International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 4; Issue: 1; January -2015

Testing Software rejuvenation policy for time and load balancing scheme in complex system using deterministic-NDST approach

Nagaraj G Cholli [1], Khalid Amin Shiekh [2], Dr. G N Srinivasan [3]

[1,2,3] Department of Information Science & Engineering,
R. V. College of Engineering,
Karnataka, India

Abstract : Software rejuvenation has become a new horizon for increasing the system reliability and availability in a long run. One of the concept in software rejuvenation policy involves rebooting the system by suspending the disk and capturing the images of processes currently running in the system. In our rejuvenation policy we make use an intelligent time and load algorithm for deciding the optimal rejuvenation period. Since at any given time there can be n number of processes running in the system with different physical memory utilizations and variable workloads running, it becomes very vital to thoroughly test the system with highly chaotic and disruptive workloads. For this we use some non-traditional form of testing which we call non deterministic system testing (NDST) to test different features and conditions of system. There are several approaches to testing complex systems using test methodologies. Certain tests sometimes fail and sometimes pass, without any deliberate or intentional change to product code, test script or environment. Those tests that result in such unpredictable and non-repeatable results can be classified as non-deterministic tests. The idea here is to first rejuvenate the system using reboot concept as it captures system images at kernel level and rejuvenate the system based on time factor or memory factor as needed using intelligent time and load algorithm. Making use of NDST suite, which creates the random test cases to exercise the components or features of any rejuvenated system, used to stress the complex system highly disruptive workloads to find race conditions and to check whether all the processes or applications are back into same form as they were before rejuvenation. But there is also a problem with NDST called random uncertainty. Non-deterministic tests can throw a spanner at months of testing done by turning things red all of a sudden. Since there may be many workloads which run in parallel and due to this the randomness happens and if there is some panic detected in the kernel or some bug found there will very little chances to reproduce the same bug . So the main idea is to make non deterministic testing more reliable and accurate by collecting workload tools which are executed in original run and reproducing the same sequence of commands and then re-executing the same run with the reproduced or generated sequence. This testing concept which uses reproduced sequence is called as D-NDST (deterministic-NDST).

Keywords: Software rejuvenation; OS warm reboot; non deterministic system testing

1- INTRODUCTION

Software rejuvenation is a process or act of gracefully terminating a running application and restarting it again to prevent aging [3]. In OS warm reboot process, before rejuvenating the kernel state is saved, including all applications or processes running on the kernel. Saving the kernel state is done by creating a complete image of kernel. Operating System reboot process is divided in two stages. Suspend & Resume. In Suspend stage kernel is called to create a snapshot of current system state later snapshot data is written to disk, finally system is rebooted. In Resume stage, when the system is turned on, grub loader runs from init before mounting any partitions, later all the data of snapshot is read from disk and loaded to kernel, kernel restores the image and thus system runs from same state where it was suspended.

When the system is rejuvenated it needs to be tested fully to check whether all processes and applications are restored to the original state and load balancing and memory consumption is not beyond expected value. So we need to use non deterministic testing for this. Deterministic Linear time has less strength compared to non-deterministic linear time [3][4]. Any discrepancy in the kernel such as resource monitoring process crashes, then to debug the process entire scenario need to be recreated which involves reproducing the last executed sequences and running it again with main job to hit the discrepancy in the kernel. Same mechanism can be used to test rejuvenating systems in an inter-networked environment or cluster. In networked systems the main concerns involve effectively selecting the optimal rejuvenation

period, collecting successfully executed workloads using non-deterministic testing ,executing the reproduced sequence on the same set of nodes in a cluster.

The remnant of paper is organized as follows: we discuss software rejuvenation technique review with warm reboot, as reported in Section 2. Then, in Section 3 we deal with overview of NDST and its approaches that implements D-NDST. Section 4 operatively describes the experimental analysis and result. Concluding remarks and future work are dealt with in Section 5.

2-SOFTWARE REJUVENATION TECHNIQUE REVIEW

Software rejuvenation techniques depends on policy defined considering variable workload and optimize rejuvenation time, rejuvenation time is calculated depending on variable workload which is given by system. System periodically checks the workload and update to rejuvenation manager.

