Monitoring and Mitigating Software Aging on IBM Cloud Controller System

Harish Sukhwani[1], Rivalino Matias Jr.[2], Kishor S. Trivedi[1], and Andy Rindos[3]

[1]Department of Electrical & Computer Engineering, Duke University, Durham, USA
[2]School of Computer Science, Federal University of Uberlandia, Uberlandia, Brazil
[3]IBM Corporation
hvs2@duke.edu, rivalino@ufu.br, ktrivedi@duke.edu, rindos@us.ibm.com

***Abstract*—As enterprises continue to move their workloads from traditional server-room environments to private cloud-based systems, there is an increasing desire and ability for companies like IBM to centrally monitor the systems on behalf of their customers to proactively help to mitigate any potential failure scenarios. In this paper, we investigate failures caused by software aging affecting an enterprise-class cloud controller system. We describe a service developed to continuously analyze the key system/application metrics from customer systems, identifies potential aging-related failure scenarios within the next two days, and generates a list of tasks for the development-operations team at IBM to mitigate the potential failures. To help the team prioritize the tasks, we propose a prioritization scheme to assign severity to such tasks. From our analysis of two months of offline data, we find that the tasks generated have a precision of around 0.80 and recall of 1, which means that our approach did not miss any aging-related failure event, with around 80% of the failure events being true.**

***Index Terms*—software aging, software rejuvenation, cloud computing, failure forecasting**

I. Introduction

Although it is increasingly popular for organizations to run their services on public cloud computing service providers such as Amazon Web Services (aws.amazon.com) and IBM Softlayer (www.softlayer.com), many organizations are still wary about moving all their services and data to the cloud due to security reasons, regulatory requirements and other issues [1]. Such organizations are deploying private clouds in their data-centers, and moving their workload from their traditional server-room environment to a well-managed cloud infrastructure [2]. We study the IBM PureApplication System [3], which is a cloud computing system in a box and can be readily deployed to create a private cloud environment in an enterprise.

Private cloud service comes with various benefits, however, it also comes with a higher cost, especially in terms of onsite management [4]. Thus, ensuring the operational efficiency of the private cloud infrastructure is of great importance nowadays. For ‘cloud in a box’ products such as PureApplication, IBM provides support services to help customer resolve issues raised by them. There is an increasing expectation from IBM to resolve issues more proactively and mitigate potential failure scenarios. To fulfil this expectation, a software development team at IBM PureApplication developed a comprehensive monitoring service that monitors hardware component failures, workload deployment failures and other existing failure scenarios in customer systems. This monitoring service is used by the development-operations (DevOps) team. It has helped reduce the number of service tickets and resolve issues quicker. We enhance this service to continuously analyze and mitigate potential software aging [5] issues in the customer systems.

In this paper, we describe a service developed to perform software aging analysis for key system/application metrics customized for each customer system, with the goal to identify potential aging-related failure scenarios within the next two days. This service generates tasks for the DevOps team with the necessary maintenance (e.g., rejuvenation) procedures. Taking the uncertainty of our prediction estimates into account, we design a scheme to assign a severity level to each task. We focus on software aging-related issues in the cloud controller node of the PureApplication System called PureSystems Manager, which plays a critical role in letting users deploying workloads, manage the system resources, maintaining the system state in the event of an outage.

II. IBM PureApplication System

IBM PureApplication System is a pre-configured, open platform for Platform as a Service (PaaS) solution that simplifies middleware deployment and management through pattern-based deployment model for on-premise data centers [6]. In a PureApplication system, workloads run as a pattern, which is essentially a collection of Virtual Machines (VM) that run collectively to deliver the required service. The pattern defines all the application configuration specifics, dependencies, and deployment related details.

In our research, we focus on the PureSystems Manager (PSM) node which performs various administrative tasks such as deploying the workload patterns, load-balancing workload patterns across compute nodes, monitoring the performance of patterns and handling failure scenarios at pattern-level. PSM plays a critical role from system dependability perspective. The major subsystems of the PSM node are shown in Figure 1. From the software aging perspective, at the application-level, we focus on processes belonging to API Servers and Workload Deployer (IWD) subsystems since they perform almost all the administrative tasks on the PSM.

