service. Performance degradation, error rate increased and even a sudden hung/crash are bound to occur after the Ultra-large-scale data centers' long time running. To counteract this problem, Huang *et al.* [7] proposed a proactive recovery method called software rejuvenation which involves occasionally stopping the software application, removing the accrued error conditions and then restarting the application in a clean environment or intermediate state. Techniques of various rejuvenation methods have been proposed, including probabilistic model-driven recovery [2], recursive restartability [8], and recovery-oriented computing [9].

The simplest way is to reboot the VM directly. [2] uses Bayesian estimation and Markov decision theory to provide controllers that choose good, if not optimal, recovery actions according to a user-defined optimization criteria. The effect of the recovery actions, in turn, prove the accuracy of the fault localization. Such an iterative process, and ultimately accurately find the real root cause of the fault. The system will recover through some sequence of actions. [12] conducts a comparison between the current state of the platform and the optimal platform state, then make a choice among three options: Service Creation, Service Migration and Service Recovery. Its main purpose is optimal resource allocation. [8] constructing the "optimal" (lowest-MTTR) restart tree. But it focused on

ISSN: 2005-4270 IJDTA

reactive rather than proactive restarts. Most services in data center are sequential jobs. Traditional rejuvenation methods don't consider the dependent relationship of other nodes and mutual aging effect. They haven't compare the whole distributed system's efficiency.

This paper proposed a new rejuvenation strategy called Group rejuvenation. We mainly concern the applications in VMs. We know that one characteristic of VM is works without interruption. Our main objective is to guarantee the applications in VMs without interrupt. Every node in the network is not isolated, they have some relationships with each other. Any node's aging may lead to its neighbor nodes' performance decline; another case is that a node rejuvenate after its associated nodes' rejuvenate in a very short period of time. Then the availability of the sub-network is greatly reduced. The overall goal of our approach is to diagnose system problems using the output of the monitors and choose the optimal rejuvenation that are most likely to restore the system to a proper state at minimum cost. The basic idea is that build the network dependency graph based on service according to the actual network situation. Each node represents an element (a physical machine, a virtual machine, or a module, *etc.*), and the edge and arrow represent the service relationship between each other. Aging determination is by NARX-SPRT method. Combined with the dependency graph we can identify which nodes need to be rejuvenated. This can greatly save the total service disruption time, while reducing some nodes' (near the client side) rejuvenation times.

To achieve the proposed objective, we build an automatic, online performance diagnosis and rejuvenation framework. When the network established, the service dependency relationship builder subsequently constructs the causality graph automatically at service level. Monitor on each node collect the real-time data, then send all the data to Predicter to determine whether aging phenomenon exist. The analysis engin will be triggered when the Predicter find that node A is aging and may need to rejuvenate. Analysis engine passes the causality graph iteratively, and determine which nodes need to be rejuvenate. Finally implement all rejuvenate according to the determination result.

To measure the effectiveness of the method and the framework, we have conducted several evaluations include theoretical and experimental methods using several widely-used applications. Availability results show that group rejuvenation significantly outperforms single node rejuvenation method. It reduces the downtime of the whole system ranging from 0.6153 to 0.9780. We further show that the performance impact of group rejuvenation due to complicated relationship generation and aging diagnosis is also very small compared to the traditional rejuvenation method. Our evaluating results show that the constant ratio of internal and external causes can reduce 0.36 rejuvenation times of some nodes during one week experiments.

In summary, the contributions of this paper are:

- We provide a new causality graph building method. Using this causality graph, we can greatly reduce the judgment of aging and rejuvenation of nodes.
- We define the ratio of internal cause and external cause, P. This helps lower the frequency of node rejuvenation. While enhance the availability of the entire network.
- Group rejuvenation can reduce the unavailable time of the network, as it in a sense merged a number of rejuvenation time.
- The design, implementation and evaluation of the framework confirm the effectiveness of group rejuvenation.

