International Journal of Computer Applications Technology and Research
Volume 5– Issue 7, 422 - 432, 2016, ISSN:- 2319–8656

Program Aging and Service Crash

Shahanawaj Ahamad, Ph.D.
Dept. of Computer Sc. & Software
Engineering, College of Computer
Science & Engineering,
University of Ha'il,
Ha'il, K.S.A.

Abstract: Program aging is a degradation of performance or functionality caused by resource depletion. The aging affects the cloud services which provide access to big data bank and computing resources. This suffers large budget and delays of defect removal, which requires other related solutions including renewal in the form of controlled restarts. Collection of various runtime metrics are more significant source for further study of detection and analysis of aging issues. This study highlights the method for detecting aging immediately after their introduction by runtime comparisons of different development scenarios. The study focuses on aging of program and service crash as a consequence.

Keywords: Program; Aging; Cloud service; Renewal, Metrics.

1. INTRODUCTION

Aging occurrence comprises in expansion of crash rate or performance deprivation of computers as it implements, which is due to the collection of faults in computers state and depletion of resources, for example, physical space [1] [2]. This occurrence is known to experts since quite a while. Early indications of software aging were found in 1960s [3]. As programming is growing in size and difficulty, program aging is perceived in expanding number of long-running computers, together with communications. Program aging is accredited to indirect computer program bugs. Researches in early 1990s on telecom computers [6] emphasized high occurrence of viruses and bugs which, when activated, don't instantly bring about program crash, however show themselves as space spillage, lock of unreleased files, corrupted data and accumulation of numerical fault, slowly degrading computers performance and finally to failure. Sometimes such viruses and bugs are too subtle or too expensive, making it impossible to be expelled during improvement. Studies at AT&T Bell laboratories on error enduring program recognized as program renewal as economical solution to counter program Aging [1] [7] [3]. Program renewal is a positive method to prevent performance deprivation and crashes from program aging. It comprises incidental or periodical tidy up of aging impacts (which are accomplished by computer program restart, or by more difficult procedures), so as to delay crashes and reinstate performance. Program renewal represents a unique type of defensive program support contrasted with different types of defensive program upkeep [8], which were centered on to install updates so as to avert field crashes [9] or redesign a package, to adapt to outdated quality [10].

Accuracy of Aging Oriented Crash (AOC) detection approaches is largely determined by aging indicators. A well-designed aging indicator can precisely indicate AOC. If subsequent renewals are always conducted at real crash-prone state, renewal cost will tend to be optimal and significant with optimum schedule. But unfortunately, prior detection approaches based upon explicit aging indicators [11] [12], [13], [14], [15] [16], [17]. These approaches don't function well especially in face of dynamic workloads. Mostly they miss some crashes which lead to a low recall. Insufficiency of previous indicators motivates to seek novel indicators. There are some motivational aspects as follows.

1.1 Insufficiency of Explicit Aging Indicators

To distinguish normal state and crash-prone state, a threshold should be present on aging indicator. Once aging indicator exceeds threshold, a crash occurs. Traditionally, a threshold is set on explicit aging indicators. For instance, if CPU utilization exceeds 90%, a crash occurs. However, it's not always case to happen. External observations do not always reveal accurately internal states. Here internal states are referred to as some normal events (e.g. a file reading, a packet sending) or abnormal events (e.g. a file open exception, a round-off error) generated in computers. Abnormal events are more concerned here. CPU utilization is observed as real number in range 0%, 100% while abnormal events are very limited. Therefore an abnormal event correlates with multiple observations. When a crash-prone event happens, CPU utilization is 99%, 80% or even 10%. Therefore explicit aging indicator cannot signify AOC sufficiently and accurately; and if computers fluctuation is taken into account, situation gets even worse. This is also a reason why it's so difficult to set an optimal threshold on explicit aging indicators in order to obtain an accurate crash detection result.

1.2 Entropy Increase in VoD Computers

As explicit aging indicators fall short in detecting AOC, turning to implicit aging indicators is helpful. Some insights are attained from [18] and [15]. Both of them treated program aging as a complex process. With their motivation it is believed that entropy can be a measurement of complexity, which has a potential to be an implicit aging indicator. VoD system entropy increases with the degree of program aging. VoD systems run for 52 days until a crash occurs. By manually investigating reason of crash, it is assured to be an AOC. Entropy value of CPU utilization every day is calculated. It's apparent to see entropy values of last four days are much larger than ones of first four days nearly at all scales. Especially, entropy value of day 52 when computers ailed is different significantly from others. However raw CPU utilization at crash state seems normal which means crash cannot be detected if using this metric as an aging indicator. Therefore, it can be a potential aging indicator in this practice.

1.3 Conjecture

According to above observation, it provides a high level abstraction of properties that an ideal aging indicator should satisfy. *Monotonicity*; since program aging is a gradual deterioration process, aging indicator should also change consistently with degree of aging, namely increase or decrease monotonically. As most essential property, monotonicity provides a foundation to detect Aging Oriented Crash accurately. *Stability*; indicator is capable of tolerating noise or disturbance involved in runtime performance metrics. *Integration*; as program aging is a complex process affected by multiple factors, indicator should cover se influence from multiple data sources, which means it is integration of multiple runtime metrics.

In cloud computing most of PMs are converted to VMs [19]. With constant use of simulated machine and VMM causes program aging. As VMM is concept level between hardware and operating computers, many functions are running over it and are not boot up regularly.

2. LITERATURE SURVEY

Numerous studies have been carried out on program aging and renewal. Program aging is an old topic but it lacks research owing to which safety critical functions face program aging. However, some research on program aging is done in simulated setting.

Rivalino Matias et al. [20] conducted research on space associated program aging problems which produces aging and associated crashes. They concentrated on space leakage glitches. They conferred disadvantages of utilizing renowned computers wide and function-specific aging indicators and suggest effective results for their circumstances. Space-associated aging impacts are produced by space leakage and space disintegration issues. Space leakage is a computer program fault which is caused by improper utilization of space organization practices. Space leakage happens when a function process assigns space blockages and do not discharge them back to operating computers in course of its runtime. Researchers tried to discover space leakages both in user level and kernel level by utilizing aging indicators. Aging indicators detects errors in a computer in working state. Computers wide aging indicators provide information on sub computers constituents. They conducted experimentations with help of computers wide aging indicators free/used physical space and swap space. However these indicators indicate false signal regarding space usage. So aging free baseline are utilized to compare space usage with it for improved outcome. Function particular aging indicators give particular information about individual function process. For identifying space leakages, utilization of process resident set size (RSS) as aging indicator is suggested. Checking RSS in combination with procedure simulated size is an improved approach than utilizing only RSS which alone can't uncover right space.