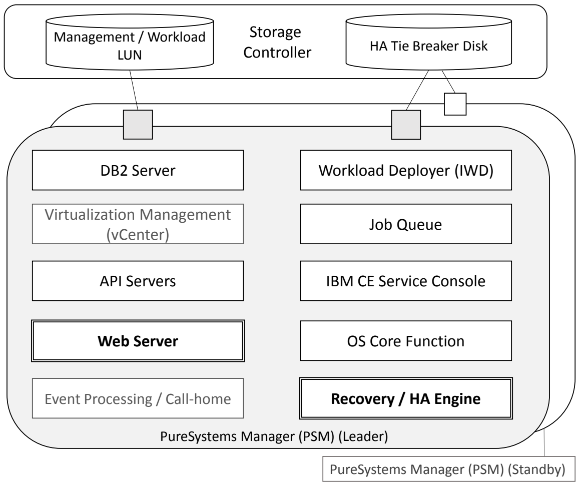


Fig. 1: Software Subsystems in PureSystems Manager (PSM)

PSM nodes are configured as a pair in warm-standby mode, providing redundancy at both hardware and software levels. For the active node, the operating system (OS) and all the software subsystems are running. For the standby node, the OS is booted up, and few software subsystems (shown in bold in Figure 1) are running continuously. In the case of active node failure, the standby node starts all the other subsystems of the PSM, some of which connect with the management logical unit number (LUN) on the storage node to sync its internal state with the internal state of the ex-active node, before taking over as the active node. Then the new active node attempts to bring up the now-standby node using various strategies (soft reboot, hard reboot, etc.). It also investigates any potential hardware failure scenarios and reports it on to the console for the system administrator.

III. Software Aging issues in PureApplication System

To understand the nature of software failures encountered in the field, we review the information collected in the failure reports and corresponding defect reports in the repository, spanning across five years. Due to the large number of defects in repository (10,000+), we first review the defects suggested by IBM software engineers and then search for reports containing keywords such as "OOM", "out of memory", "out of disk", "slowing","leak", "increa*", "decrea*", "exhaust", as suggested in [7]. Following are three representative issues of software aging that reoccurred frequently and motivated our research.

1. **Increasing usage of disk partitions (/var/log, others)**
During PSM's operation, partitions such as `/var/log` are filled up at a consistent rate. If the usage of partitions such as /var/log reaches 100%, it can lock up the process writing to the log file, causing the system to become unresponsive. Thus, uncontrolled growth in the disk usage can be considered as a software aging effect. Log files for the past one to two weeks are necessary to find the root-cause of any critical issues that could arise in the PSM. PSM runs the `logrotate`[1] utility as a nightly `cron` job that rotates the log files, compresses them and removes files that are older than two weeks. We experienced three potential issues with this. In situations where the log files were growing faster than expected, the disk filled up before the nightly logrotate job kicked in. In another case, the logrotate utility tried to copy and rotate a huge log file, but the partition did not have enough space to copy that file; hence logrotate skipped that file, which kept growing and eventually filled up the partition. In yet another case, a certain set of log files were not covered by the logrotate tool in the earlier releases of software, and those eventually filled up the partition. Continuous analysis of disk usage would help spot issues in case the log rotation doesn't work as expected or the backup mechanism fails to kick in.

2. **Degrading response time**
In some systems, it was observed that displaying the list of patterns deployed was taking too long (100s of seconds rather than 10s of seconds). Upon investigation, it was found that the CPU utilization and memory usage of one of the IWD processes were unusually high. The IWD provides service to store metrics corresponding to each pattern. After a detailed investigation, it was found that a legacy collection of patterns (that were heavily used in these systems) were making Secure Shell (SSH) connections to this process too frequently. Besides, most of the connections were ephemeral. Thus the process spent too much time managing the SSH cache with entries that were hardly reused. This problem was fixed by reducing the frequency of the connections made by the patterns. Although this issue sounds like a performance problem, it started showing up in the customer system only after few weeks of usage, due to a memory leak in the Transmission Control Protocol (TCP) connection queue of the same IWD process, which grew in memory footprint only when the number of concurrent connections was high.

3. **Thread leakage**
In PSM, the jobs are executed by worker processes. A master process manages this pool. Master and workers communicate using Named Pipes[2], where master process creates a pipe to receive inputs from the worker, and the worker process creates a pipe to receive inputs from the master. Both master and worker processes spawn a thread to listen to their respective pipes. Occasionally, the worker process is terminated via a `kill` command, upon which the worker process terminates its input listening thread, removes the input file pipe and sends an end-of-file (EOF) to the output file pipe for master process. When the listening thread on master process sees the EOF, it verifies if the worker's input file pipe is deleted, before it removes the input file pipe and terminates itself. In rare situations, the worker's input file pipe was not deleted before the master verified, and thus the master's input thread got leaked. We observed that the master process was leaking around 30 threads per day, which started slowing down the job queue after a few days of running.

