# Granger Causality-aware Prediction and Diagnosis of Software Degradation

Pengfei Zheng*†, Yangfan Zhou†, Michael R.Lyu†‡, and Yong Qi*

*Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China
†Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong
p.f.zheng.phd@stu.xjtu.edu.cn, {yfzhou, lyu}@cse.cuhk.edu.hk, qiy@mail.xjtu.edu.cn

*Abstract*—**Software that continuously runs over a long period of time has been frequently reported encountering "gradual degradation" issues. As time progresses, software tends to exhibit degraded performance, deflated capacity, exhausted physical resource or deteriorated QoS (Quality of Service). Different from transient software anomalies, this issue is a chronic degrading process and usually persists until the software is eventually unavailable. We name it "Software Degradation" or "Degradation" for short. In this paper, we propose a framework GVAR, utilizing Granger Causalities to predict and diagnose software degradation. GVAR is evaluated via an 8-day experiment on a VoD (Video on Demand) platform Helix-Serv. The experimental results show that GVAR can predict the TTF (Time to Failure) of degraded software in an accuracy of 80.1%, remarkably outweighing the widely used ARMA and Sen's Slope Estimator approaches. Moreover, GVAR can guide diagnosing the potential root cause of degradation issues.**

*Keywords*-**Performance Degradation; Causality Analysis; Granger Causality Test; Vector Auto-Regression;**

## I. INTRODUCTION

Software degradation was initially discovered and reported from multi-user telecommunication systems at AT&T [1]. As time progresses, the communication platform will enter a "failure-prone" state. That is, even not within the time-frame of peak workloads, it became inclined to lose ongoing calls. Moreover, the response time for telephonic connect was significantly prolonged and highly possible to be denied eventually. In most cases, after the system maintainers halt the "crippled" platform and enforce it to reboot, the platform could resume to normal conditions and its service quality could recover as well. Through extensive investigation, researchers found this phenomenon highly resembles the process of "physical-fitness degradation". Thus in this sense, we correspondingly name it "Software Degradation".

Software degradation is usually an accumulative process caused by activation of faults that reside in software, especially those faults related to memory management, resource management, concurrency control, fragmentation, round-off error and data corruption. For example, the software degradation problem of JVM is validated most related to memory leak [7]. Accumulation of memory leaks will gradually exhaust the available memory of JVM and significantly degrade JVM performance in 3 ways: (1) the frequency of GC will become higher and higher due to shortage of memory and excessive GC will bring great burden to system throughput (2) long-term accumulation of leaked objects (objects that are useless but still referenced) will lead sever memory fragmentation in JVM heap and requests for allocating continuous (e.g. an integer array of 1MB) area is very probable to fail (3) JVM may crash with an "Out of Memory" error when all available memory is depleted.

To counteract software degradation, "Software rejuvenation" [2] is an effective anti-degradation approach to assure high availability for software systems. Rejuvenation usually involves stopping the running software occasionally and "cleaning" its internal state. The "cleaning" measures usually include: (1) garbage collection to clean unreferenced objects (2) flushing kernel PCBs to clean orphan and zombie processes (3) unlocking unreleased mutexes/spinlocks long enough (4) eliminating processes which are resource-hogs (5) restarting the components that with no response (6) rebooting the VMs in virtualized environment (7) and halting and restarting a staggering physical node. However, there are two problems, which are decisive to the cost and effectiveness of rejuvenation.

*Problem 1: At what time to execute rejuvenation actions*? Software rejuvenation must be executed prior to the time point when the system degrades to be unavailable (usually crash or hang). Thus for a running system, we must in advance predict the "unavailable deadline" caused by degradation and schedule rejuvenation actions proactively.

*Problem 2: What kind of rejuvenation actions to schedule*? The rejuvenation measures must be selected in accord with the root cause of software degradation. Otherwise, the rejuvenation will be in vain and waste computing resources.

To answer these two questions, we propose a framework *GVAR* (based on *Granger Causality Analysis* and *Vector Auto-Regressive Model*). The contributions of GVAR can be briefly summarized in three perspectives.

*Contribution 1- Detecting software degradation via the monitored "health conditions" of a target software system.* In GVAR, we deploy tracer to monitor performance (e.g., response time, throughput), resource utilizations (e.g., CPU utilization, Working-set size), service quality (e.g., video streaming bandwidth) and other relevant system metrics to detect whether there exists degrading trend within the monitored "health data". To detect degrading trends of "health data", we design a *Modified Cox-Stuart Test* for statistical detection of software degradation.