In Section 2, we describe the framework. Section 3 introduce the framework in detail. Including the relationship generation, aging detection and diagnosis and group rejuvenation. Section 4 presents experiments using both theoretical and experimental methods to evaluate system availability. Section 5, we review the previous work in software aging diagnosis and rejuvenation.

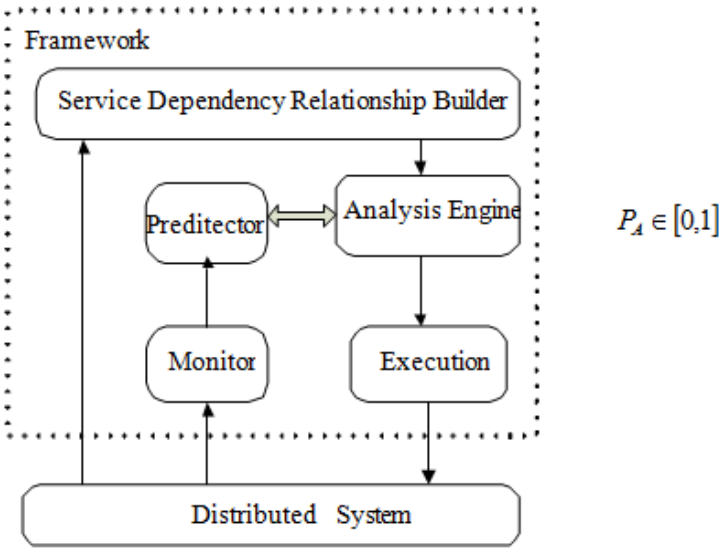


Figure 1. The Framework of Group Rejuvenation

2. Framework Overview

The core modules of the framework are a causality graph builder and an analysis engine. The causality graph builder can automatically construct a directed acyclic causality graph. The analysis engine is in charge of finding the nodes which most need to be rejuvenated.

Figure 1 presents an overview of the components interaction. In the target distributed system, this framework is deployed in each node and works in a distributed manner. The data collection module collect the node's resource status like CPU, Memory, number of threads or number of connections. These metrics are used by the predictor to detect the software aging phenomenon and estimate the service time to crash. It is integrated by a Machine Learning model(NARX[4]) trained to estimate the time to crash due to complex software aging phenomenon. The Analysis engine is triggered when the predictor module detect and predict that someone(nodes) will hung or crash follow by a window time. We define the causes of node aging consists of two parts, $R_I(x)$, the internal causes of aging on node(x) , and $R_E(x)$, the external causes of aging

On node(x) . P_A is the ratio of R_I and R_E. $P_A = \frac{R_I}{R_E}$, where

T is the threshold determines whether the node should be rejuvenated. If $P_A(x)$ is greater than T, node(x) should be labeled that it need to be rejuvenated. Then iteratively goes to the back end nodes along the paths in the service dependency graph. Calculate their P_A , and then decide whether they should be rejuvenated. Find all nodes (Y(y1,y2,...,yn)) that eligible for rejuvenation in this way. It uses breadth-first search algorithm and the search stops when $P_A \leq T$ and $Y=\phi$. Implement all the rejuvenation at the end of the algorithm. The simplest rejuvenation here is reboot. Migration is needed to guarantee that during the rejuvenation, any on-going or new request to the service is not missing. We assume that all failures were curable through rejuvenation of either a single node or a group of such node.

3. Framework Design

3.1. Aging Detection and Rejuvenation Time Prediction

The data collection module collects the node resource status from all nodes and the predictor detects the Time To Crash (TTC) of the node. The predictor is conducted by NARX model and SPRT method [4]. Firstly, use NARX model to training a normal state model of node based on priori data (all collected metrics of the node resource status). Then the threshold can be defined by SPRT. The predictor estimates the time to crash of the node based on the monitored run-time data. In [4], it is shown the effectiveness of this algorithm to estimate the time to crash of a web service under complex and undeterministic aging scenarios. We note that the predictor could be changed with new detection/estimation failures methods to maintain the effectiveness of the framework in new environments.