In summary, complex systems such as cloud controllers

[1] www.linuxcommand.org/man_pages/logrotate8.html

[2] http://www.linuxjournal.com/article/2156

have many system-wide and application-specific resources that could potentially exhibit software aging. Although such systems are thoroughly tested, some of the defects are identified only after they are in operation at customer sites, since their activation and error propagation conditions are complex and are uncovered only in the field [5]. Moreover, such failures could have been avoided if we had monitored and analyzed the corresponding system or application metric and performed rejuvenation either at the application or the system (node) level. This motivated our approach to continuously monitor and analyze the usage of resources in the customer systems and prepare a dashboard with a list of corrective actions for the DevOps team members who can work directly with the customers/field engineers and prevent failures from occurring.

IV. Software Aging analysis of customer systems

Software aging phenomenon is detected through *aging indicators*, which refers to the system variables that can be directly measured and can be related to the software aging phenomenon [5]. Aging indicators can be classified as system-wide or application-specific [8], [9]. Identifying both classes of aging indicators helps us identify the potential source of software aging and thus apply selective rejuvenation action [10]. The issues found in the reports (Section III) provide an initial list of aging indicators. Then, we reviewed the aging indicators from literature, and map them to our PSM system and outline our findings in Table I.

TABLE I: Aging indicators for PSM node

Class	Aging Indicators
System-wide	Free swap memory [11], disk usage of key partitions (e.g. /var/log), mean response time [11]
Application-specific	real memory consumed [11], [12], [13], no. of threads [10], no. of open files [10]

A. Aging Indicators

Although total memory consumed by the system is the most cited aging indicator in the literature [9], recent studies [14], [15] have demonstrated that application-specific aging indicators such as heap usage or resident set size (RSS) of the application are more accurate in predicting memory-related aging. In our case study, we monitor the virtual memory size (VMS) and the RSS for each application. In our future work, we plan to capture the heap usage for each application, which could be a more accurate aging indicator than RSS. Most of the system-wide and application-specific aging indicators are collected using the NMON[3] tool running in controller node's Linux OS. All the measurements are taken once every five minutes.

B. Analysis Techniques

We first determine if the data collected shows an increasing trend over time, indicating increasing memory usage or slowing response time and so on. To detect monotonic trends in time-series data, we use the Mann-Kendall test [16], [17] at 5% level of significance with correction for serial correlation using the Yue and Pilon method [18]. To model the growth trend, we pick some of the popular techniques from the literature: Theil-Sen's slope (TS) [19], [11], linear model (LM) [20], quadratic model (Quad.) [20], growth curve model (Growth) [20], piecewise linear model (PLM) with three breakpoints [10]. For datasets that exhibit seasonal behavior as well, we used the time-series techniques such as Holt, Holt-Winters [21], and ARIMA (Autoregressive Integrated Moving Average) [21]. We also attempt variants of Holt-Winters with multiplicative and damped trend.

Using the model that best fit our dataset, we predict the future values of the aging indicator (for two days ahead in our case) and estimate the mean prediction along with the prediction interval (PI). The PI is an estimate of the range in which a single new future observation will fall, with a certain probability [22].

C. Analysis Results & Inference

For our analysis, we fetch aging indicator datasets for a six-day period from a system running the largest workloads among all customers analyzed (more than 750 VMs). This data were collected seven days after the latest PureApplication release [23] was loaded, to make sure that the system had a steady workload and no severe errors were encountered due to the new release. Our goal is to predict the value taken by aging indicators for the next two days (details in next section). From the time-series dataset collected for each aging indicator, we consider the data points for the first four days as training data (∼67%) and the next two days as test data (∼33%). In Figures 2, 3, this separation is shown with a vertical dash-dot line. To evaluate the quality of prediction, we rank the models with three criteria, viz. Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE), and select the model with the best median rank. Aging analysis results for various PSM controller and workload deployer processes are summarized in Table II.

The mean slope corresponds to the slope estimated by the fitted LM. The intercept value of trend line corresponds to the memory usage at the beginning of the analysis window. By estimating the slope as a percentage of the intercept value, we get an idea of the relative magnitude of the growth. We make the following observations:

1) Almost all application processes show an increasing RSS. Since the slope and the intercept vary broadly across applications, it is inappropriate to have a fixed threshold of memory usage across all the applications.
2) Few applications show an increasing VMS. Its growth pattern usually correlates with the corresponding RSS.
3) Notice the staircase pattern of memory growth from Figures 2 and 3. Since application memory usage grows in large infrequent steps, the prediction techniques need to be robust enough to catch it. We found the piecewise linear model to be the best fit across most of the datasets.