*Contribution 2- Modeling and forecasting the performance, resource utilizations, and QoS of a target system.* According to results of forecasting, TTF (Time to Failure) or TTE (Time to Exhaustion of Resources) can be estimated and the "deadline" for rejuvenation can be determined. To model and predict the TTF/TTE from monitored metrics, we design a Granger Causality augmented *Vector Auto Regressive Model* (*VAR*)*,* which outperforms the popular performance modelling techniques ARMA (introduced by Grottke [5]) and Sen's Slope Estimator (introduced by Garg [4]). (Machine Learning models are usually inappropriate for fast, automatic prediction of system performance, since feature selection or over-fit tuning usually requires human heuristics and the training phase is generally time-consuming. Thus ARMA model is prevailing since it makes a much better trade-off between accuracy and complexity and does rely on human tricks.)

*Contribution 3- Diagnosing the potential root causes of software degradation.* From the monitored "software health" metrics that are detected degrading trends, we utilize *Multivariant Granger Causality Analysis* to construct a causality map between these "degraded" software metrics. After that, we apply *Pearce-Kelly Topological Sort Algorithm* to transform the causality map into a causality tree and then back-traverse the tree to identify the root cause of the degradation issue. In our 8-day experiment on a VoD platform Helix-Serv, we have successfully identified one potential root cause of the degradation problem of Helix-Serv.

The rest of this paper is organized as follows: In Section II, we review some related work contributing to software degradation analysis. In Section III and Section IV, we respectively introduce Multivariate Granger Causality Test and Vector Auto-Regression Model. In Section V, software health monitoring is explained. In Section VI, degradation detection and degradation diagnosis are introduced. In Section VII, we evaluate modelling and prediction accuracy of GVAR. Finally in VII, we conclude this paper.

## II. Related Work

Current software degradation analysis can be divided into two categories: the analytical model-based methodologies and the measurement based-methodologies. In general, related analytical models contain Markov process [2], semi-Markov process, semi-Markov reward process, hidden-Markov process, stochastic petri net and so on. The basic idea of analytical model-based methodologies is to construct a stochastic process to model of the state transition of target systems. Usually the primary contributions are to solve these models for determining the optimal rejuvenation strategy. The optimal rejuvenation strategy will maximize system availability and minimize the cost due to outage.

Measurement-based methodologies lay particular emphasis on statistical or machine learning-based approaches. Performance metrics or resource utilizations are collected periodically during operational monitoring of the target system. The collected multi-dimensional data (that corresponds to multiple parameters) can be seen as features reflecting the health condition of the system. When it runs over a certain period, statistics can be estimated from the collected data to test whether degrading trend exists. The data collected can also be used to train measurement-based models, for example ARMA [5], linear regression model [4], M5P [6], MSET and so on. Then the models can forecast performance (or resource usage) in the near future, or decide whether degradation occurs (treat this as a binary classification problem).

In [5], Grottke et al. studied the development of resource usage in an Apache web server while subjecting it to an artificial workload. They first collected data on several system resource usage & activity parameters. Non-parametric statistical methods were then applied toward detecting & estimating trends in the data sets. Finally, they fit time series models (ARMA model, Auto Regressive & Moving Average model) to the three parameters collected: (1) response time (2) throughput (3) used swap space. The ARMA model could adequately predict the future resource usage of the Apache web server in this paper. Based on the models employed, proactive management techniques like software rejuvenation triggered by actual measurements can be built.

The ARMA models used in [5] are completely unrelated and mutual influences between performance (or resource) metrics are not taken seriously. In this paper, we will improve the ARMA model by our GVAR in three ways: (1) Detecting the statistical causalities between performance (or resource) metrics, via Granger Causality Test. (2) Taking advantage of causalities between performance (or resource) parameters to model and forecast the dynamic progress of software degradation more accurately (3) Making use of the causalities between performance (or resource) parameters to guide root cause analysis

## III. GRANGER CAUSALITY ANALYSIS

In many fields of science or engineering, a problem of great importance is to detect whether there exist unidirectional or bidirectional causal relations among a group of variables. Causal relations have diverse definitions in different situations. Here we adopt the concept of causal relation (or *Causality*) defined by Wiener: Given two time series A and B that are measured synchronously, if the prediction accuracy of A can be improved by incorporating the information of B, B is statistically causal to A. In another way, it means that there exists causality from B to A. Clive Granger applies Wiener's definition of causality and proposes a method using linear regression model to detect the causality between two time series. Meanwhile the magnitude of causality can be measured. This method is widely used in economics and neurosciences, called *Granger Causality Test (GCT)*. It was improved by Gewek and Seth [16] to extend the bivariate GCT to multivariate GCT, and our starting point is to utilize multivariate GCT to analyze the causal relations among the monitored performance (or resource) parameters.