Domenico Cotroneo et al. [21] concentrated on program aging phenomenon associated with integer runoffs. Integer runoffs are neglected issues in review of literature. Arithmetical aging related bugs signify problems for numerous long-run program functions, for example control of industrial computers and signal handling. It is difficult to evade and fix mathematical bugs, dearth of prominence for computer mathematical and program development languages by developers. Researchers presented some examples of integer overflow that causes aging. They highlighted about mathematical aging related bugs. Integer associated bugs happens when computers analyst put infinite mathematical integers to finite range. They examined and debated on various instances of numerical aging related bugs in MYSQL open source DBMS to provide physical world issues. But owing to lack of interest for integer runoffs problems, such particular renewal techniques were not developed or than restarting DBMS server. Program renewal techniques mitigate impact of program aging because of integer runoff, it lacks forecast about aging. So periodically sampling integer variables to approximate expected time to runoff for variables at run time and activate renewal methods in relation to estimated value. For future studies, performance appraisal and optimization of approach, to use method to or type of computers, and to include floating-point faults.

Autran Macedo et.al [22] suggested space related aging effects. Researchers explained, what means space management functions inward functions process, focus on two space issues that bring about program aging; disintegration and leakage. They described procedure of space-related program aging concentrating on actual and extensively accepted space allocator and offered a tentative study which shows by what means space break-up and leakage take place and by what method they accumulate over time so as to bring about computers aging-related crashes. For exploratory study, test lab consist of Intel Pentium Octa Core, 3GHz, 2GB RAM, operating on Linux with glibc 2.10.1.y formed 2 programs(Mfrag and Mleakage) to device test cases identified with space discontinuity and space spill. Though total free space in heap is bigger than amount asked for, another space is asked for in light of fact that stack is experiencing external space break-up. Asking for another block to OS infers ingoing in kernel mode, which presents an additional overhead which punishes procedure performance. Space leakage inside OS kernel influences whole computers and not a particular process, whose impacts stay until operating computers restart/reboot. In future studies, concentration is on experimenting space related impacts in simulated setting and not vulnerable to space breakup.

Lei Cui et al. [23] concentrated on in simulated setting for program aging shortcoming. Aging rate is perceived by critical experimentations on physical and VMs and recognize contrasts around two, and recommend a component code-based practice for crash foresight through computers call, then perform a model in VM official level to foresee crash time and revive. They led four experiments to recognize aging event in physical machines (PM) and VMs, and figure rate of aging for assessment. For investigation, three sorts of computers resources were gathered for measurements that demonstrate program aging Space resources; (1) Free Space and Active Space; (2) Processor resources containing User and Computers Time; (3) IO resources for instance, Block Read or Write Count and IO Waiting Time. Amid measurable examination, declining of free space size was found. Relating aging rate between PM and VM, aging rate is more prominent in VM in contrast with PM. Scholars proposed code-based strategy to expect renewal time bring up highlight models through computers calls.

Kehua Su et al. [24] proposed a work on program renewal in simulated setting (SRVE) to manage program aging marvel of VM screen and VM, and to make them enhance execution. Program running in VM computers is not boot up every now and again, so aging issues exist in VMM and VMs. So creators proposed some renewal computers for it. Methods give renewal of VMM and VMs, however it can't give zero idle time of administrations.

Kenichi Kourai et al. [25] proposed another method for quick renewal of VMM known as warm VM reboot. As VMM is basic program for running VMs, its execution debasement influences all VMs that rely upon it. So creator here proposed another program renewal procedure that recoveries both idle time cost and time. Warm-VM reboot empowers effectively rebooting a VMM by suspending and continuing VM. They created two components: on-space suspend/resume of VMs and snappy reload of a VMM. At the point when a VMM is restored by computers reboot working computers running on VMs based on top of a VMM likewise are boot up when VMM is revived. This expands idle time of managements by working computers. Here they contrasted warm VM renewal and VM movement. In this paper they clarified issue of program renewal of VMM. They executed their examination in view of Xen and performed a few experiments. They thought about various renewal strategies like computers reboot, icy VM, warm VM and VM relocation. Contrasted and an ordinary reboot, warm VM reboot decreased idle time by 75% at most. Warm VM reboot accomplished higher aggregate output than computers utilizing VM relocation and a typical reboot.

Fumio Machida et al [26] exhibited issues of ability of performance administration in a simulated information center (VDC) that has numerous administrations utilizing virtualization. Performance ability is an idea of a blended metric of execution and accessibility. Clients of a VDC demand a specific level of function execution in a service level agreement (SLA). VDC suppliers choose an ideal server design and administration operations for ensuring function execution and increasing accessibility. They concentrated on position algorithm of simulated machines and renewal plans for VMs and VMM in a VDC. VM positions, which allocate VMs to functions, are chosen for fulfilling execution prerequisites under predetermined number of physical servers. Renewal timetable are chosen for VMs and VMMs for expanding general computers accessibility in a VDC. Amid down time of a VM, number of accessible function occurrences declines and execution of function administration go down. Amid down time of a VMM, VMs and functions running on same physical server are down too. Objective of performance ability administration in a VDC is to find an ideal VM arrangement with ideal renewal plans for VMs and VMMs. ideal VM position and timetables enhance general accessibility and execution of VDC under restraints of execution levels of functions determined in SLAs.

Kenichi Kourai et al. [27] proposed another procedure for quick renewal for VMM called as warm VM reboot. When a VMM is restored working computers running on VMs based on top of a VMM additionally are boot up. This expands idle time of administrations contributed by working computers. It requires long investment to reboot numerous working computers in parallel when VMM is boot up.

Aye Myat Paing et al. [29] concentrated on renewal of VMMs effectively without influencing VMs. They joined renewal procedures with Live VM movement innovation for better advancement of resources utilization. By utilization of stochastic Petri nets they gave a model utilizing time based renewal for VMM. To assess model they gave numerical examination.