As we use group rejuvenation to rejuvenate the distribute system, a special window time should be considered in this framework. Suppose t1 is the time when the detector find node x aged. Then the analysis engine decides whether x should be rejuvenated (depend on PA(x)). The analysis engine also finds all nodes (Y(y1,y2,...,yn)) that eligible for rejuvenation after traversal the service dependency graph. The rejuvenation implemented after all judgments. We define C(c1,c2,...,cn) is the aged nodes which detected by the detector, but they don't eligible for rejuvenation. The performance of node C(c1,c2,...,cn) will picks up after the rejuvenation. While the window time should guarantee C(c1,c2,...,cn) will not hung or crash between t1 and the time their performance picks up.

3.2. Dependency Relationship Based on Service

Our method has the similar assumption and methodology to Orion [5] that is the traffic delay between dependent services often exhibits "typical" spikes that reflect the underlying delay for using or providing these services. But the primary differences are: a. Our method focuses on a limited set of applications which use TCP as their underlying transport protocol although it can be extended with extra effort. b. Our method relies on the new properties of the modern operating system such as network statistical tools and kprobe [6] used to probe the system call. c. We leverage traffic delay to determine the dependency directions rather than determine the dependency structure in the dependency graph which reduces the risk of wrong dependent relations and computational complexity (from $O(n^2)$ to $O(n)$). According to our observations, TCP protocol takes a dominated position in all the protocols leveraged by common applications like Mysql, Tomcat, *etc*. and almost all mainstream linux operating systems have integrated with network statistic tools such as netstat, tcpdump and kprobe. Therefore our method can be used in most of distributed systems running on linux operating system.

Different from Orion, we use a two-tuple (ip, service name) instead of three-tuple (ip, port, proto) to denote a service considering that a service may utilize multiple ports. For example, in a three tiered system, a web server may access the application server through a random port. So if we set port as an attribute of a unique service, the dependency graph becomes dramatically huge even though the requests are all issued by the same service. In a distributed system, ip denotes a unique host and service name denotes a unique service running in the host. We follow the definition of service dependency in Orion system that is: if service A requires service B to satisfy certain requests from its clients, A→B. For instance, a web service need to fetch the content from a database kept by a database service, so we say the web service depends on the database service. And in this paper, we are also only concerned with client-server applications which are dominant in modern applications.

The first step of our method is to use the connection information to construct a skeleton of the service dependency graph. Executing an off-the-shelf tool, netstat, in a host, we get

a list of all the connection information including protocol, source, destination and connection state. We extract the source and destination information which connected by TCP protocol. Each of the connection is organized in the format source_ip: port→destination_ip: port, we call it a channel. The channel is very close to the service dependency pair except one point: it dose not contain a service name but a port. The following trivial work is to map a service port to a service name. To get the service name with respect to a local port is easy by querying the port information. But to a remote port, the Relationship Builder in the local host need to send a query to the Relationship Builder in the remote host. After the mapping procedure, the skeleton of a service dependency graph in a local host is established. But one problem stays unresolved. The transportation between client and server is bidirectional which means we may get an opposite service dependency when observing in different hosts. For instance, when observing in host 192.168.1.117, we get the service connection (192.168.1.117, httpd)→(192.168.1.115, tomcat); but in host 192.168.1.115, we get (192.168.1.115, tomcat)→(192.168.1.117, httpd). To address this issue, we use and improve the traffic delay assumption mentioned above. In a client-server structure, a common observation is the packets sent by the server change with the ones sent by the client. Therefore we use the lag correlation of the send traffic between two services to distinguish the dependency direction. To get the send traffic of a specific service, we count the number of packets transmitted by a specific process through probing the function netdev.transmit triggered when the network device wants to transmit a buffer. Assuming X is the send traffic of service A, Y is the send traffic of service B, the lag correlation between X and Y is defined as

$$\rho_{XY}(k) = \frac{\sum_{t=0}^{N-1} Y(t)X(t-k)}{\sqrt{\sum_{t=0}^{N-1} X^2(t)\sum_{t=0}^{N-1} Y^2(t)}} \qquad k \in Z$$

Where k is the lag value, it can be positive and negative. In our system we set the absolute value of k at 30 which can capture almost all the traffic delay. Our objective is to find a best k which maximize ρ_{XY} (k) namely

$$k^* = \{\max(\rho_{XY}(k), k \in [-30,30]\}$$

According to the sign of k*, the dependency direction is determined. If k*> 0, A→B; else B→A. Figure 2 demonstrates the lag correlation between httpd and memcached applications. The result shows that k*= 4 implies httpd→Memcached confirmed in reality.