Firstly, we provide some brief introductions to bivariate (pairwise) GCT and the later the extension of multivariate GCA will be demonstrated. Suppose there are two different time series X(t) and Y(t), and we define:

$$X(t) = \hat{X}(t) + \varepsilon_1(t), where\, \hat{X}(t) = \sum_{i=1}^{p1} a_{1i} X(t-i) + \sum_{i=1}^{p2} a_{2i} Y(t-i) \quad (1)$$

$$X^*(t) = \hat{X}^*(t) + \varepsilon_2(t), where\, \hat{X}^*(t) = \sum_{i=1}^{p3} b_{1i} X(t-i) \quad (2)$$

Equation (1) is called unrestricted model and Equation (2) is called restricted model. p1, p2, p3 are the maximal number of lagged observations incorporated into the model, and they are all called *Lag Order* for endogenous or exogenous variables. $\varepsilon_1(t)$ and $\varepsilon_2(t)$ are the regression residuals respectively for X(t) and $X^*(t)$. If the variance of $\varepsilon_1(t)$ is less than the variance of $\varepsilon_2(t)$ (that is to say by incorporating the information of Y(t), the variance of $\varepsilon_2(t)$ is reduced to $\varepsilon_1(t)$) , it is claimed that Y(t) is the Granger Cause of X(t) (Y(t)→X(t) in abbreviation).The magnitude of causality from Y(t) to X(t) can be measured by the log ratio of the residuals:

$$G_{Y(t)\rightarrow X(t)} = ln \frac{var[\varepsilon_2(t)]}{var[\varepsilon_1(t)]} \quad (3)$$

For test whether the causality from Y(t) to X(t) is statistically significant, we use:

$$F_{Y(t)\rightarrow X(t)} = \frac{(RSS_2 - RSS_1)/p}{RSS_1/(T-2p-1)},\; F_{Y(t)\rightarrow X(t)} \sim F(p, T-2p-1) \quad (4)$$

$RSS_1$ is the sum of square for time series $\varepsilon_1(t)$ and $RSS_2$ is the sum of square of time series $\varepsilon_2(t)$. The larger $F_{Y(t)\rightarrow X(t)}$, the more probable Y(t) →X(t). And F-test can be used to test whether or not Y(t)→X(t) because $F_{Y(t)\rightarrow X(t)}$ follows F-distribution.

Importantly, bivariate Granger causality test is easy to generalize to the multivariate (conditional) case in which the causality from Y(t) to X(t) is tested in the context of multiple additional variables Z(t), P(t), R(t) . . . (Geweke [15]). In this case, Y(t)→X(t) if knowing Y(t) reduces the variance in X(t)'s prediction error when all other variables Z(t), P(t), R(t) . . . are also included in the regression model. To illustrate, for a system of three variables (1,2,3), we represent the noise covariance matrix of the unrestricted model as:

$$\rho = \begin{bmatrix} var(\varepsilon_{1U}) & cov(\varepsilon_{1U},\varepsilon_{2U}) & cov(\varepsilon_{1U},\xi_{3U}) \\ cov(\varepsilon_{2U},\varepsilon_{1U}) & var(\varepsilon_{2U}) & cov(\varepsilon_{2U},\varepsilon_{3U}) \\ cov(\varepsilon_{3U},\varepsilon_{1U}) & cov(\varepsilon_{3U},\varepsilon_{2U}) & var(\varepsilon_{3U}) \end{bmatrix} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} & \Sigma_{13} \\ \Sigma_{21} & \Sigma_{22} & \Sigma_{23} \\ \Sigma_{31} & \Sigma_{32} & \Sigma_{33} \end{bmatrix} \quad (5)$$

where all $\varepsilon_{iU}(t)$ (i=1, or 2, or 3) are the residuals from the unrestricted linear models including all variables. A useful partition of the matrix is given by the second equality. For n variables, there are n unrestricted models and n restricted models, with each restricted model omitting a different predictor variable. For example, the noise covariance matrix of the restricted model omitting variable 2, with its partition, is:

$$\rho = \begin{bmatrix} var(\varepsilon_{1R}) & cov(\varepsilon_{1R},\varepsilon_{3R}) \\ cov(\varepsilon_{3R},\xi_{1R}) & var(\varepsilon_{3R}) \end{bmatrix} = \begin{bmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{bmatrix} \quad (6)$$

, where all $\varepsilon_{iR}(t)$ are estimated from the restricted model omitting variable 2. The Granger causality from variable 2 to variable 1, conditioned on variable 3, is given by:

$$F_{2\rightarrow 1|3} = ln \frac{\rho_{11}}{\Sigma_{11}} \quad (7)$$

$F_{2\rightarrow 1|3}$ also follows F-distribution, therefore through F-test Granger Causality from 2 to 1 for a group of 3 variables can be tested. This situation of three variables can be easy to extend to multiple variables and this is the *MGCT (Multivariate Granger Causality Test).*