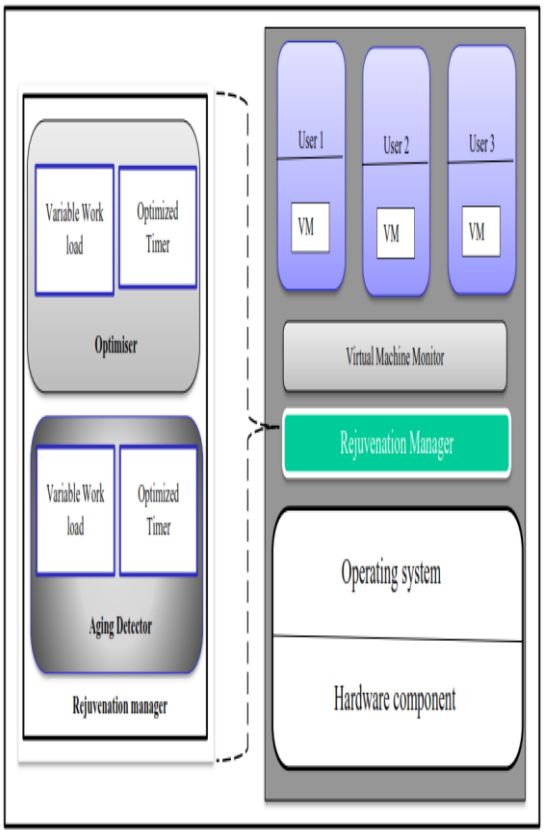

Figure 1 System architecture used for software rejuvenation in complex system

Figure shows the architecture of Rejuvenation manager, it is consists of aging detector which detects the software aging point and optimizer to optimize the timer value for a point of rejuvenation. Aging detector and optimizer has two components namely variable workload and timer policy which perform their defined function respectively. Aging detector obtains the value provided by rejuvenation manager periodically and checks for the need for change in rejuvenation time depending on workload and this is updated for rejuvenation manager, if there is need for change in rejuvenation time then rejuvenation manager allows optimizer to change the time, the function of optimizer is to optimize the time depending the values provided by aging detector based on workloads.

2-1 WARM REBOOT PROCESS

The functioning of Warm reboot process is described in the following flow diagram.

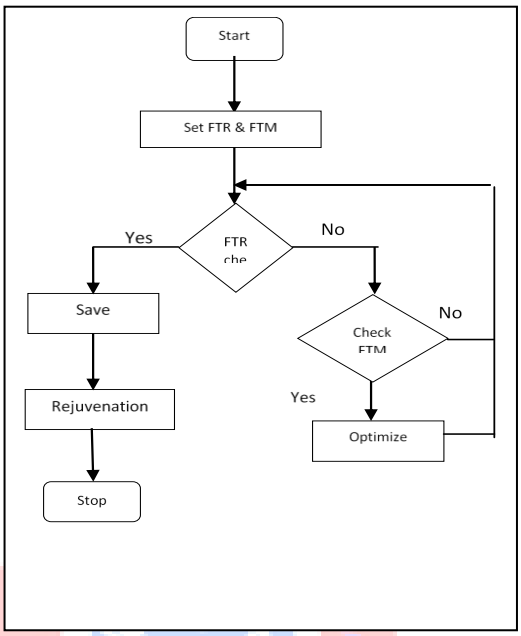

Figure 2 warm reboot process

The figure 2 describes working of Warm reboot process as per the software rejuvenation policy defined for **intelligent time and load algorithm**. First the predetermined rejuvenation time also called as FTR (fixed time for rejuvenation) and threshold value of memory called as FTM (fixed threshold memory) is set. By comparing system time with rejuvenation time, kernel state is saved if time value is equal and system is rejuvenated. If time is not equal then memory usage is compared with threshold memory value in block check FFM (Fixed Free Memory). In Warm reboot module availability value when compared to normal reboot module is high, because here complete kernel is saved as image and stored on hard disk, after reboot grub loader extract this image and kernel image will be loaded. Hence providing loss-less data and interruption free applications running even after reboot.

2-2 SAVING STATE OF COMPLEX SYSTEMS

With the help of S2disk program, before rejuvenating cluster of complex systems, all system processes need to be saved in the form of images in a non-corrupted form. S2disk program will save the state of the whole system to the disk before shutting down the system. After restarting the system S2disk program will restore the exact system state**.** ***initramfs-tool***s helps to initiate S2disk program at Debian (TM) kernel package. Once the all system kernel images are captured using s2disk program, the complex systems in a network are rejuvenated using **intelligent time and load algorithm** which determines the system time to rejuvenate under variable work load conditions. Once the system is rejuvenated it needs to be completely tested using of non-deterministic system testing (NDST) across all parameters such as state of admin processes, workload conditions, input/output response time etc.