For system-wide aging indicators, only disk usage for /var/log shows a growth trend with a slope of 0.21% / day (Figure 5). The current slope is small due to major changes

[3] http://nmon.sourceforge.net/

TABLE II: Aging analysis of memory usage in various Applications

Process	Aging indicator	Trend detected	Mean slope (MB/day)	Intercept (MB)	Best-fit technique
PSM Controller process					
Appln1	RSS	No	–	–	–
	VMS	No	–	–	–
Appln2	RSS	Yes	161.57	1871.22	PLM
	VMS	Yes	160.76	2064.86	PLM
Appln3	RSS	Yes	12.01	1286.26	TS
	VMS	Yes	1.48	1956.45	Quad.
Appln4	RSS	Yes	0.911	841.92	Growth
	VMS	No	–	–	–
Appln5	RSS	Yes	25.13	1421.34	PLM
	VMS	Yes	0.56	2340.84	LM
Appln6	RSS	Yes	8.63	718.42	TS
	VMS	No	–	–	–
Workload Deployer process					
iwd1	RSS	Yes	173.37	4313.42	PLM
	VMS	Yes	18.48	5222.61	PLM
iwd2	RSS	Yes	15.51	3324.65	PLM
	VMS	No	–	–	–
iwd3	RSS	No	–	–	–
	VMS	No	–	–	–
iwd4	RSS	Yes	7.01	2740.19	Quad.
	VMS	Yes	2.29	4315.69	PLM

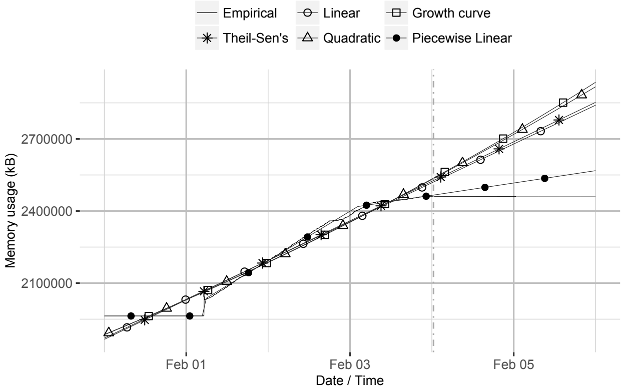


Fig. 2: RSS usage for PSM “Appln2” process

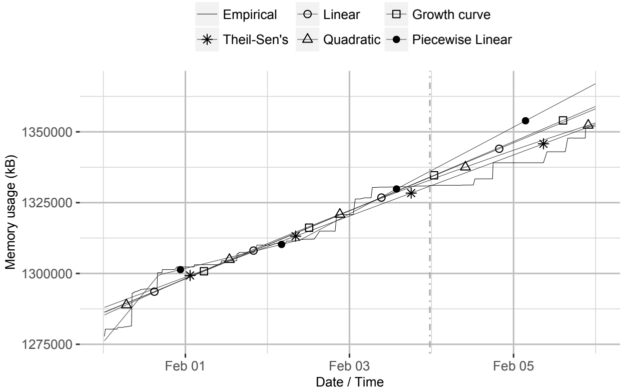


Fig. 3: RSS usage for PSM “Appln3” process

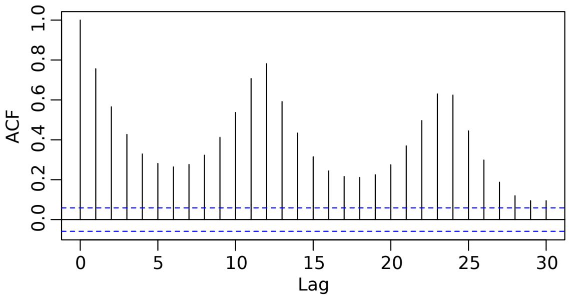


Fig. 4: ACF plot for disk usage of /var/log partition

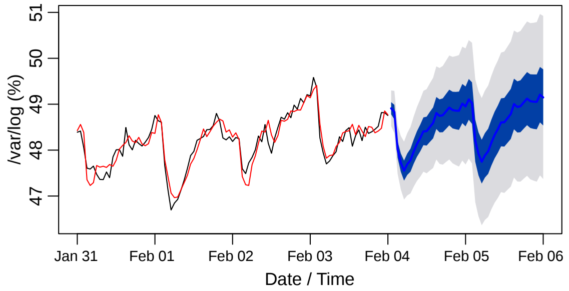


Fig. 5: Predicting disk usage of /var/log partition using Holt-Winters technique (blue and gray area represents 50% and 95% prediction intervals respectively)

in logging for this release; however, an earlier investigation found the usage growing at 2.5% per day, which caused serious problems in the absence of robust logrotation mechanisms. Due to nightly logrotation operation, it has a saw tooth wave pattern with growing trend and exhibits a seasonal pattern with one day period. From the autocorrelation function (ACF) plot in Figure 4, we also observe an hourly seasonal pattern as well (at lag 12, 24 and so on), but we cannot explain it. We remove the hourly seasonal pattern by using hourly approximated data; thus our dataset has a period of 24 (corresponding to 24 samples per day). Since the data show both trend and seasonality, we model these properties using Holt, Holt-Winters and ARIMA methods. We find that Holt-Winters (with additive trend) is the best fit, followed closely by Holt-Winters with multiplicative seasonal trend. We also performed one-step cross-validation across techniques that were close best-fit (Figure 6), which provides us forecast estimates across rolling origins [22]. We see that Holt-Winters is the best fit technique