## IV. Vector Auto-Regressive Model

VAR (Vector Auto-Regressive) model was introduced by Sims [14] to Econometrics and VAR has been acknowledged the most successful and flexible multivariate model in time series analysis, to describe and analyze the dynamics of complex systems. The mathematical formulation of VAR is: where ε(t) is discrete white noise, and y(t) represents response variables of a dynamic system and x(t) represents the variables have significant granger causalities to y. In this paper, the response variables are the parameters indicating system performance or resource utilizations.

$$\begin{aligned} y(t) = {} & \phi_{0,1}y(t-1)+\phi_{0,2}y(t-2) + \cdots + \phi_{0,p}y(t-p)+ \\ & \phi_{1,1}x_1(t-1)+\phi_{1,2}x_1(t-2) + \cdots + \phi_{1,p}x_1(t-p)+ \\ & \phi_{2,1}x_2(t-1)+\phi_{2,2}x_2(t-2) + \cdots + \phi_{2,p}x_2(t-p)+ \\ & \vdots \\ & \phi_{q,1}x_q(t-1)+\phi_{q,2}x_q(t-2) + \cdots + \phi_{q,p}x_q(t-p)+ \\ & \varepsilon(t) \\ & ,\ t=1, 2, 3, \dots, T \end{aligned} \tag{8}$$

From Equation (8), it shows that each endogenous variable in VAR is not only in an "Auto-Regressive" fashion but also influenced by other variables. The elements in coefficient matrix Φ represent the magnitude of influence (or Causality). When an element $\varphi_{mn}$ in Φ is very close to zero, it means that the corresponding causality can be neglected. All the coefficients can be estimated by LMS (Least Mean Square) estimation method.

In [5], Grottke estimates three independent ARMA models to model and predict the dynamic progress of system performance and resource utilizations (i.e. response time, throughput and used swap space). These three independent ARMA models constitute a simultaneous linear equations-based model to describe and model the process of software degradation. The disadvantage of this methodology is that it ignores the mutual influence (i.e. causalities) between performance or resource parameters. In this paper, we will show that GVAR takes advantages of these causalities, which can more accurately model and predict the process of software degradation, and can help diagnose the root cause of software degradation (cf. Section VII).

## V. Software Degradation Monitoring

Helix Server is a universal streaming media delivery platform with industry-leading performance, integrated content distribution, user authentication, Web services support, and native delivery of RealMedia. Helix Server is developed and distributed by the RealNetworks Enterprise, a renowned digital media service corporation. Helix Server supports multiple On-Demand-Streaming protocols such as RTSP (real-time streaming protocol), RTP (real-time transport protocol), HTTP (hyper-text transport protocol) and MMS (multimedia messaging service). Report Server Statistics (RSS) is a subsystem of Helix Server, which can periodically write performance statistics into rotated logs. We implement configurative scripts and programs to extract and parse the performance statistics into a database. RSS supports more than 50 distinct performance metrics for system monitoring and troubleshooting, however, only 24 of them (performance metrics in categories of system resource utilization and service quality of VOD) are relevant to software degradation analysis (most of the irrelevant statistics are associated with network multicast/broadcast protocols). The monitored metrics of our experiments are listed below (within one RSS log period):

A. *Time Report Category:* (1) *Uptime (TT)* indicates the elapsed time since the server booted.
B. *QoS Category*: (2) *Total Bandwidth Output (TBO)* indicates the actual rate at which Helix Server is delivering outgoing streams. (3) *Total Bandwidth Subscribed (TBS)* means the cumulative bandwidth added by all Subscribe Request issued from client media players. If a clip is 512Kbps encoded, each client accessing this clip issues a 512Kbps Subscribe Request to inform Helix Server to allocate appropriate output bandwidth (referenced 512Kbps) for transmitting streams of this clip. (4) *Subscription Fulfillment (SF)* equals to Total Bandwidth Output divides Total Bandwidth Subscribed, a percentage value to measure the general playback quality of client media players.
C. *CPU Usage Category*: (5) *User Time (UT)* indicates the percentage of processor time slots allocated to process rmserver.exe (Helix Server), not counting kernel time. (6) *Privilege Time (PT)* indicates the percentage of time slots used by the OS kernel on behalf of Helix Server.
D. *Memory Usage Category*: (7) *Working Set (WS)* means the amount of physical memory allocated to Helix Server. (8) *Cache Hit Ratio (CHR)* indicates percentage of memory operations that are carried out using the server's memory cache. The cache is defined and managed by Helix Server and unrelated to L1/L2 hardware cache). (9) *Mallocs (MA)* is the number of calls to Helix Server's shared memory block allocator. (10) *Frees (FR)* is the number of calls to the server's shared memory block

deallocator. (11) *Page Faults (PF)* is the number of virtual memory pages failing to be addressed in RAM and must either be mapped immediately in memory or retrieved from the pagefile in hard drive. (12) *Memory Allocated Overhead (MAO)* is the amount of memory allocated but not in use.