3-NON DETERMINISTIC TESTING

There are several approaches to testing complex systems using test methodologies. Certain tests sometimes fail and sometimes pass, without any deliberate or intentional change to product code, test script or environment. Those tests that result in such unpredictable and non-repeatable results can be classified as non-deterministic tests. When compared, the technique used for validating and testing deterministic logic differs from Testing and validating non-deterministic logic [9]. Non deterministic testing is highly useful as it tests system randomly against different odds after rejuvenation to uncover bugs or panic which normal deterministic tests fail to find.

3-1 Non Deterministic Testing Overview and Limitations

The issue of maintaining the availability rate, while rejuvenating, environment dependent complex systems, are very high. Implementing s2disk program needs to be tested completely with all possible workloads to ensure all the processes [P_1, P_2, P_3, P_4……P_n] in the given environment are balanced.

There need to be some non-standard forms of testing that creates random test cases to exercise components and features of a rejuvenated system or rejuvenated node in a network. Non Deterministic System Test [NDST] suit are written with an objective to stress the system and to discover the latent regression issues that may not be uncovered by other well-defined test methodology. NDST suit handles various subsystems with any complicity and chaotic workloads intended to find race conditions. It runs on a sized environment so as to maximize CPU and memory utilization with heavy stress testing and high capacity workloads.

Non-deterministic mechanism are used due to the diverse nature of different applications or processes running on complex systems, at a given point it becomes almost impossible to track the last followed path, and can throw a spanner on a long run, raising an issue of repeatability of bugs or the lack of ability to re-create the bug and does not guarantees the two instances of same test with same results. Hence, there is no way to compare two tests [10].Due to this inconsistency there need to words shifting non-determinism to determinism raised. The result is D-NDST (Deterministic-Non-Deterministic System Testing).

The table 1 shows the comparison between NDST and D-NDST.

Table 1 NDST, D-NDST Comparison

NDST	D-NDST
Low predictability & repeatability for bug or panic verification	Medium rate of predictability & a very high rate of repeatability rate (80%+ using post command replay)
Higher rate of no fix bugs	Low rate of no fix bugs
Highly randomized	Controlled random test
Great testing method which tests multiple complex features	Excellent testing method with higher reliability

3-2 APPROACH OF D-NDST

The objective of this technique is to device and implements methods to increase the probability of bug reproduction for Non Deterministic tests. The approach matches the description of a record and replay method. The system data from last execution of NDST will be collected using a data collector engine. The data collected will be processed to identify the sequence of operations executed on the system in an independent environment or the nodes of a network in a dependent environment during the test.

The tool being developed will create a time based map of the command execution sequence .The command sequence thus deciphered will be fed to an execution engine. The role of the execution engine would be to replay the sequence to track the last followed path and recreate the panic or bug that was hit in the original suite run.

The test bed containing the information about the system or information about the nodes in a network should be same with respect to the past execution. For a system in a network or cluster other parameters such as management. IP, object names, DNS, NIS should be same. This is because the different nodes might have different configurations which lead to the creation of panic in the system.

3-3 D-NDST COMPONENTS

The basic components of Deterministic-NDST (D-NDST) makes deterministic testing highly reliable which includes complex systems or node in a network and the execution engine. Execution engine supports the data collector component to collect data fired on the rejuvenating system to test its processes P_1, P_2, P_3..... P_n [image show, process status, process memory consumption etc.] and the status of all the process including success, failure and pending is collected. Execution environment provides a framework in which test scripts can be written and executed. The Data Collector Engine collects metrics for each rejuvenated system.

There are two types of workloads to be considered, object workloads and client workloads. The client workloads provide basic client services such as server request, response, update etc. The object workload includes hundreds of functionalities such as system access, s2disk copy image check, load running, memory consumption , node in a networked environment, information about other nodes, etc.,. To reproduce the panic the main challenge is to save the previous configuration of nodes and note when the object workload started.

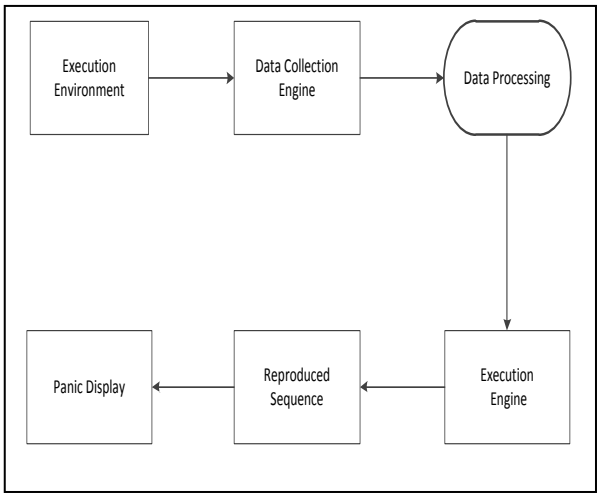

Figure 3 Basic components of D-NDST

3-4 D-NDST JOB LIST WITH EXECUTION

The job list for D-NDST includes the collection or sub categories of jobs that are executed as part of this approach.