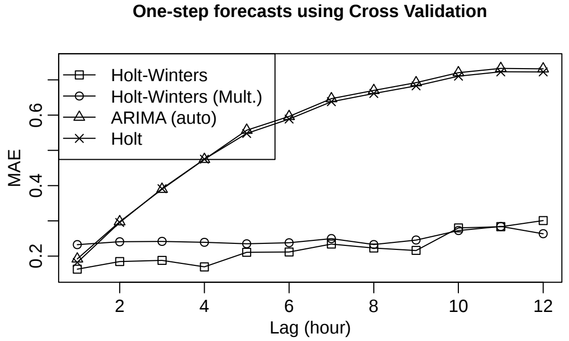


Fig. 6: Cross validation of multiple techniques to predict disk usage of /var/log partition

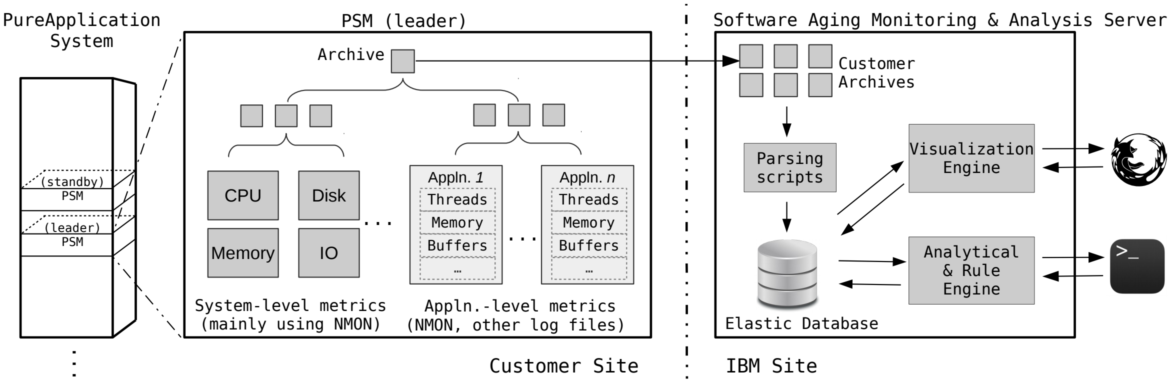


Fig. 7: Architecture of the monitoring and data analytical infrastructure

across all lag values (up to 12 hours). We did not find any growth trend for the remaining aging indicators discussed in Table I, and skip their analyses for brevity.

V. Continuous Analysis and Mitigation of Software Aging

The primary goal of our research is to develop a service that helps to detect and mitigate software aging in cloud systems deployed at the customer site. This service lists software rejuvenation tasks that can be performed by a centrally-located DevOps team. As mentioned earlier, we consider the following two requirements: a) notification horizon of two days, b) severity level associated with the task notification.

A. Monitoring and Data Analytical Infrastructure

The architecture of our monitoring and data analytical infrastructure is shown in Figure 7. Log files capturing various metrics are collected together as an archive and sent to the IBM support service. Due to the complexity involved in transferring this archive, we decided to send it to the IBM site every 24 hours. In our analysis (summarized in Section III), this frequency was reasonable to identify most aging-related problems. The required datasets are parsed from the log files and loaded into an Elastic[4] database. This data can be visualized interactively on a web browser using the Kibana[5] visualization engine. Most of our research and development effort goes into developing the "Analytical & Rule Engine" block. This block pulls the datasets from Elastic database, analyzes it using the algorithm presented in Subsection V-C, and generates actionable items that are pushed into the Elastic database, which can be viewed on the Kibana dashboard.

B. Challenges involved

Our first challenge is to accurately estimate the "time to resource exhaustion" for each aging indicator corresponding to each system deployed at the customer site. Since each system is running a different workload and has been running since a different start time, we perform customized analysis for each aging indicator in each system. As new dataset arrives, we extract data for the past two days to evaluate the models that fitted well on the data predicted for the same period earlier. We use this best-fit model to predict the aging indicator for the next prediction window of two days. Thus we use the most recently validated model for our predictions.