E. *Client Connection Category*: (13) *Total Subscribers (TS)* is the number of players connected online. (14) *New Subscribers (NS)* is the new player connections since the start of this RSS interval.

F. *Network Transmission Category*: (15) *Packets (PK)* is the number of network packets written to Network Interface Controller within this RSS Period. (16) *NoBufs (NB)* is the number of ENOBUFS erros returned form UDP sockets. (17) *WouldBlocks (WB)* is the number of blocked packet in queue. (18) *Resend (RS)* is the number of packets resent. (19) *Accepts (AC)* the number of incoming socket connections.

G. *Mutex Collisions Category*: (20) *Mutex Collisions (MC)* (21) *CPU Spinning (CPS)*

*H. Handle Category*: (22) *Opened Files (OF)* 23 *Concurrent Threads (CT)*

*I. Process Activity Category*: (24) *Main Loop Iterations (MLI)*

The testbed is composed of a VOD server (Helix-Serv 5.3.0) and three client terminals connected via a LAN. Specific hardware configuration is as follows: Intel Xeon 8 core 1.6GHZ Processor, plus 4GB Memory plus 1Gbps Network Adapter for VOD server and Intel Pentium4 2.59GHZ plus 1GB plus 100Mbps for each client machine. Helix server supports a great many streaming media formats such as .rm, .ra, .rmvb, .mov and .mpeg. During our experiments, we deployed 160 .rmvb files on Helix-Serv. These files were the popular movies in the recent years within our campus. We requested all the movies and the popularities that they were played from the VOD Center on our campus. Unfortunately we could not get their server access logs for workload replay. Thus, we implement a workload generator (i.e., Clients Simulator), adopting RTP/RTSP protocol for generating user connections concurrently. The simulated users were created and removed on the client side periodically to simulate the diurnal pattern of real workloads. Each user requests media files on Helix server according to a designated probability distribution. The probability density of each file is positively related to the playback frequencies of a movie in our campus. Preliminary server capacity estimation was also executed as a preparing work. Each Client Simulator will generate no more than 200 clients at a time to prevent overload. The RSS monitoring period is set to 1 minute and the log rotation duration is set for a week. The entire experiments continued for 8 days (defacto 12356 minutes) and eventually rmserver.exe is exceptionally crashed. From our data collection procedures, we totally obtain 12356 consecutive data points for each of the 24 performance metrics listed.

## VI. Degradation Detection and Diagnosis

### A. *Software Degradation Detection*

The most typical syndrome of software degradation is gradual deterioration of system performance, gradual depletion of system resource, or gradual reduction of QoS, all of which can be embodied by long-term descending or ascending trends of specific performance, resource utilization, or QoS metrics. By this means, trend detection, sometimes called statistical test for trend utilizes statistical hypothesis test to examine whether metrics in representation of time series data generally exhibit a chronic increasing or decreasing trend. More specifically, we devise a modified Cox-Stuart (cf. Fig.1), to detect the "degrading trend" of all metrics. Eventually among all the 24 metrics, with statistical significance level 0.05 and 12356 testing samples for each of them, 10 metrics are detected exhibiting degrading trends. They are: (1) PK with statistical significance (p-value) 4.3e-4, (2) UT with 1.7e-5, (3) PT with 2.6e-5, (4) CHR with 8.2e-3, (5) PF with 1.01e-3, (6) MLI with 3.7 e-5, (7) TBO with 2.8e-4, (8) TBS with 9e-5, (9) NS with 0.0067, and (10) MAO with 3.4e-4.

### B. *Mining Causalities Between Metrics with Degradation Trend*

As introduced in Section III, we apply Multivariate Granger Causality Test (MGCT) to inspect the causalities between metrics that exhibit significant software degradation phenomenon. There are two points to be noticed: (1) Causality is not identical to correlation (i.e. Pearson Correlation Coefficient). Correlation measures non-directional, linear relation between two variables while causality is directional dependency not confined to just linear relation. (2) MGCT is a joint probability approach, not identical to pairwise Granger causality test for multiple variables. The result of MGCT for the 10 aged performance, service quality or resource metrics is illustrated in Fig. 2. Non-aged metrics are not considered for MGCT because we believe a pattern that "aged metric A" causes "non-aged metric B" and B causes another "aged metric C" is unlikely. In this figure, bidirectional arrows indicate bidirectional causalities between two metrics, unidirectional arrow indicate A Granger causes B where A is the arrow-tail and B is the arrowhead.