Thandar et al. [30] introduced a Markov model for investigating accessibility for long running functions which experience the ill effects of program aging. In that model they demonstrated accessibility, idle time and idle time budget amid renewal.

3. PROGRAM AGING CONCEPT

Program aging is not a new phenomenon but it was existing years before and suffering the systems and services. The concept was highlighted in year 1994 [10] after that several research studies have been undertaken to explore the issues, domain and develop corresponding solutions as for servers, applications, services and virtual machines.

3.1 Causes of Program Aging

There are two, entirely unmistakable, sorts of program aging. Initially, is brought about by crash of item's owners to adjust it to address evolving issues; second is consequence of changes that are made. This "one-two punch” prompts fast decrease in estimation of a program item.

3.1.1 Lack of development

Over three decades, assumptions about program have changed significantly. When interactive programming introduced, mysterious charge dialects were utilized. Nowadays, everybody tackles line access, "moment" reaction, and menu-driven interfaces conceded. Program is old despite the fact that no one has touched it. Clients in mid-60 were energetic about item, today's clients expect more. Unless program is often redesigned, it's client's will get to be disappointed and they will change to another item when advantages exceed expenses of retraining and changing over. They will allude to that program as old and obsolete.

3.1.2 Lack of change

Despite the fact that it is crucial to redesign program to anticipate aging, changing program causes an alternate type of aging. Program designer had a straightforward idea when composing. Program is big, understanding that idea permits one to discover those segments of project, which are modified when an update or redress is required. Understanding that idea infers understanding interfaces utilized inside and amongst computers and its surroundings. Changes are made by individuals who don't comprehend unique configuration idea quite often cause structure of computers to corrupt. Under those circumstances, changes will be conflicting with unique idea; indeed, they will discredit unique idea. Often harm is less, but sometimes it is very serious. After those developments, one must know both unique configuration rules, and recently acquainted exemptions with principles, to comprehend item. After numerous such changes, unique planners no more comprehend item. Individual who rolled out improvements, never did. At the end, no one comprehends adjusted items. Changes take longer and will probably present new "bugs". Change incited aging is frequently exacerbated by the certainty, maintainers feel they don't have sufficient time to redesign documentation. Documentation turns out to be progressively incorrect in this manner rolling out future improvements much more troublesome.

3.1.3 Lack of space allocation

Aging is computers delay brought about by crash to discharge dispensed space. Documents develop and require pruning. At time space distribution routine do not discharge all space that is assigned. Gradually, swap and document space are reduced and execution corrupts. This issue is frequently a configuration crash and is aftereffect of absence of progress or worsened by changing use designs. A clean up procedure mediates and clean up record computers and space, enhanced schedules make clean up happen quickly and computer program is considered totally "cured".

4. AGING IMPACT ANALYSIS

Examination of aging impacts (i.e., sort of invalid states brought on by aging) and aging markers in this area demonstrates how aging is showing difficulty in program computers. Aging markers are a critical zone of study, since they are instrumental for identifying when computers state is inclined to aging crashes, by observing them amid computers execution. Aging markers are pointers of resources utilization and execution markers.

Space utilization: Empirical confirmation demonstrated free space shows most brief Time to Exhaustion (TTE) among computers resources [31], and space administration faults are a noteworthy reason for crashes [32]. Therefore, numerous studies on aging and renewal analyses program aging phenomena influencing free space, by measuring quantity of free physical space and swap space [33], and a few estimation based methodologies apply time arrangement and measurable models to these variables.

Execution debasement: SAR reported execution corruption in program computers influenced by aging. A reason for execution corruption is exhaustion of computers resources: for example, utilization of physical space builds time required by space distribution methods and waste gathering instruments, since their computational intricacy is an element of measure of space regions that apportioned [34] [35]. An expanding demand reaction time and a diminishing output accounted for web functions, web servers [2], and CORBA-based functions [36]. Renewal are activated when nature of administration (e.g., as far as reaction time or throughput) is underneath a given edge.

Resources utilization: Program aging effects few sort of resources. Other than space-related resources (e.g., physical space, simulated space, swap space, cache space), studied papers manage these kinds of resources:

- File computers-related resources, for example, stream descriptors and record handles [31] [37] [38];
- Capacity, whose space is consumed by awful administration [39];
- Computers related resources, for example, attachment descriptors [37] ;
- Concurrency related resources, for example, bolts, strings and procedures [31] [38];
- Function particular resources, for example, DBMS shared pool locks [40] and OSGi references [41].

In a few studies, methodology proposed is not constrained to a particular resource, but concentrated on distinguishing inaccurate API use and wrong exemption handlers which bring about a resources spillage. Working example, [38] presents a methodology which mines resources utilization designs by checking API calls, and gives an exploratory assessment on open source programs in light of Java I/O and concurrent APIs. A normal sort of resources spillage in Java projects is characterized by outlets and record handles, because of defective exemption handlers that don't discharge these resources [37] [38]. Resources likewise are influenced by program aging depending on sort of computers, for example, free disk space in DBMS computers [39]. Some works investigate a more extensive arrangement of resources. In [31], a system of UNIX workstations was checked to distinguish aging patterns in utilization of a few resources (identified with simulated space, OS portion, file computers, disk, and organize), and critical aging pattern was seen in procedure table size and in document table size (despite the fact that their TTE is lower than TTE of free space).

Anyhow aging impacts mentioned above, re-exist other sort of aging impacts focused in late works. A field in which program renewal is studied is identified with security assaults, that is, presence of pernicious clients to access unapproved resources or to make computers inaccessible. Security assaults occur and continuously trade off a computers over a drawn out stretch of time (e.g., PIN phishing through brute force speculating, or flood assaults which trigger program aging wonders), which are lessened by intermittently reviving a computers, for example, by changing cryptographic keys, by restarting negotiated procedures, and by randomizing area of information and guidelines in space [1] [42] [43] [44] [45] [46]. A challenge in sending program renewal for security reasons for existing is to characterize exact aging pointers which are identified with security assaults. At present, aging rate are accepted at outline time [42] [47] or have to be founded on flawed assault/interference indicators which could raise false cautions and miss assaults [48] [49].