The next challenge is to account for the uncertainty in the prediction estimates by using the prediction interval estimates. Assuming that the new dataset arrives at midnight, let us analyze the RSS usage for "Appln3" at midnight for three consecutive days using the linear model (Figure 8). For the sake of this example, let us assume an RSS usage threshold of 1.35GB before rejuvenating the application. Figure 8 shows the prediction mean estimates for the next two days along with 50% prediction interval and 95% prediction interval for three consecutive days. Let us define our scheme for assigning "Severity" for our alerts. In the next two days, if the resource is expected to cross the threshold only within the 95% interval, but not within the 50% interval, then there is a small possibility of resource exhaustion. Let us mark such alerts as "Yellow" (low severity). If the resource is expected to cross within 95% and 50% interval, then there is a higher possibility, and such alerts can be marked as "Orange" (medium severity) (Figure 8b). If the mean prediction is also expected to cross the threshold within two days, we can mark such alerts as "Red" (high severity) (Figure 8c). Thus we use the prediction intervals to assign a severity rating to our alerts.

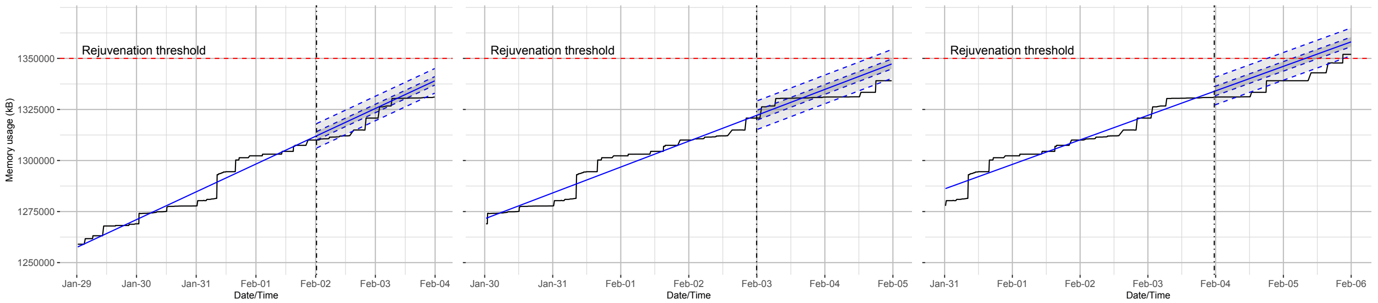
C. Continuous Analysis scheme for software aging

Inputs to the algorithm:

- *N* is the notification horizon, the minimum number of days required for the DevOps team to take action (2 days in our case).
- *Customers* is the list of customer system IDs.
- *AgingIndicators_all* is the list of aging indicators defined for each customer. Each aging indicator defines the slope estimation technique with variable *technique* and its corresponding parameters with variable *params*, threshold for alert with variable *limit*, and corresponding action with variable *action*.
- *CurrentTime* indicates current time while generating an event.
- *technique_all* is a list of all slope estimation techniques.

[4] https://www.elastic.co/

[5] https://www.elastic.co/products/kibana



(a) Day 1 - No alert

(b) Day 2 - “Yellow” alert

(c) Day 3 - “Red” alert

Fig. 8: PSM “Appln3” RSS usage process over three consecutive days

We are assuming all aging indicators show an increasing trend; however, our scheme can easily accommodate other cases.

Algorithm 1 Continuous analysis scheme to detect software aging in customer systems (run nightly)

```
for each customer in list customers_all do
    for each AgI in list AgingIndicators_all do
        AgI_Past ← measured AgI values for past N days
        for each technique in list technique_all do
            Fetch AgI_PastPred$technique as an array of predicted AgI values
                from N days before
            Evaluate goodness-of-fit between AgI_Past and AgI_PastPred$technique
            if best-fit technique then
                AgI$technique ← technique
                AgI$params ← parameters for technique
            end if
        end for
        Fetch AgI values for past 4 days and estimate trend using Mann-Kendall test
        if trend is +ve then
            Estimate AgIPred for time [CurrentTime : CurrentTime + N days]
                using (AgI$technique, AgI$params)
            AgIPred_Mean ← mean AgIPred at time CurrentTime+N
            AgIPred_Upper95 ← 95% upper PI of AgIPred at time CurrentTime+N
            AgIPred_Upper50 ← 50% upper PI of AgIPred at time CurrentTime+N
            if (AgIPred_Upper95 > AgI$limit) & (AgIPred_Upper50 > AgI$limit)
              & (AgIPred_Mean ≥ AgI$limit)
                create Event (CurrentTime, customer, AgI, “Red”, AgI$action)
            else if (AgIPred_Upper95 > AgI$limit) & (AgIPred_Upper50 ≥ AgI$limit)
                create Event (CurrentTime, customer, AgI, “Orange”, AgI$action)
            else if (AgIPred_Upper95 ≥ AgI$limit)
                create Event (CurrentTime, customer, AgI, “Yellow”, AgI$action)
            end if
        end if
        Estimate AgIPred for time [CurrentTime: CurrentTime + N days]
            using all techniques and store
    end for
end for
```