**Modified Cox-Stuart Test**

```
T= [T_1,T_2,...,T_n] is the input time series for degrading trend.
α is the significance level for detecting degrading trend

1: Divide T into two halves:
   If n is an even number,
      $T_{first}=[T_1,T_2,...,T_{n/2}]$, $T_{second}=[T_{n/2+1},T_{n/2+2},...,T_n]$, $n^*=n/2$;
   If n is an odd number,
      $T_{first}=[T_1,T_2,...,T_{(n-1)/2}]$, $T_{second}=[T_{(n+1)/2},T_{(n+1)/2+1},...,T_n]$,
      $n^*=(n-1)/2$.
2: If n is an even number, pair each element $T_i$ (i=1, 2... n/2) in $T_{first}$
   with $T_{i+n/2}$ in $T_{second}$ :
      $(T_1,T_{n/2+1}), (T_2,T_{n/2+2}), ..., (T_{n/2},T_n)$;
   If n is an odd number, pair each element $T_i$ (i=1, 2... (n-1)/2) in
   $T_{first}$ with $T_{i+(n-1)/2}$ in $T_{second}$ :
      $(T_1,T_{(n+1)/2}), (T_2,T_{(n+1)/2+1}), ..., (T_{(n-1)/2},T_n)$.
3: For the ith pair, let $T_{i1}$ represents the first element and $T_{i2}$
   represents the second element then compute $S_i=sgn(T_{i2}-T_{i1})$ for
   each pair, where i=1,2,...,n/2 when n is an even number and
   i=1,2,..., (n-1)/2 when n is an odd number.
4. Synthetize all $S_i$ into a cumulative statistic $S=\sum S_i$.
5: Transform S into a normalized statistic z, which follows N(0,1):
      $z=(S+1)/sqrt(n/2)$.
6: Compute one sided p-value of z according to cumulative
   distribution function of standard normal distribution.
7: If p-value<α, degrading trend is detected, else there is no
   degrading trend.
```

Fig. 1 Modified Cox-Stuart Test

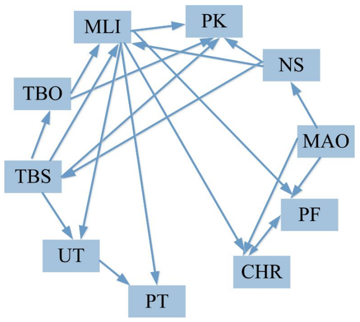

Fig. 2 Multivariate Granger Causality Graph

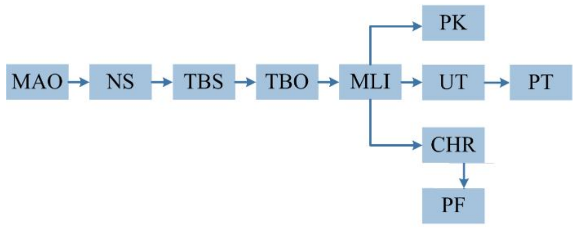

Fig. 3 Diagnosis Map based on Causality (DMC)

## C. Software Degradation Diagnosis based on MGCT

Even though multiple distinct performance metrics exhibit software degradation features, not all of the aged metrics are the root causes of software degradation. In our experiment, metrics affiliated to CPU usage, memory utilization, VOD service quality, network transmission, and player connection are all detected aged, but it is unlikely they all of simultaneously trigger and arouse the degradation process of software degradation. Thus to discriminate "superficial degradation" and "internal degradation" is crucial for diagnosis. Fortunately, MGCT graph provides us good opportunities but still needs some additional work: (1) Applying DFS (Depth-First-Search) for each node (metric) in MGCT graph, to detect whether there exists cycles. If discovered, eliminate the edge that forms this circle with least statistical significance of Granger causality test (i.e. with largest p-value). This procedure continues until all circles in this graph are eliminated and consequently the MGCT graph turns into a DAG (Directed Acyclic Graph). The temporal complexity is O(V!). (2) Applying Pearce-Kelly Topological Sort Algorithms [17] to find a linear ordering of all metrics, in complexity $O(V^2+E)$. (3) Eliminate all transitive causalities that is if $V_A$ is the precursor of $V_B$ after topological sort and $V_B$ is the precursor of $V_C$ and $V_A \rightarrow V_C$, then eliminate this causality, in complexity $O(V^2)$.

After topological sort, the *Diagnosis Map based on Causality* (*DMC*) is eventually yield by GVAR, with vertices representing abnormal metrics and the edges representing the magnitude of causality between the two metrics. The root (or roots) of DMC are the metric causing almost all others metrics to exhibit software degradation patterns, thus in statistical sense, the most probable root cause of degradation is inspected. The diagnosing result is shown in Fig. 3. From the figure, we can obtain some instructional knowledge: (1) PK, UT, PT, CHR and PF are on top level of the DMC, which are "external" degradation symptoms. (2) MLI, TBO, TBS are middle-level transitive metrics which propagate degradation. (3) MAO and NS are the "internal" degradation factors, which are high possibly the root cause of degradation.