Another sort of aging impacts examined in a couple of recent works, which are alluded as other aging impacts, are identified with amassing of numerical faults [2] and space discontinuity [50] [51]. These sort of aging impacts are not inexorably brought about by bugs in program, but rather are identified with nature of floating-point mathematics and space allotment calculations, separately. An occurrence of numerical mistakes, such things in writing aging markers ready to gauge degree of faults in computers state were not found.

Finally, numerous studies propose models and methodologies for managing aging paying little attention to which particular sort of resource exhaustion or aging impact is experienced, which is normally instance of model-based studies.

The greater part of past studies concentrated on program aging impacts are identified with space utilization [31], [52], [53], performance corruption [54], [55] or both [2], [36], [34], [52]. These two perspectives are most regular issues happening in non-safety-critical computers and they are considered by an expanding number of SAR studies. These issues are less persistent for safety-critical systems. For example, an occurrence of program which experiences a safety confirmation process, dynamic space administration is avoided to achieve stringent safety integrity levels. In contrast, none of investigated research handled math issues, for example, collection of round-off faults. These faults are more applicable in safety-critical settings, with the fact that computer program is in charge of controlling physical actuators and mistaken outputs have extreme results. A surely understood case of aging crash identified with numerical faults happened in patriot rocket computers, which was created by a round-off mistake in change of aggregate execution time from a whole number to a floating-point number [33].

5. CONSEQUENCES OF PROGRAM AGING

Indications of program aging reflect those of human aging: (1) proprietors of aging program discover it difficult to stay up with business sector and lose clients to more up to date items, (2) aging program debases in its space/time execution as an after effect of slow collapsing structure, (3) aging program frequently gets to be "buggy" on account of faults presented when changes are made. Each of these results are expensive to proprietor.

5.1 Crash

As programs get aged, it becomes greater risk for crash. This "weight increase" is an aftereffect that an easy approach include an element, includes new code. Adjusting existing code to handle new circumstances is troublesome in light of the fact that code is neither surely understood nor well documented. At first, there is more code to change, a change that is made in a couple parts of unique project, now requires alternate many segments of code. Second, it is hard to discover schedules that are changed. Subsequently, proprietors can't include new components quickly. Clients change to a more youthful item to get those components. Organization encounters an eminent drop in income; when they draw out another version, it is important to a decreasing client base. They endeavor to stay aware of business sector, by expanding their work power, expanded expenses of changes, and delays, leads to further loss of clients.

5.2 Decreased Performance

As size of project develops, it puts more stress on PC space, and more defers as code are swapped in from mass storage. Program reacts slowly; clients must upgrade their PCs to get good response. Performance likewise diminishes as a result of poor configuration. Program is no more useful and changes unfavorably influence execution. The new items, whose unique configuration reflected requirement for newly presented elements will run quicker or utilize less space.

5.3 Declining Consistency

As program is kept up, mistakes are inevitable. In early years of industry, eyewitnesses record circumstances in which every mistake adjusted presented (all things considered) more than one fault. Every time an endeavor was made to reduce crash rate of computers, it deteriorated. Only decision was to forsake item or quit repairing bugs.

6. CLOUD SERIVCES AND CRASH

Principle thought behind cloud computing is highlighted in 1960, John McCarthy envisioned, general people would get computing services like a utility. Term "cloud" is utilized in many milieus, e.g., in 1990, delineating broad ATM networks. Later Google's CEO Eric Schmidt utilized word to portray business model for enabling services across over Internet in 2006, which began to pick up popularity. The term cloud computing is utilized for most part as a marketing term in various situations depict extensive thoughts. But lack of standard meaning of cloud computing has created market build-ups, as well as lot of doubt and perplexity. Consequently, as of late there are work on institutionalizing meaning of cloud computing. As an illustration, work in looked at more than 20 unique definitions from an assortment of sources to affirm a standard definition. This study embrace meaning of cloud computing gave by National Institute of Standards and Technology (NIST) [56].

NIST meaning of cloud computing "Cloud computing is a model for empowering advantageous, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storages, functions, and services) which are quickly provisioned and discharged with negligible management effort or service provider interaction."

Principle purpose behind presence of various impression of cloud computing is that cloud computing is quite an old technology, yet rather a new operations model which unites present technologies to run business in an unexpected way. The vast majority of advances utilized by cloud computing, for example, virtualization and utility-based estimation, are not new. Rather, cloud computing influences these existing innovations to meet mechanical and economic necessities of today's demand for IT.

Cloud service failure can happen once or frequently. There causes for it but aging of program are also one of them, which is propagated with the program bugs. When disaster does strike, there’s no physical data center you can visit to investigate the problem. Below are given some reasons for cloud service crash [63].

6.1 Cloud Provider Downtime

This is because of service provider who provides the cloud infrastructure management and hosting service. It the proper back is done and distributed computing in adopted then risk can be reduced and prevent downtime.

6.2 Security Attacks

If security requirements and concerns are not taking in advance attention than most probably system can be weak and major risk to attack.

6.3 Storage Failures

This can be a top level risk to cloud service crash and system down. Storage failure is serious cause of service crash and unavailability.

6.4 Human Error

This is the mal practice done by the professional who manage the cloud application and service manager. Level of expertise and experience must be as good as possible.

6.5 Demand fluctuation

Sometimes the demand of cloud service are more sometime less. In this type of situation it is managed by system to extend and shrink the infrastructural context to manage the increased and decreased demand of service.

6.6 Third Party Service Failures

The incorporated application and services provided by the third party have to be continuously monitor to proper function of cloud service. Many times the third party application are down on which the current service depends also crash.

6.7 Quality of Service

Poor quality if service , which may have many parameters like network, speed of response, system performance , quality of streaming etc. can also affect the operation the cloud service.

6.8 Poor crash recovery procedures

Every cloud based organizations have to manage the strong recovery system procedure for service crash. The application should include the recovery mechanism and procedure for effective and timely practice.

6.9 Application Bugs

There can be certain types of bugs found in application at run time or even at deployment time which can lead to a service crash and system failure.

7. PERFORMANCE ANALYSIS OF SERVICE DEGRADATION

A significant consideration is dedicated to empirical investigation of program aging from original computers. Since, marvel shows itself as performance corruption and/or resources utilization specialists concentrated on methodologies for extracting estimations from computers [1].