Table 2. D-NDST suite

JOB LIST	OBECTIVES	REMARKS
#PREJOB	• Task initialization • Configuration initialization	Test features such as status, correct copying of image, memory consumption, recovery actions, volume stress, and system stress.
#MAIN	• Replay object workload	Runs client workloads on systems
#POSTJOB	• Log collector • Bug re-producer	Workload generator for last execution.

NDST suites start with Prejob, main and post job where Prejob and post job are 100% same for every executions which follows the main run including setting up client workload and object workload which run in parallel. Due to this randomness happens and commands executed vary from run to run.

3-5 D-NDST EXECUTION ENGINE FLOW

Figure 4 depicts the structure and flow of execution engine calling relationships describing conversion of NDST into D-NDST.

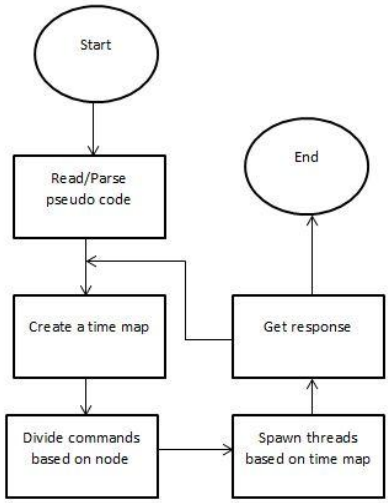

Figure 4 Execution engine flow

The execution engine is the main functional component which generates the time based sequence of original NDST executed commands. The execution engine after generating the time based sequence of commands creates the sub process or child processes for each command sequence to generate it parallel or sequentially as depicted in the original suite. When panic occurs in a system it is stored as a core file and can be automatically or personally inspected to check whether it matches the last hit panic or bug.

4- EXPERIMENTAL ANALYSIS AND RESULTS

Application performance on non-deterministic executions systems does not only depend on the amount of resources provided by the Complex system, but also on co-scheduled applications which share the resources, configuration of system, warm rejuvenation with s2disk program at optimal rejuvenation period. Running D-NDST suites couple of time, there are maximum chances of hitting the panic or bug, rather than running whole original suite which will create command sequence at every different execution.

Table 3. Initial results

Probability of panic recreation after running NDST	0 to 10%
Time for recreation	Many iterations(still very less chances)
Conclusion	No repeatability

Table 4. Final results with D-NDST

Probability of panic recreation after running NDST	70 to 85%
Time for recreation	Within few iterations
Conclusion	Higher repeatability

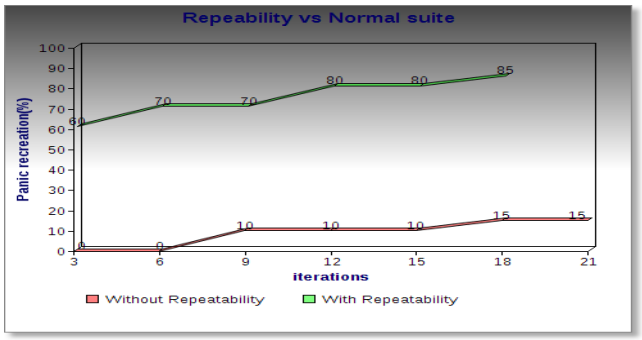

Figure 5 Repeatability VS Normal suite

The figure 5 shows the graphical representation of recreating panic in new execution. The module provides a great flexibility and performance in recreating the panic or bug in any rejuvenated system to check whether all processes, sub processes and applications are working as were in original suite before rejuvenation.

5- CONCLUSION

This paper mainly focuses on testing a software rejuvenation policy for complex system using deterministic-NDST approach. The concept of warm reboot using intelligent time load balancing policy was introduced to rejuvenate the system at optimal rejuvenation period using s2disk programs. Non deterministic testing mechanism ensures rejuvenated system enclose all processes, sub processes and applications which were working as original before rejuvenation. This concept can be applied across different suites besides NDST to recreate the panic with more accuracy and in short span of time. This approach increases the probability of bug or panic reproduction from mere 10 % to 85% approximately. In future more work can be done in this area by considering client workloads and trying to reproduce panic even when the nodes are replaced in a given networked environment.