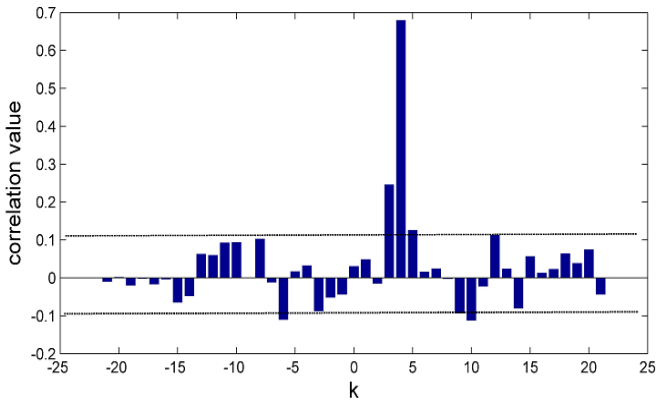


Figure 2. The Lag Correlation of Send Traffic Between Httpd and Memcached

3.3. Group Node Selection

The analysis engine is in charge of traversing and finding "all" nodes that eligible for rejuvenation according to the service dependency graph generated by the service dependency graph builder.

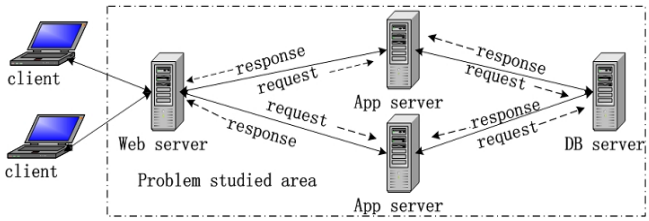


Figure 3. The Topological Relationship

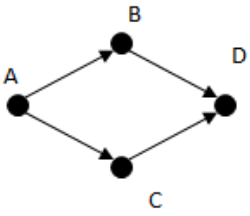


Figure 4. Dependency Relationship

Figure 4 is a service dependency relationship diagram of the net shows in Figure 3. Simple it is, it contains all the essential relationship, *e.g.*, A→B is father-child relationship, while A→B and A→C is And-Or relationship. ServiceTimeA is the total time spend on the request start from A till it end on A. LocalTimeA is the time spend on the request just on node A. Assumes that the network transmission delay can be ignored. The following relations are obtained：

$$\text{ServiceTimeA} = \text{LocalTimeA} + \max(\text{ServiceTimeB}, \text{ServiceTimeC}) \quad (3)$$

$$P_A = \frac{LocalTime_A}{ServiceTime_A - LocalTime_A} \quad (4)$$

We propose a new metric to measure the impact that internal and external causes of aging on the node: Tcp Request Latency (abbreviated as TRL). TRL is obtained by measuring the latency between the last inbound packet (*i.e*. request) and the first outbound packet (*i.e*. response) passing through a specific port. We substitute TRL_A for $ServiceTime_A$. According to the formula 3 and dependencies between nodes, we can get the LocalTime of any node. Although this metric is simple, it works well in our framework. According to our observations, most of applications use TCP protocol as their fundamental transport protocol like Mysql, Httpd, *etc*. Hence TRL can be adopted to represent the impact of aging on nodes.

We assume that any rejuvenation time is fixed in a network, and its value is t_m. Consider this situation, node A and B have the relationship A→B. If the analysis engine decides that A is aging but need not to rejuvenate, while B need to rejuvenate. We define t' is the time from B's rejuvenation end to A's performance returns to a normal state.

$$t = \Phi(d1,d2,...,dn) \quad (5)$$

d is the aging degree of each node resource metric collected by data collection module, while t is the time from now till x hung or crash. Φ is the relation function. The predictor models this relation function by NARX model.