To evaluate job load of computers, number of jobs succumbed to computers every day, on the basis of their start time is considered. To measure performance, normal length of jobs every day, that is, by calculating mean estimation of term of jobs finished every day, on basis of their completion time and submission time. Irrespective of the fact that these measurements give a halfway sign of job load and of performance, these measurements are effectively figured from dataset that was accessible, and empower a preparatory examination of performance debasements.

Study of performance of computers in general (i.e., without part information by node/line and by utilization period) did not point out any diminishing patterns of performance. Rather, some performance debasement patterns on a little subset of nodes and lines in supercomputer was discovered. This recommends program aging wonders are limited in a particular piece of computers for particular sorts of job load. Assumption is generated because of (i) program aging problems (e.g., resources leakages) that are present in program of influenced nodes or lines, and/or (ii) particular sorts of jobs which generate these problems. The investigation on nodes displayed a performance debasement pattern (i.e., a huge increment in normal span of jobs).

Granger causality test pointed out that these performance debasement patterns appear not to be identified with variations in job load. This outcome gives some certainty that patterns are not identified with random varieties of job load, and it is worth to investigate these drifts more carefully. If the normal job accomplishment time is expanding, job load showed comparable varieties, in this manner providing reason to feel ambiguous about presence of a program aging marvel behind that performance pattern.

8. PARAMETERS OF SERVICES PERFORMANCE DECLINATION

More extensive scope of cloud service, which related to programs, gets ineffective. Aside from security issues, for example, securing innumerable bits of individual information scattered in cloud or conceivable protection infringement, countermeasure arrangements are set up to control computer program weakness. An issue suffered by one occupant raise to another occupant in the event that they share same service. Overcoming dangers connected to vulnerabilities and giving more dependable, higher quality service than contenders are greatest achievement component. Program weakness has not been altogether taken care of in best practices for conventional program improvement yet, so technology is not developing enough to manage shortcomings in cloud service. Weakness control of cloud service requires comparing counter measures for anticipated weakness issues when scope of service clients is not constrained.

It gives information and markers of performance, where that performance level influences reception of cloud services by clients. Performance is among points of interest that ought to be accessible in cloud services since performance affects clients and service providers. To assess performance, consider a few criteria to assess components that influence performance of cloud services, including normal reaction per unit time and normal holding up time per unit time and others in properties of performance measures of cloud [57].

• SaaS - Evaluation is made by clients specifically relying upon performance measures, velocity of reaction, dependability of specialized services and accessibility.

• Pass - Evaluation is made by clients specifically or by implication relying upon performance measures in light of detail, efficiency, dependability, specialized service and middleware capacity.

• IaSS - Performance measures are resolved relying upon framework performance, limit, unwavering quality, accessibility, and versatility.

Performance and assessment are measured relying upon responsive time, efficiency, and timing in executing jobs, viz., and handling activities in suitable period. The SLA, agreement confines clients and cloud service providers. Levels of service or quality of service (QOS) are considered. Service performance is portrayed by reaction time, profitability, accessibility, and security. Quality of Service (QoS) in cloud shows level of performance and dependability, regardless of the fact that attributes of quality of service got consideration before development of cloud, performance, homogeneity as well as principles [58].

To perform an operation at which capacity and security are accomplished, viz., any service ability to guarantee classification of a bit of information worked with, traded or put away, confidentiality of correspondences, legitimacy and safety of traded or put away information, and protection of client and his correspondence implies against any sort of danger, notwithstanding or characteristics of exactness, unwavering quality, adaptability and convenience [59]. There are a few dangers and confronting clients when they utilize cloud services. They are expanding, that when service providers or resources exist outside local extension, viz., they are under various laws. In connection to appropriation of cloud in advanced education, there are some difficulties, for example, security, performance, proficiency and control [31].

Security element influence performance through security sway on network framework, for instance, is case with DDoS assaults which broadly affect network performance. This danger or any dangers that undermine cloud environment, it will be a noteworthy matter for clients and suppliers [60]. These assaults are hurtful to computing. Protecting SQL assault permits assailant access to database. Likewise, in flood assaults assailant sends a solicitation for resources to cloud so rapidly that he exploits capacity of any remote resources by normal clients and here numerous assaults which incapacitate security of cloud happen [61]. Security is a vital need to cloud clients. Decision is to make utilization of cloud services founded on level of secrecy, honesty, adaptability, and security services accessible, and this is a sign to rivalry among service providers which prompts advancement of cloud computing [30]. There are a few issues confronting cloud computing clients, as far as access to information through Internet; any shortcoming in level of security in cloud is debilitating secrecy of clients information stored. In addition, quality of association influences level of performance in conveyance of services, high cost of private automated computing contrasted and open and blended segments, at cost of quality, performance and security [40].

9. RELATIONSHIP OF SERVICE PERFORMANCE WITH AGING PARAMETER

When computing resources almost depleted, variety of resources variable will bring about or resources variables to change. Three imperative resources variables change clearly. An inquiry is the way to recognize which resources variable is main driver of harming steadiness of resources dissemination of PC. The underlying driver of program aging is buffer expand, unmoved CPU reduction and cache diminish separate in three times of simulation, and test where output of the model fit perceptions.

Particularly, input of real estimation of one parameter and starting estimation of or two parameters into model, and figure estimation of or two parameters with time. For instance, the real estimation of buffer and introductory estimations of inactive CPU and cache as input parameters. The inactive CPU and cache of dynamic model is figured out at next interim by repeat, with real estimation of buffer as input value. Further, the cache or inactive CPU as input parameters individually and ascertain other two parameters. Results are not recorded on the grounds that output of element model can't fit observations.

Buffer use increment causes change of other parameters, will output comparable dynamics as exploratory perceptions. There are a few reasons: (1) this model utilizes three variables, so bond between three variables is not precisely depicted; (2) buffer utilization increment is the reason of other two variables, numerous auxiliary elements are excluded in this model; (3) More higher request of polynomials will bring higher exactness, while utilizing quadratic polynomials. In spite of substantial mistakes, this model is compelling, on the grounds that the item in simulation is to investigate which resources variable is underlying driver of program aging, rather than precisely gauging estimation of cache or accessible CPU. This examination helps to comprehend conduct of computer program when it slowly ages.