D. Evaluation of our scheme

We analyze the above-described algorithm with over two months of offline data for the RSS usage of “Appln3” and “Appln6” respectively, by randomly choosing thresholds falling within the range of data and generating the list of alerts. Across all severity levels, we find the alerts are generated with a precision of around 0.8 and recall of 1.0. Thus for every threshold violation, we find at least one event generated with any severity level. We can conclude that the slope estimation techniques are adapting well to different data patterns. Another reason is the quality of the aging indicator since application-specific aging indicators are significantly superior to system-wide aging indicators. Note that most of the false negative alerts were those generated too early (a few days sooner than our notification horizon), which is not necessarily harmful. We leave further tuning and analysis of the scheme as future work.

VI. Related Work

Our work is inspired by the solution developed earlier for the IBM Director system management tool at IBM [10] which were focused on cluster environments. Our work differs in two aspects: i) Aging detection and analysis are performed in a centralized analytics server as opposed to performing it in each cluster node; ii) Rejuvenation tasks are managed by a centralized DevOps team located at IBM, as opposed to the customer itself. In addition to providing this as a service to customers, it provides a wide-view of issues running across all customer systems. We plan to use this approach to find patterns of software aging issues among customers running similar workload or running same PSM software release and resolve issues proactively. We also plan to extend the analytical subsystem to detect software aging at hypervisor and operational profile level. Authors in [10] also analyzed the availability of cluster system using time-based vs. inspection-based techniques and found inspection-based rejuvenation results in much higher availability. We preferred to take this approach as well.

Software aging-related issues have been observed in various open-source cloud-oriented software systems [7], virtualization systems [24], and middleware systems [13]. In [20], the authors reported software aging in the memory usage of the node controller and other critical components in Eucalyptus[6] cloud management system. They used time-series prediction techniques to estimate time to reach critical resource usage threshold. To increase system availability, they proposed rejuvenating the system within a safe time limit before the critical resource usage is attained. However, their work did not account for the uncertainty in the prediction estimates. Also, they assumed the system could be rejuvenated (automatically / manually) as soon as the safe time limit is reached, whereas we consider a remote monitoring and DevOps scenario.

[6] http://www8.hp.com/us/en/cloud/helion-eucalyptus.html

VII. Conclusion & Future work

In this paper, we share our experience monitoring and mitigating real software aging issues in the cloud controller node of IBM PureApplication systems deployed at customer sites. We discuss three representative software aging issues that frequently occurred, viz. increasing usage of disk partitions, degrading response time, and thread leaks. Failures caused by such defects could have been avoided if we received early warnings for potential failure scenarios and corrective actions were taken preventively. We built a system at IBM site that continuously monitors and analyzes key aging indicators for all the customer systems, and generates alerts with a lead time of two days. By taking into account the uncertainty involved in such predictions, we propose a prioritization scheme that assigns severity to such tasks, which helps the DevOps team prioritize the aging mitigation tasks along with their other responsibilities. In this study, we also share our insights and guidance on trend estimation techniques that work the best for each aging indicator. By having a custom rejuvenation threshold for each aging indicator per customer, we conclude that continuously monitoring and analyzing key aging indicators can help prevent failure scenarios in systems deployed at customer sites.

Our current study is restricted to the IBM PureApplication platform. Future works could apply the same approach to other cloud platforms (e.g., Eucalyptus). We plan to evaluate the algorithm under different DevOps scenarios, sampling periodicity (e.g., 6 hours, 12 hours), and shorter prediction durations (e.g., 4 hours, 8 hours). Thus our scheme could be expanded to work in both online and continuously. Also, we plan to expand this model in the future taking other relevant system/application parameters into account, while still keeping the scheme general enough. Also, we plan to enhance alerts to include information related to prior fixes applied by the DevOps team using IBM Watson APIs[7].

Acknowledgement

This research was supported in part by IBM under a faculty award and an IBM student fellowship. The authors would like to thank Hugh Hockett and Christopher Laffoon for mentoring Harish through the project, to Nicholas Schambureck for support with the software infrastructure, and to Derrick Foley and Keith Rafferty for motivation and support for this research.