When the predictor finds that node x aging at time t1, the predictor subsequently calculate the time t through d, P_A and formula 4. Here, t means the node x will hung or crash at t+t1 just because its internal causes of aging. If $t< t_m + t'$, it indicates that node x need to be rejuvenated. Otherwise we find the children of x, and use the same method to decide whether they should to be rejuvenated. The policy also keeps track of past restarts to prevent infinite restarts (two adjacent restart time is shorter than a certain time window) of "hard" failures.

The purpose is to achieve optimum rejuvenation of the entire network. We don't consider the upper bound of the number of nodes in the rejuvenation. This involves the infrastructure, the bandwidth, *etc.* Users can set the upper bound of the framework according to different situations. This paper uses a threshold value of the maximum number of nodes to determine the cluster regeneration according to the hardware conditions. According to the different environment, we will use appropriate optimization algorithm to calculate the number of nodes to achieve optimal regeneration in future work.

4. Experimental Evaluation

In this section we present the results obtained after evaluating the models proposed. To collect the process and operating system performance metrics, we use some off-the-shelf tools such as collectl; to collect other metrics, we develop several tools from the scratch, such as tcpdep to generate the service dependency relationship. In the following, we will give the details of our experimental methodology and evaluation results in TPC-W benchmark.

4.1. Evaluation Methodology and Effectiveness

We use both theoretical and experimental methods to evaluate system availability. Here, the effectiveness of the group rejuvenation is represented by the ratio of downtime of the whole system whether to use group rejuvenation.

Node A and B are two sequential nodes in a service, as in Figure 4. Suppose both of their aging curves are conform to the weibull distribution, which is the theoretical basis of reliability analysis and life test. Then we define the function of aging degree of any nodes as follows:

$$A(x) = 1 - e^{-(x/\lambda)^k} \tag{6}$$

where $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter of the distribution.

$$T_{D'} = T_{ABD} \times \sum_{0}^{t} [\Phi(A_A(x_A) - D_A) \times \Phi(A_B(x_B) - D_B) \times \Phi(T_{AB} - (x_B - x_A))] \tag{7}$$

$$T_{D'A} = T_{AD} \times \sum_{0}^{t} [\Phi(A_A(x_A) - D_A) \times \Phi((x_B - x_A) - T_{AB})] \tag{8}$$

$$T_{DA} = T_{AD} \times \sum_{0}^{t} \Phi(A_A(t) - D_A) \tag{9}$$

$T_{D'}$ is the total downtime of group rejuvenation (both A and B are aging in a small window of time) in a fixed period of time T_s. $T_{D'A}$ is the total downtime of node A in T_s eliminate the times it rejuvenated in group rejuvenation. T_{DA} is the total downtime of node A in T_s. T_{ABD} is the average downtime of both A and B rejuvenate. T_{AD} is the average downtime of node A. D_A and D_B are aging threshold of A and B respectively. T_{AB} is the

node rejuvenation interval threshold. Φ is a piecewise function. P_r, the ratio of downtime of the whole system whether to use group rejuvenation is defined as follows:

$$\Pr = \frac{T_{group-re}}{T} = \frac{T_{D'} + T_{D'A} + T_{D'B}}{T_{AD} + T_{BD}} \tag{10}$$

A change in the scale parameter λ has no effect on the ratio. When it increases to a certain degree, the ratio P became NaN. As a result, we change the time unit in the test five and six in Table 1. The ratio P has the same trend with shape parameter k. The result shows that group rejuvenation do reduce the total downtime of the system. Its effectiveness is mainly depends on the aging model.