10. RESULTS AND DISCUSSIONS

To distinguish a relationship amongst measurements and aging, the initial step is to assess connection with individual measurements. Pearson connection coefficient between every metric and aging patterns were evaluated; this coefficient are utilized to test a direct connection between two variables [1]. Measurements with a measurably noteworthy relationship (p-esteem < 0:05) are observed. All measurements identified with computers size are connected with program aging. This affirms assumption that there occurs an association between program aging and program multifaceted nature. Few measurements don't show direct connection with numerical implications. It is included in resulting investigation; there is a non-linear relationship (alone or in blend with or measurements) not found by this preparatory test.

Statistical regression models were adopted to acquire a quantitative relationship between program measurements and aging. Since a direct relationship with a few measurements was watched, various straight relapse models were assessed. To make this model, common relationship among measurements, e.g., a program with high LOC will have a high number of capacity affirmations, needs to be managed; this connection prompts an instable model, since little change in information bring about expansive change in model. Therefore, stepwise strategy to manufacture model, viz., particular variables are presented or expelled from model and a statistical significance test is performed to choose top model. This technique created a linear model with one variable.

The model is described by high standard deviation of residuals (1:3185 MB/h). This high fluctuation influences expectation for program modules with a low aging pattern (1 MB/h), prompting a high normal relative mistake (1:686 106%). In addition, free variables are portrayed by a high inter correlation, since one variable is brought into model by stepwise technique.

To get an exact model, Principal Component Analysis (PCA) strategy was adopted, which changes dependant variables in a small number of uncorrelated variables [62]; yet, this methodology did not enhance model. An exponential and a logarithmic model was assessed, however they were not ready to give better accuracy. Absence of basic and exact model is because of heterogeneity of program modules. A perceptible element of dataset is expansive scope of qualities in aging patterns, they vary by a few order of magnitude. Therefore, dataset was isolated into two disjoint clusters, in particular Big Aging and Little Aging, which were exclusively broke down. Two gatherings were considered because of low number of program modules. Subsequently breaking dataset, stepwise technique to gatherings was connected. Substantially exact model in both cases were acquired. Both models fulfil hypotheses of residuals' homoscedasticity, typicality, and un-correlation with autonomous variables. Specifically, model for little aging gathering is portrayed by a low standard deviation and a satisfactory normal relative fault (around 11%). Dependant variables incorporated into this model, Ratio Comment To Code and Count Line Inactive, don't have all the representative of program complexity; nonetheless, given high relationship between dependant variables, it is inferred that aging pattern of program of this gathering is identified with project size, both dependant variables have a place with this sort of measurements. Although model for Big Aging gathering is superior to opening model, it is described by a high mistake. It is suspected that mistake was because of presence of trace module in gathering, since it is described by a low intricacy and high aging patterns. Therefore, this example is an anomaly and expelled it from gathering;

resultant model was much more precise, with a low normal relative error, around 8%. This outcome was because of immaturity of trace module, which was influenced by extreme aging-related bugs irrespective of the fact that it was a moderately basic module. In this gathering, aging is identified with size of modules (LOC); model represents complex quality of code (Volume). At last, to clarify contrast between two gatherings, and to apply right model to another program, i.e., excluded in dataset, it was assessed on the possibility to characterized modules into gatherings utilizing program measurements. To choose best components, i.e., measurements, to use in classifier, feature choice computers was applied, in particular independent features methodology [62]. This technique performs a statistical test for each individual component, showing that distinction is unrealistic to be random variation; if contrast is sig times lower than standard mistake, then highlight is not regarded valuable for ordering. The test is performed by assessing

$$se\ (A\text{-}B)=\sqrt{\frac{var\ (A)}{nA}+\frac{var\ (B)}{nB}} \quad (1)$$

$$\frac{|mean\ (A)-mean\ (B)|}{se\ (A-B)} > sig \quad (2)$$

Where A and B are same component measured for two divisions, and nA and nB number of samples in classes. Many components were considered utilizing diverse estimations of sig. Adequacy of every arrangement of components utilizing leave- one out strategy: n-1 samples are utilized for preparing a classifier, and rest of the example is utilized for testing classifier; the rest of the samples are utilized for various splitting of dataset. For classification, two-class SVM classifiers were adopted.

Table 1 shows results of put one out validation of best classifier (sig = 3:4). This classifier is effective in 9 out of 10 cases; it is not exact in the event of trace module, which was beforehand appeared to be an irregular example. Feature choice and leave one out authentications were revised without trace service, and best classifier accurately characterized samples in all cases. This outcome bolsters utilization of program measurements for characterizing program regarding aging. Measurements of best classifier were Volume (mean), Effort (mean), Volume (difference), N1 (change), N2 (fluctuation), Length (change).

Table 1: Leave-one-out validation (LA = Little Aging, BA = Big Aging) for sig = 3:4.

Module	Reference class	Anticipated class
Garbage collector	BA	BA
JIT Compiler	BA	BA
Trace	BA	LA
Common	LA	LA
Repository	LA	LA
Load balancing	LA	LA
Xerces	LA	LA
Httpd	BA	BA

11. CONCLUSION

Relationship between program measurements and program aging on ten computer program functions were researched. Program functions belong to two particular gatherings, in which aging impacts are insignificant (Little Aging), and program fundamentally influenced by program aging (Big Aging). A basic model ready to foresee aging impacts of both gatherings at same time were not found, in this manner they are broke down independently. There exist two exact multiple linear regression models for modeling two programs function clusters. Aging patterns in Little Aging group appears to be connected with program size, while difficulty of project as far as operands and administrators, i.e., Halstead measurements, are considered for Big Aging groups. It is probable to arrange program functions in one of two gatherings by utilizing program measurements. Halstead measurements ended up being most appropriate for this reason. These results empower utilization of program measurements for adapting to program aging at advancement time. The classification, of another program is made by recognizing its class (Little Aging or Big Aging), and then applying a customized linear regression model.

Clouds have developed as a definite worldview for managing and assigning services over network. Rise of cloud computing is rapidly shifting prospect of IT, and altering long-held assurance of utility computing into a reality. In spite of noteworthy advantages offered by cloud computing, current advancements are not sufficiently developed to understand its maximum capacity. Many problems in this field, counting programmed resources provisioning, power administration and safety administration, are getting consideration from examination group. There is still enormous opportunity for analysts to make noteworthy contributions in this field, and convey huge effect to their advancement in industry. Examination of aging impacts and aging pointers reports that space and performance issues were most studied in literature.