References

[1] McKinsey and Company, "From box to cloud: An approach for software development executives," http://www.mckinsey.com/business-functions/business-technology/our-insights/from-box-to-cloud, January 2015, accessed: 2017-08-15.

[2] IBM, "Advantages and options of private cloud computing," https://www.ibm.com/developerworks/rational/library/private-cloud-advantages-options/, accessed: 2017-08-15.

[3] ——, "A tour of the hardware in IBM PureApplication System," http://www.ibm.com/developerworks/websphere/techjournal/1407_woolf2/1407_woolf2.html, accessed: 2017-08-15.

[4] Aerohive Networks, "Public or private cloud: The choice is yours," http://media.aerohive.com/documents/901259441_Aerohive-Whitepaper-Public-or-Private-Cloud.pdf, accessed: 2017-08-15.

[5] M. Grottke, R. Matias, and K. Trivedi, "The Fundamentals of Software Aging," in *IEEE WoSAR*, Nov 2008, pp. 1–6.

[6] IBM, "How, where, and why IBM PureApplication fits in your cloud," http://www.ibm.com/developerworks/websphere/techjournal/1506_dejesus/1506_dejesus-trs.html, accessed: 2017-07-15.

[7] F. Machida, J. Xiang, K. Tadano, and Y. Maeno, "Aging-Related Bugs in Cloud Computing Software," in *IEEE WoSAR*, Nov 2012, pp. 287–292.

[8] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A Survey of Software Aging and Rejuvenation Studies," *ACM JETC*, vol. 10, no. 1, pp. 8:1–8:34, Jan. 2014.

[9] N. A. Valentim, A. Macedo, and R. Matias, "A Systematic Mapping Review of the First 20 Years of Software Aging and Rejuvenation Research," in *IEEE WoSAR*, Oct 2016, pp. 57–63.

[10] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive Management of Software Aging," *IBM Journal of Research and Development*, vol. 45, no. 2, pp. 311–332, Mar. 2001.

[11] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi, "Analysis of Software Aging in a Web Server," *IEEE Trans. on Rel.*, vol. 55, no. 3, pp. 411–420, Sept 2006.

[12] R. Matias, I. Beicker, B. Leitao, and P. R. M. Maciel, "Measuring software aging effects through OS kernel instrumentation," in *IEEE WoSAR*, Nov 2010, pp. 1–6.

[13] G. Carrozza, D. Cotroneo, R. Natella, A. Pecchia, and S. Russo, "Memory Leak Analysis of Mission-Critical Middleware," *Journal of Systems and Software*, vol. 83, no. 9, pp. 1556 – 1567, 2010.

[14] F. Machida, A. Andrzejak, R. Matias, and E. Vicente, "On the effectiveness of Mann-Kendall test for detection of software aging," in *IEEE WoSAR*, Nov 2013, pp. 269–274.

[15] R. Matias, A. Andrzejak, F. Machida, D. Elias, and K. Trivedi, "A Systematic Differential Analysis for Fast and Robust Detection of Software Aging," in *IEEE SRDS*, Oct 2014, pp. 311–320.

[16] H. B. Mann, "Nonparametric Tests Against Trend," *Econometrica*, vol. 13, no. 3, pp. 245–259, 1945.

[17] M. G. Kendall, *Rank Correlation Methods*, 4th ed. Griffin, London, 1970.

[18] S. Yue, P. Pilon, B. Phinney, and G. Cavadias, "The influence of autocorrelation on the ability to detect trend in hydrological series," *Hydrological Processes*, vol. 16, no. 9, pp. 1807–1829, 2002.

[19] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, "A Methodology for Detection and Estimation of Software Aging," in *IEEE ISSRE*, Nov 1998, pp. 283–292.

[20] J. Araujo, R. de S. Matos, V. Alves, P. R. M. Maciel, F. V. de Souza, R. M. Jr., and K. S. Trivedi, "Software Aging in the Eucalyptus Cloud Computing Infrastructure: Characterization and Rejuvenation," *ACM JETC*, vol. 10, no. 1, pp. 11:1–11:22, 2014.

[21] J. P. Magalhães and L. M. Silva, "Prediction of performance anomalies in web-applications based-on software aging scenarios," in *IEEE WoSAR*, Nov 2010, pp. 1–7.

[22] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2013.

[23] IBM, "IBM Pureapplication Software Suite v2.2.2," http://www-01.ibm.com/common/ssi/rep_ca/5/877/ENUSZP16-0445/ENUSZP16-0445.PDF, accessed: 2017-07-15.

[24] F. Machida, D. S. Kim, and K. S. Trivedi, "Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration," *Performance Evaluation*, vol. 70, no. 3, pp. 212 – 230, 2013.

[7] https://www.ibm.com/watson/