Table 1. Different Weibull Parameters and the Ratio Results

	λ_A	k_A	λ_B	k_B	P
1	1	1	1	5	0.3577
2	1	0.5	1	5	0.1955
3	1	0.2	1	5	0.0220
4	3	0.2	1	5	0.0220
5	10	1	15	1.5	0.9992
6	100	1	1000	1.5	0.9989

The sample interval in all the data collection tools is 1 second in the experimental method. The framework is only evaluated in a controlled distributed system. But we believe it works well in a real system without exceptions. The controlled system contains five physical server machines hosting the benchmarks and four client machines generating the workload. Each physical server machine has a 8-core Xeon 2.1 GHZ CPU and 16GB memory and is virtualized into five VM instances including domain 0 by KVM. Each VM has two vcpu and 2GB memory and runs a 64-bit CentOS 6.2. TPC-W is a transaction processing benchmark which is used to emulate online book shopping. In our controlled environment, we employ Apache Httpd, Apache Tomcat and Mysql as the web, servlet application and database service respectively and these services run in dedicated VM instances. We adopt Siege [1] to generate the HTTP requests randomly. To mimic the real performance problems, we inject faults in the benchmark. For the software aging, we inject MemLeak: a memory-bound application continually consumes memory of the application server. The fault mentioned above will be repeated for more than 20 times and last 5 minutes. To get the ground truth, we will log the fault injection time. We adopt collectl to collect data from client side. We can see the general trend of the service by these data. Figure 5 shows the context switch of the client for one hour with group rejuvenation. To get more precise downtime ratio, we adopt Jmeter to access to the client all the time. Collect service parameters such as response time to monitor the whole downtime of the system during a week. The ratio of downtime of the whole system whether to use group rejuvenation is 0.8322 for one day. The ratio up to 0.6153 at best in our experiment.

We record the restart time of each node for one week experiments with group rejuvenation and without group rejuvenation. We inject the same faults. The results shows that single node rejuvenation times has no much change in the first two days. Some nodes' rejuvenation times reduce 36 percent in a week time at best.

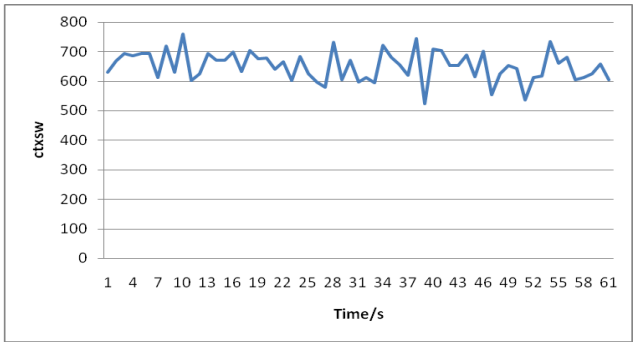


Figure 5. Context Switch of the Client

4.2. Discussion

The simplest decision of the rejuvenation group is to rejuvenate all the possible node once they were found aging. This method also improves availability of the system compared to the normal rejuvenation(*e.g.* just reboot any node once they are aging). And its diagnosis cost reduced. The ratio of the internal and external causes is mainly used to reduce some node's rejuvenation times. Especially when the rejuvenation time of some subsystems is too long. In the experiment, we use probability analysis of the data to generate the threshold of the ratio of the internal and external causes. The rate of aging greatly affects the ration P_A. We assume P_A is constant in the experiments.

Considering the hardware of the system, the rejuvenation time should be different every time, especially after the machine work for a long time. Figure 6 shows one node's rejuvenation times in 25 minutes. We assume the rejuvenation time of each node is constant in the experiments. We select the mean value of the time after the node rejuvenates 50 times.