In this paper, condition of specialty of cloud service performance degradation have been highlighted based on one parameter of program aging among various other parameters and specific causes. The study have shown that a significantly aged program is a high risk and performance failure, can lead to permanent crash of services. Program renewal and SAR have been identified some level of solution approaches but still under investigation and further research to find out a suitable solution of Aging Oriented Crash.

12. REFERENCES

[1] Huang, Y., Kintala, C., Kolettis, N., and Fulton, N. 1995. Computer program renewal: analysis, module and functions. In Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers, Twenty-Fifth Int'l. Symp.

[2] Grottke, M., Matias, R., and Trivedi, K. 2008. The fundamentals of computer program aging. In Computer program Reliability Engineering Workshops, 2008. IEEE

Int'l. Conf.

[3] Bernstein, L. and Kintala, C. 2004. Computer program renewal. CrossTalk 17, 8, 23–26.

[4] Avritzer, A. and Weyuker, E. 1997. Monitoring smoothly degrading computerss for increased dependability. Empirical Computer program Engineering 2, 1, 59–77.

[5] Marshall, E. 1992. Fatal error: how patriot overlooked a scud. Science 255, 5050, 1347–1347.

[6] Bernstein, L. 1993. Innovative technologies for preventing network outages. AT & T TECH J. 72, 4, 4–10.

[7] Wang, Y.M., Huang, Y., Vo, K.P., Chung, P.Y., and Kintala, C. 1995. Checkpointing and its functions In Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth Int'l. Symp.

[8] Kajko Mattsson, M. 2001. Can we learn anything from hardware preventive maintenance? In Engineering of Complex Computer Computerss, 2001. Proceedings. Seventh IEEE International Conference on. IEEE, 106–111.

[9] Adams, E. 1984. Optimizing preventive service of computer program products. IBM Journal of Research and Development 28, 1, 2–14.

[10] Parnas, D. 1994. Computer program aging. In Proceedings of the 16th international conference on Computer program engineering. IEEE Computer Society Press, 279–287.

[11] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational computer program computerss," in Computer program Reliability Engineering, 1999. Proceedings. 10th International Symposium on. IEEE, 1999, pp. 84–93.

[12] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi "Analysis and implementation of computer program renewal in cluster computerss," in ACM SIGMETRICS Performance Evaluation Review, vol. 29, no. 1. ACM, 2001, pp. 62–71.

[13] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of computer program aging in a web server," Reliability, IEEE Transactions on vol. 55, no. 3, pp. 411–420, 2006.

[14] J. Alonso, J. Torres, J. L. Berral, and R. Gavalda, "Adaptive online computer program aging prediction based on machine learning," in Dependable Conference on. IEEE, 2010, pp. 507–516.

[15] Y. F. Jia, L. Zhao, and K.Y. Cai, "A nonlinear approach to modeling of computer program aging in a web server," in Computer program Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific. IEEE, 2008, pp. 77–84.

[16] P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. Lyu, "An automatic framework for detecting and characterizing performance degradation of computer program computerss," Reliability, IEEE Transactions on, vol. 63, no. 4, pp. 927–943, 2014.

[17] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of computer program aging," in Computer program Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on. IEEE, 1998, pp. 283–292.

[18] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu "Computer program aging and multifractality of space resources," in 2003 33rd Annual IEEE/IFIP International Conference on Dependable Computerss and Networks (DSN). IEEE Computer Society, 2003, pp.721.

[19] J. Araujo, R. Matos, P. Maciel, R. Matias, and I. Beicker, "Experimental evaluation of computer program aging effects on the eucalyptus cloud computing infrastructure," in Proceedings of the Middleware 2011 Industry Track Workshop. ACM, 2011, p. 4.

[20] Rivalino Matias Jr., Bruno Evangelista Costa, and Autran Macedo, "Monitoring Space-Related Computer program Aging: An Exploratory Study" ,IEEE, 2012.

[21] Vols. Domenico Cotroneo, and Roberto Natella, Monitoring of Aging Computer program Computerss affected by Integer Overflows, IEEE, 2012.

[22] Autran Macêdo, Taís B. Ferreira, and Rivalino Matias Jr, "The Mechanics of Space-Related Computer program Aging," IEEE ,2011.

[23] Lei Cui, Bo Li, Jianxin Li, James Hardy, and Lu Liu, "Computer program Aging in Simulated Environments: Detection and Prediction," IEEE, 2012.

[24] Kehua Su, Hongbo Fu, Jie Li, and Dengyi Zhang, "Computer program Renewal in Virtualization Environment," IEEE, 2011.

[25] Kenichi Kourai, and Shigeru Chiba, "Fast Computer program Renewal of Simulated Machine Monitors," IEEE, Vol 8, No 6, 2011.

[26] Fumio Machida, Dong Seong Kim, Jong Sou Park, and Kishor S. Trivedi, " Toward Optimal Simulated Machine Placement and Renewal Scheduling in a Simulated Data Center," IEEE, 2008.

[27] Kenichi Kourai, and Shigeru Chiba, "A Fast Renewal Technique for Server Consolidation with Simulated Machines, " IEEE, 2007.

[28] Fumio Machida, Jianwen Xiang, Kumiko Tadano, and

Yoshiharu Maeno, "Combined Server Renewal in a Simulated Data Center.

[29] Aye Myat Myat Paing, and Ni Lar Thein, " High Availability Solution: Resource Usage Management In Simulated Computer program Aging, Aye Myat Myat Paing, and Ni Lar Thein, " High Availability Solution: Resource Usage Management In Simulated Computer program Aging.

[30] Thandar Thein, Sung-Do Chi, and Jong Sou Park," Availability Analysis and Improvement of Computer program Renewal Using Virtualization," Economics and Applied Informatics, Years XIII, 2007.

[31] Garg, S., Van Moorsel, A., Vaidyanathan, K., and Trivedi, K. 1998b. A methodology for detection and estimation of computer program aging. In Computer program Reliability Engineering, 1998. Proc. Ninth Int'l. Symp.

[32] Sullivan, M. and Chillarege, R. 1991. Computer program Defects and Their Impact on Computers Availability—A Study of Field Crashs in Operating Computerss. In Fault-Tolerant Computing, 1991. FTCS-21. Digest of Papers., Twenty-First International Symposium. IEEE, 2–9.