REFERENCES

[1] H. Okamura, T. Dohi, Analysis of a Software System with Rejuvenation, Restoration and Checkpointing[J], Service Availability, 2008:110-128
[2] M. Grottke , L.Li, K. Vaidyanathan, K.S. Trivedi, Analysis of Software Aging in a Web Server[J], IEEE Transactions on Reliability, 2006, 55(3): 411-420.
[3] Paul, Wolfgang J. ; Pippenger, N. ; Szemeredi, Endre ; Trotter, William T "On determinism versus non-determinism and related problems.". Foundations of Computer Science, 2013., 24th Annual Symposium on Digital Object Identifier: 10.1109/SFCS.2013.39 Publication Year: 2013 , Page(s): 429- 438
[4] .; Jeannot, E. ; Dongarra, J.J. "Robust task scheduling in non-deterministic heterogeneous computing systems" Shi, Z.Cluster Computing, 2006 IEEE International Conference on Digital Object Identifier: 10.1109/CLUSTR.2006.311868 Publication Year: 2006 , Page(s): 1- 10
[5] A. van Moorsel and K. Wolter, "Analysis of restart mechanisms in software systems," Software Engineering, IEEE Transactions on, vol. 32, no. 8, aug. 2006.
[6] M. Sullivan and R. Chillarege. Software Defects and Their Impact on System Availability - A Study of Field Failures in Operating Systems. In Proc. 21st IEEE Intl. Symposium on Fault-Tolerant Computing, pages 2-9, 2011.
[7] Test and validation of a non-deterministic system — True Random Number Generator Udawatta, K. ; Ehsanian, M. ; Maidanov, S. ; Musunuri, S.High Level Design Validation and Test Workshop,2008. HLDVT'08 IEEE International Conference.
[8] Brinksma, E "Testing times: on model-driven test generation fornon-deterministic real-time systems" Application of Concurrency to System Design, 2004. ACSD 2004. Proceedings. Fourth International Conference on Digital Object Identifier: 10.1109/CSD.2004.1309110 Publication Year: 2004 , Page(s):3-4
[09] Kishore.S.Trivedi, Sanders, W.H. Chau, "A Performability-oriented Software Rejuvenation Framework for Distributed Applications", IEEE Computer, Vol.24, Issue 9, 2005, pp570-579.
[10] Kishore.S.Trivedi, Vaidyanathan.K, Goseva-Postojanova.K, "Modeling and Analysis of Software Aging and Rejuvenation", Proc. 33rd Annual Simulation Symposium., IEEE Computer Society Press, Los Alamotos, CA, 2000, pp.270279.
[11]Paulo J F, Kishor S Trivedi, Pedro Barbetta.A, "Accelerated degradation test applied to software aging experiments" IEEE Computer, Vol 59,Issue 1, 2010, pp102-114.
[12]Letian Jiang, Guozhi Xu, Xiangyu Peng, "Time and Prediction based software rejuvenation policy", Information Technology and Computer Science, Kiev, 2010, pp114-117.
[13] Sachin Garg, Y.Huang, C.Kintala, Kishore.S.Trivedi, "Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality", Proc. 1stFaultTolerant Symposium, 1995, pp.22-25.
[14] Vaidyanathan K, Kishore.S.Trivedi, "A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems", Proc. 10th Int'l Symposium on Software Reliability Eng., IEEE Computer Society Press, Los Alamitos, CA, 1999, pp.84-93.
[15] Sachin Garg, Puliafito, Telek, Kishore.S.Trivedi, "Analysis of Software Rejuvenation using Markov Regenerative Stochastic Petri Net", Proc. 6thInt'l Symposium on Software Reliability Eng., IEEE computer Society Press, Los Alamitos, CA, 1995, pp.24-27.
[16] Virtual Machine Migration Comparison. "VMware vSphere vs Microsoft HyperV". A principled technologies test report, commissioned by VMware Inc. Principled Technologies Inc.2011.
[17] Dawei Huang, Deshi Ye, Qinming He, Jianhai Chen, and Kejiang Ye. "VirtLM: a benchmark for live migration of virtual machine". In Proceedings of the 2nd ACM/SPECInternational Conference on Performance engineering (ICPE '11). ACM, NewYork, NY, USA, 307316. 2011.