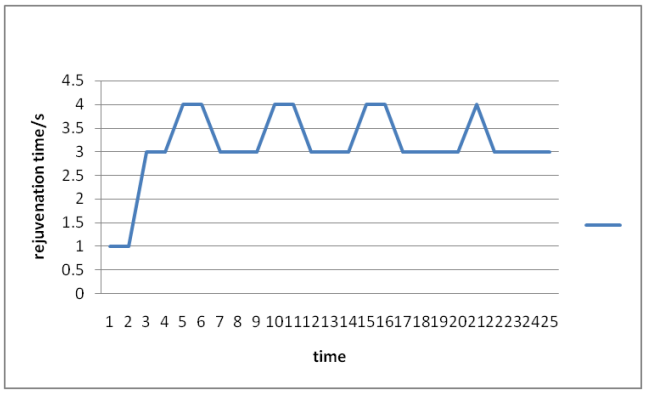


Figure 6. Rejuvenation Time

5. Related Work

The objective of aging rejuvenation or system recovery is to improve the availability of the system. The work is mainly divided into two parts, the first is the aging diagnosis, followed by regeneration choice. Software aging has been studied in the UNIX operating system [3] and the Apache Web server [11]. [3] design and implement an SNMP based, distributed monitoring tool used to collect operating system resource usage and system activity data at regular intervals, then apply statistical trend detection techniques to this data to detect/validate the existence of aging. The metric "Estimated time to exhaustion" they proposed is similar to t in formulator (4) in 3.3. [11] collect and log data on several system resource usage and

activity parameters on a web server while subjecting it to an artificial workload. Time-series ARMA models are then constructed from the data to detect aging and estimate resource exhaustion times. Recently, server consolidation using virtual machines (VMs) is being widely carried out. Many VMs are running on top of a VMM. Since a VMM is long-running software and is not rebooted frequently, the influences due to software aging accumulate more easily than the other components. [10] just uses time-based rejuvenation. [12] is conducted by M5P Machine Learning Algorithm. In this framework, we use NARX and SPRT method to diagnosis aging, which presents promising results in terms of accuracy prediction in software aging scenarios [4].

[10] proposed a technique for fast rejuvenation of VMMs called the warm-VM reboot. It enables only a VMM to be rebooted by using the on-memory suspend/resume mechanism and the quick reload mechanism. To achieve transparent software rejuvenation, [12] presented a self-reconfigurable framework which contains three conducts, Service Creation, Service Migration and Service Recovery. The approach is based on a mathematical programming model plus a machine learning module. This paper proposed a new rejuvenation decision based on dependency relationship and the parameter PA. It conducted using reboot or live-migration. The differences is that the number of VM rejuvenation is different each time. We call this group rejuvenation which can significantly improve the system availability and reduce the number of the VM rejuvenation in a very long time running.

A recursively restartable system [8] also uses restart groups(subtrees in a restart tree). In this reactive restarts, its restart tree is a hierarchy of restartable components, in which nodes are highly fault-isolated and a restart at a node will restart the entire corresponding subtree. This method has the same assumption with [9] that each component is failure-isolated.

Bug are everywhere. So bug diagnosis techniques are needed. Fault localization is a key component of bug diagnosis. Casual relationship is often used to find casual paths or root causes. [13] suggests the failure inducing chain (FIC) should ideally be computed by comparing the failing run with the run of the corrected program with the same input. While for practical debugging, corrected versions are not always available. [2] combines diagnosis and recovery actions into an iterative process that can accurately locate faults. Meanwhile recover a system through some sequence of actions. If the faults hypothesis and the recovery are invalid, some iteration steps will greatly increase the system downtime. This paper uses traffic delay and the lag correlation of the send traffic to construct the service dependency graph. Integrated with the ratio of internal causes of aging and external causes of aging, a coarse-grained root cause might be found in future work.

6. Conclusion

This paper mainly proposed a new rejuvenation decision based on service relationship, group rejuvenation. Construct the service dependency relationship is important in this method. As the aging estimation and group decision are mainly depend on the relationship. Using a machine learning method and SPRT to detect aging. Then combine the ratio of internal and external causes and the service dependency relationship to decide the rejuvenation group.

We observed in our experiments that group rejuvenation do reduce the total downtime of the system. The downtime in group rejuvenation account for only 2.2 percent of normal rejuvenation sometimes. It can also reduce the rejuvenation times

of some node based on the ratio of internal and external causes during long time experiments.

Acknowledgements

This work was partially supported by the Hainan Province Natural Science Fund: No. 20156233, and 20151003.