[33] Grottke, M., Li, L., Vaidyanathan, K., and Trivedi, K. 2006. Analysis of computer program aging in a web server. Reliability, IEEE Transactions on 55, 3.

[34] Carrozza, G., Cotroneo, D., Natella, R., Pecchia, A., and Russo, S. 2010. Space leak analysis of mission-critical middleware. Journal of Computerss and Computer program 83, 9, 1556–1567.

[35] Cotroneo, D., Orlando, S., Pietrantuono, R., and Russo, S. 2011b. A measurement-based ageing analysis of the jvm. Computer program Testing Verification and Reliability.

[36] Cotroneo, D., Natella, R., Pietrantuono, R., and Russo, S. 2010. Computer program aging analysis of the linux operating computers. In Computer program Reliability Engineering (ISSRE), 2010 IEEE 21st Int'l. Symp.

[37] Weimer, W. 2006. Exception-handling bugs in java and a language extension to avoid them. Lecture Notes in Computer Science 4119 LNCS, 22–41.

[38] Zhang, H., Wu, G., Chow, K., Yu, Z., and Xing, X. 2011. Detecting resource leaks through dynamica mining of resource usage patterns. In Dependable Computerss and Networks Workshops (DSN-W), 2011 41st Int'l. Conf.

[39] Bobbio, A. and Sereno, M. 1998. Fine grained computer program renewal models. In Computer Performance and Dependability Symposium, 1998. IPDS'98. Proceedings. IEEE International. IEEE, 4–12.

[40] Cassidy, K., Gross, K., and Malekpour, A. 2002. Advanced pattern recognition for detection of complex and Networks, 2002. Proc. Int'l. Conf.

[41] Gama, K. and Donsez, D. 2008. Service coroner: A diagnostic tool for locating osgi stale references. EUROMICRO 2008 - Proceedings of the 34th EUROMICRO Conference on Computer program Engineering and Advanced Functions, SEAA 2008, 108–115.

[42] Sousa, P., Bessani, A., Correia, M., Neves, N., and Verissimo, P. 2010. Highly available intrusion tolerant on 21, 4, 452 –465.

[43] Tai, A., Tso, K., Sanders, W., and Chau, S. 2005. A performability-oriented computer program renewal framework for distributed functions. In Dependable Computerss and Networks, 2005. Proc. Int'l. Conf.

[44] Valdes, A., Almgren, M., Cheung, S., Deswarte, Y., Dutertre, B., Levy, J., Saidi, H., Stavridou V., and Uribe, T. 2003. An architecture for an adaptive intrusion-tolerant server. Lecture Notes in Computer Science (including subseries Lecture Notes.

[45] Cox , B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong,A., and Hiser, J. 2006. N-variant computerss: a secretless framework for security through diversity. In Proceedings of the 15th conference on USENIX Security.

[46] Roeder, T. and Schneider, F. 2010. Proactive obfuscation. ACM Transactions on Computer Computerss (TOCS) 28, 2, 4.

[47] Nguyen, Q. and Sood, A. 2009. Quantitative approach to tuning of a time-based intrusion-tolerant computers architecture. In Proc. 3rd Workshop Recent Advances on Intrusion-Tolerant Computerss. 132–139.

[48] Aung, K., Park, K., and Park, J. 2005. A model of its using cold standby cluster. Lecture Notes in Computer AUNG, K., PARK, K., AND PARK, J. 2005. A model of its using cold standby cluster. Lecture Notes in Computer Science 3815 LNCS, 1–10.

[49] Nagarajan, A. and Sood A., "SCIT and IDS architectures for reduced data ex-filtration", DSNW, 2010, Dependable Systems and Networks Workshops, Dependable Systems and Networks Workshops 2010, pp. 164-169, doi:10.1109/DSNW.2010.5542601.

[50] Grottke, M., Matias, R., and Trivedi, K. 2008. The fundamentals of computer program aging. In Computer program Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.

[51] Macedo, A., Ferreira, T., and Matias, R. 2010. The mechanics of space-related computer program aging. In

Computer program Aging and Renewal (WoSAR), 2010 IEEE Second Int'l. Workshop on.

[52] Matias, R., Barbetta, P., Trivedi, K., and Filho, P. 2010a. Accelerated degradation tests applied to computer program aging experimentations. Reliability, IEEE Transactions on 59, 1.

[53] Shereshevsky, M., Crowell, J., Cukic, B., Gandikota, V., and Liu, Y. 2003. Computer program aging and multifractality of space resources. In Dependable Computerss and Networks, 2003. Proc. 2003 Int'l. Conf.

[54] Magalhaes, J. and Silva, L. 2010. Prediction of performance anomalies in web-functions based-on computer program aging scenarios. In Computer program Aging and Renewal (WoSAR), 2010 IEEE Second Int'l. Workshop on.

[55] Zhao, J. and Trivedi, K. 2011. Performance modeling of apache web server affected by aging. In Computer program Aging and Renewal (WoSAR), 2011 IEEE Third International Workshop on. 56 –61.

[56] A. Andrzejak and L. Silva, "Using machine learning for non intrusive modeling and prediction of computer program aging," in Network Operations and Management Symposium, 2008. NOMS 2008. IEEE IEEE, 2008, pp. 25–32.

[57] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Computer program aging analysis of the linux operating computers," in Computer program Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on IEEE, 2010, pp. 71–80.

[58] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds in IEEE DSN, 2013.

[59] P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. Lyu, "An automatic framework for detecting and characterizing performance degradation of computer program computerss," Reliability, IEEE Transactions on vol. 63, no. 4, pp. 927–943, 2014.

[60] M. U. Ahmed and D. P. Mandic, "Multivariate multiscale entropy: A tool for complexity analysis of multichannel data," Physical Review E, vol. 84, no. 6, p. 061918, 2011.

[61] L. Cao, A. Mees, and K. Judd, "Dynamics from multivariate time series," Physica D: Nonlinear Phenomena, vol. 121, no. 1, pp. 75–88,, 1998.

[62] J. F. Cadima and I. T. Jolliffe, "Variable selection and the interpretation of principal subspaces," Journal of agricultural, biological, and environmental statistics, vol. 6, no. 1, pp. 62–79, 2001.

[63] http://www.eweek.com/cloud/slideshows/nine-common-reasons-cloud-systems-crash.html