### *D. Explanations on the Diagnosis*

Memory Allocated Overhead (MAO) represents memory allocated but not in used. The inflating trend of MAO implies two possibilities, memory leak and zombie clients (or zombie threads) of Helix Server. Since Working set (WS) does not exhibit depletion trend, thus zombie clients seem to be the most probable cause of degradation. Additional corroborative evidences are MAO causing decreasing of NS and transitively causing reducing of TBS, TBO and MLI. Since zombie clients (just like zombie processes conventionally in OS, occurs when streaming for clients exceptionally terminate but Helix Server is not aware this and still conserve the session state plus file cache for it) accumulate and occupy more and more available capacity of Helix Server, new arrival clients will more and more likely to be rejected, until Helix Server is fully saturated and stuck by inactive clients and finally crashed. Therefore, the total requested bandwidth (TBS) and reduced, causing actual output bandwidth decreased. Due to Helix Server becomes more and more inactive, thus main loop iterations (MLI) for admitting and handling video requests will be more and more infrequent, thus thereupon CPU time (UT plus PT), memory access (CHR plus PF) and network transmission (PK) of Helix Server all degrade as time progresses. In fact, during our experiment when Helix Server approaches the final crash point. CPU time degrades to no higher than five percent and all other metrics are also at extremely low level, which additional strengthen our analysis of zombie clients.

Unlike a speculative diagnosis derived from a single metric or multiple mutually independent metrics, the DMC provided by GVAR utilizes an evidence chain to diagnose the root cause of software degradation. Diverse degradation phenomena in our experiments seemingly coincide with the diagnosing result of GVAR, which is a good supporting case for GVAR.

## VII. Modelling and Predicting Software Degradation

GVAR utilizes MGCT to improve the explanatory variable selection of VAR model to achieve the augmented TTF/TTE prediction of degradation. For example, to model and predict MLI, the VAR model should only incorporate explanatory variable TBS, TBO and NS (according to MGCT, cf. Fig. 2). However, in comparison, classic VAR model must incorporate all other nine variables, even though they may have no relation with the degradation of MLI. This gratuitously increases model complexity and the risk of prediction inaccuracy.

In this section we evaluate the modelling and prediction accuracy of GVAR, together with Sen's Slope Estimator and ARMA models, over the 10 aged metrics detected in Section VI-A. Sen's Slope Estimator (introduced by Garg [4]) and ARMA model (introduced by Grottke [5]), are both well acknowledged techniques in modeling & prediction of software performance or resource usage. The modelling accuracy is quantified as fitness-of-fit (defacto Coefficient of Determination, Adjusted $R^2$) on the training data set (75% of the 12356 data points for every metric) and the prediction accuracy is quantified as the relative accuracy on TTE (time to exhaustion, defacto the crash time in our experiment). The optimal lag orders for GVAR and VAR are estimated according to BIC criterion through our automatic trial-and-error preliminary tests. GVAR incorporates the causal variables for a variable to model and predict, (e.g. incorporating main loop iterations for user-mode processor time and privilege processor time). Form the standpoint of information theory, extending information source can enhance accuracy of decision, thus theoretically GVAR is superior to those univaritate models. The comparative results are shown in Table I., which testifies that GVAR outperforms ARMA model and Sen's Slope Estimator in both modelling and predicting TTE/TTF of degradation.

## VIII. Conclusions

In this paper, we propose a framework GVAR, in combination of Multivariate Granger Causality Analysis and Vector Auto-Regression Model for diagnosis and TTF (Time to Failure) prediction of software degradation. Via an 8-days-long experiment on a Helix-Serv VoD platform, GVAR successfully identifies the root cause of a real degradation issue, by virtue of multi-variant causality analysis. Moreover, through evaluation, GVAR generally exhibit accuracies of 98.17% and 80.17%, respectively in modeling and forecasting the TTF/TTE of software degradation, and significantly outperforms classic methodologies such as Sen's Slope Estimator and ARMA.

## Acknowledgements

This work was conducted when the first author was a visiting student in Shenzhen Research Institute, The Chinese University of Hong Kong. The work was supported by the National Basic Research Program of China (973 Project No. 2011CB302603), the National Natural Science Foundation of China (Project No. 61100077), the Shenzhen Basic Research Program (Project No. JCYJ20120619152636275), and the Research Grants Council of Hong Kong (Project No. CUHK405/11). Yangfan Zhou is the corresponding author.