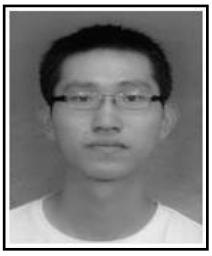
References

[1] "Siege", [Online]. Available: http://www.joedog.org/siege-home/
[2] K. Joshi, M. Hiltunen, W. Sanders and R. Schlichting, "Probabilistic model-driven recovery in distributed systems", IEEE Transactions on Dependable and Secure Computing, **(2010)**.
[3] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, "A Methodology for Detection and Estimation of Software Aging", Proceeding Ninth Int'l Symp. Software Reliability Eng., **(1998)**, pp. 283-292.
[4] S. Li and Q. Yong, "Software aging detection based on NARX model", Web Information Systems and Applications Conference. Los Alamitos, CA: IEEE Computer Society, vol. 11, **(2012)**, pp. 105-110.
[5] X. Chen, M. Zhang, Z. M. Mao and P. Bahl, "Automating network application dependency discovery: Experiences, limitations, and new solutions", In USENIX Symposium on Operating Systems Design and Implementations (OSDI), vol. 8, **(2008)**, pp. 117-130.
[6] R. Krishnakumar, "Kernel korner: kprobes-a kernel debugger", Linux Journal, vol. 2005, no. 133, **(2005)**, pp. 11.
[7] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software Rejuvenation: Analysis, Module and Applications", Proceeding 25th Int'l Symp. Fault-Tolerant Computing, **(1995)**, pp. 381-391.
[8] G. Candea, J. Cutler, A. Fox, R. Doshi, P. Gang, and R. Gowda, "Reducing recovery time in a small recursively restartable system", in Proceeding The International Conference on Dependable Systems and Networks(DSN), **(2002)**, pp. 605-614.
[9] D. Oppenheimer, A. Brown, J. Beck, D. Hettena, J. Kuroda, N. Treuhaft, D. Patterson, and K. Yelick, "Roc-1: Hardware support for recovery-oriented computing", IEEE Trans. on Computers, vol. 51, no. 2, **(2002)**, pp. 100-107.
[10] K. Kourai and S. Chiba, "A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines", Proceeding 37th Ann. IEEE/IFIP Int'l Conf. Dependable Systems and Networks, **(2007)**, pp. 245-254.
[11] L. Li, K. Vaidyanathan, and K. Trivedi, "An Approach for Estimation of Software Aging in a Web Server", Proceeding Int'l Symp. Empirical Software Eng., **(2002)**, pp. 91-100.
[12] J. Alonso, I. Goiri, J. Guitart, R. Gavalda and J. Torres, "Optimal resource allocation in a virtualized software aging platform with software rejuvenation", In Proceeding of the 22nd Int'l Symp. on Software Reliability Engineering (ISSRE 2011), **(2011)**, pp. 250-259.
[13] W. N. Sumner and X. Zhang, "Automatic failure inducing chain computation through aligned execution comparison", In Technical Report 08-023, Purdue University, **(2008)**.

Authors

LiSu, received the B.S. degrees in Electrical Engineering from North China institute of technology and M.S. degrees in Electrical Engineering from China University of Geosciences in 2004 and 2007, respectively. She now studying for a doctor's degree in Electronic and Information Engineering from Xi'an JiaoTong University.

YongQi, received the Ph.D. degree in computer science and technology from Xi'an Jiaotong University, Xi'an, China, in 2001. He is a full professor in the Department of Computer Science and Technology, Xi'an Jiaotong University. His research interests include sensor networks, operating system, distributed middleware, and services computing.



PengfeiChen, received the B.S. degree in computer science and engineering from Xi'an Jiaotong University, Xi'an, China, and is currently pursuing the Ph.D. degree in Xi'an Jiaotong University. His research interests include software availability, fault diagnosis and maintenance scheduling.

XinyiLi, received the B.S. degree in computer science and engineering from Xi'an Jiaotong University, Xi'an, China, and is currently pursuing the Ph.D. degree in Xi'an Jiaotong University. Her research interests include software availability, fault diagnosis and maintenance scheduling.