Table I. Evaluation Result for Software Aging Modeling and TTE/TTF Forecasting

| **Performance/QoS/Resource Usage Metrics** | **ARMA Model** | | | **Sen's Slope Estimator** | | **GVAR Model** | | |
|---|---|---|---|---|---|---|---|---|
| | Lag order for ARMA model | Adjusted $R^2$ for Modelling | Accuracy for TTE Prediction | Adjusted $R^2$ for Modelling | Accuracy for TTE Prediction | Lag order for VAR model | Adjusted $R^2$ for Modelling | Accuracy for TTE Prediction |
| PK | 7 | 0.972 | 0.765 | 0.726 | 0.698 | 6 | 0.991 | 0.747 |
| UT | 32 | 0.978 | 0.638 | 0.709. | 0.886 | 2 | 0.987 | 0.862 |
| PT | 34 | 0.984 | 0.624 | 0.635 | 0.755 | 2 | 0.997 | 0.851 |
| CHR | 16 | 0.943 | 0.823 | 0.627 | 0.683 | 2 | 0.973 | 0.806 |
| PF | 12 | 0.908 | 0.615 | 0.614 | 0.822 | 4 | 0.982 | 0.716 |
| MLI | 16 | 0.921 | 0.726 | 0.754 | 0.707 | 15 | 0.976 | 0.702 |
| TBO | 68 | 0.909 | 0.783 | 0.807 | 0.716 | 4 | 0.968 | 0.849 |
| TBS | 54 | 0.912 | 0.715 | 0.605 | 0.763 | 4 | 0.994 | 0.824 |
| NS | 2 | 0.928 | 0.931 | 0.684 | 0.676 | 1 | 0.986 | 0.953 |
| MAO | 9 | 0.937 | 0.741 | 0.725 | 0.752 | 9 | 0.963 | 0.761 |
| Average | 25 | 0.9392 | 0.7361 | 0.6863 | 0.7458 | 5 | 0.9817 | 0.8071 |

## REFERENCES

[1] A. Avritzer, "Monitoring Smoothly Degrading Systems for Increased Dependability," Empirical Software Engineering, vol. 77, pp. 59–77, 1997.

[2] K. Vaidyanathan, R. E. Harper, S. W. Hunter, K S. Trivedi, "Analysis and implementation of software rejuvenation in cluster systems," In Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and modeling of computer systems (SIGMETRICS 2001), pp. 62-71, 2001.

[3] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software Aging Analysis of the Linux Operating System," 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE 2010), pp. 71–80, Nov. 2010.

[4] S. Garg, A. van Moorsel, K. Vaidyanathan and K. S. Trivedi. "A Methodology for Detection and Estimation of Software Aging," in Proceedings of the 9th International Symposium on Software Reliability Engineering, pp. 282-292, 1998.

[5] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of Software Aging in a Web Server," IEEE Transactions on Reliability, vol. 55, no. 3, pp. 411–420, 2006.

[6] Javier Alonso, Jordi Torres, Josep L1. Berral, Richard Gavalda. "Adaptive online software aging prediction based on Machine Learning," in 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010).

[7] D. Cotroneo, S. Orlando, S. Russo, "Characterizing Aging Phenomena of the Java Virtual Machine," in 26th IEEE International Symposium on Reliable Distributed Systems , vol., no., pp.127,136, 10-12 Oct. 2007

[8] L. M. Silva, J. Alonso, and J. Torres, "Using Virtualization to Improve Software Rejuvenation," IEEE Transactions on Computers, vol. 58, no. 11, pp. 1525–1538, 2009.

[9] S. P. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco: Statistical diagnosis of chronic problems in large distributed systems," in 2012 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1–12, 2012.

[10] M. Gabel, A. Schuster, R. Bachrach, N. Bjorner, "Latent fault detection in large scale services," in 2012 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1-12, 2012.

[11] E. Marshall, "Fatal Error: How Patriot Overlooked a Scud," Science, pp. 1347, 1992.

[12] K.S. Trivedi, K. Vaidyanathan, K. Goseva, "Modeling and Analysis of Software Aging and Rejuvenation," in 33rd Annual Simulation Symposium, pp. 270, 2000

[13] S. Garg, A. Puliafito, M. Telek and K.S. Trivedi, "Analysis of Preventive Maintenance in Transaction Processing Systems," IEEE Transactions on Computers, 47(1), pp. 96–107, 1998.

[14] Sims, Christopher A. 1980, "Macroeconomics and Reality," Econometica, 48, pp. 1-48.

[15] Geweke. JF, "Measurement of linear dependence and feedback between multiple time series," Journal of American Statisitics Association 77(378):304-313, 1982.

[16] Seth A.K., "A Matlab toolbox for Granger causal connectiviey analysis," Journal of Neuroscience Methods,vol. 186, no. 2, 2010 pp.262-273.

[17] David J. Pearce and Paul H.J. Kelly, "A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs," ACM Journal of Experimental Algorithmics (JEA), volume 11, pages 1.7, 2007.