Towards Automated Regression Analysis and Management Throughout Software Life-Cycle

Ankur Gupta
and
Veena Tripathi
Model Institute of Engineering and Technology, Jammu, India

Software application performance tends to degrade over a period of time due to introduction of new features, feature enhancements and bugs-fixes as part of the software maintenance lifecycle. Critical applications cannot afford the performance degradation. However, teams engaged in software maintenance are not too effective in detecting potential performance impacting issues during the test and release phases of the software engineering process, since the run-time environments for individual customers are difficult to simulate. This challenge is exacerbated for applications with a large installed-base. This research paper proposes the concept of Application Baselining as a means to effectively detecting and containing software regression. It provides indication of real-time application performance by monitoring its critical parameters over long periods of time. By keeping track of the changes made to the application and its environment, their impact on application performance is correlated. The changes which adversely impact the application performance are then rolled-back to mitigate their effect. Early work towards development of such a framework is presented.

Keywords: Software maintenance, automated regression analysis, application baselining.

1. INTRODUCTION

Software maintenance for long-running products with a large installed base is a challenging proposition to say the least. The primary author has had first-hand experience of being involved in the maintenance of HP OpenView Network Node Manager[1] which has an estimated installed-base of over 15,000 instances across several product versions spanning close to three decades and spread over all geographies. Some of the unique challenges encountered in the maintenance for such products are:

a) **Extreme Heterogeneity** in terms of hardware configurations, operating systems and versions, OS patch levels, product version and patch levels, diversity in system and product run-time environments, customer use-cases and processes. This manifests in diverse performance profiles of the same code-base at different customer installations.

b) **Extreme Heterogeneity** in terms of hardware configurations, operating systems and versions, OS patch levels, product version and patch levels, diversity in system and product run-time environments, customer use-cases and processes. This manifests in diverse performance profiles of the same code-base at different customer installations.

c) **Root-Cause Analysis** becomes complicated in such environments as correlating code-changes to observed performance variations in specific instances of the product is non-trivial and requires significant manual analysis and intervention.

d) **Ensuring Effectiveness of Maintenance Process** becomes challenging as every patch release can potentially impact performance at specific product instances while having little or no performance impact at others. Negative impact of a released patch can involve patch recalls or releasing emergency patch-fixes or issuing workarounds which are expensive.

[1]http://www8.hp.com/in/en/software-solutions/network-node-manager-i-network-management-software

e) **Cascade Effect** in such maintenance environments is pronounced. Fixing a performance issue in one product version can potentially result in product fixes to be undertaken across the product family to ensure consistency. Even performance fixes in specific modules can cause significant changes to occur in other dependent modules complicating the maintenance process.

f) **Customer Satisfaction** can be hit if the maintenance process fails to adequately address performance issues. Customers expect better performance and functionality when new product patches are installed.

Run-time performance issues will still not be detected until the customer reports them in. After the defect is reported correlating the performance degradation with the potentially large number of defect fixes is again time consuming for the maintenance teams.
What is needed is a mechanism to automatically quantify the performance impact of released software updates (patches) in specific customer environments, correlate the performance impact with specific fixes to the code-base and notify the maintenance team if the performance impact is severe enough to impact customer experience. This research papers presents early work towards the development of a framework which tracks real-time software regression and helps alleviate the pain points in software maintenance as discussed above.
The proposed framework relies on maintaining detailed performance profiles or baselines for the different application processes which comprise the installed product. Any significant variations against the observed performance profiles as a result of installation of new software updates are reported back to the maintenance team with a detailed report on the modules in which variations are observed, degree of variation and the list of defect fixes in the related or dependent modules. This aids in quick detection, root-cause analysis and potentially quick turnaround time to fix and release regression causing software updates.

2. BACKGROUND AND RELATED WORK

A software application comprises of multiple processes, directories/files and each process has a particular behavior pattern with respect to the following system parameters:

—CPU Utilization trends.

—Memory Utilization trends.

—Hard Disk Utilization trends.

—Open file descriptors/files manipulated.

—Network Connections Established.

—Traffic generated.

Over a period of time we observe a baseline [2] (normal) behavior for a process and hence cumulatively for the entire application in a particular deployment (run-time) environment. Different run-time environments will have a corresponding baseline behavior for the application in question which depends on the use-case scenario of the application, hardware resources on the system and other environmental factors (other active applications on the system), etc. Thus, the same application can have a completely different performance profile on different systems and in different deployment scenarios. For applications with a large deployment base (large number of customers), it becomes difficult for the software maintenance teams to deal with following issues:

[2]http://www.cisco.com/c/en/us/support/docs/availability/high-availability/15112-HAS-baseline.htmltext

✓ Stimulate the exact customer environment to understand performance/ regression issues in the context of the customer.

✓ Manage regression issues in the context of the customer.

✓ Understand the impact of bug-fixes, feature enhancements on application performance in varied deployment environments.

✓ Provide customer context-specific maintenance and recommendations to prevent degradation in application performance.

✓ Maintain quality and performance over long periods of time.

To solve customer-reported issues maintenance teams need to retrieve a great deal of customer information which requires several rounds of communication with the customer. This is followed by the simulation of the customer environment and reproduction of the defect as observed. Thus, eliciting the complete information about the customer environment and reproduction of the defect are time-consuming activities which impact the turnaround time for the resolution of the customer issue.

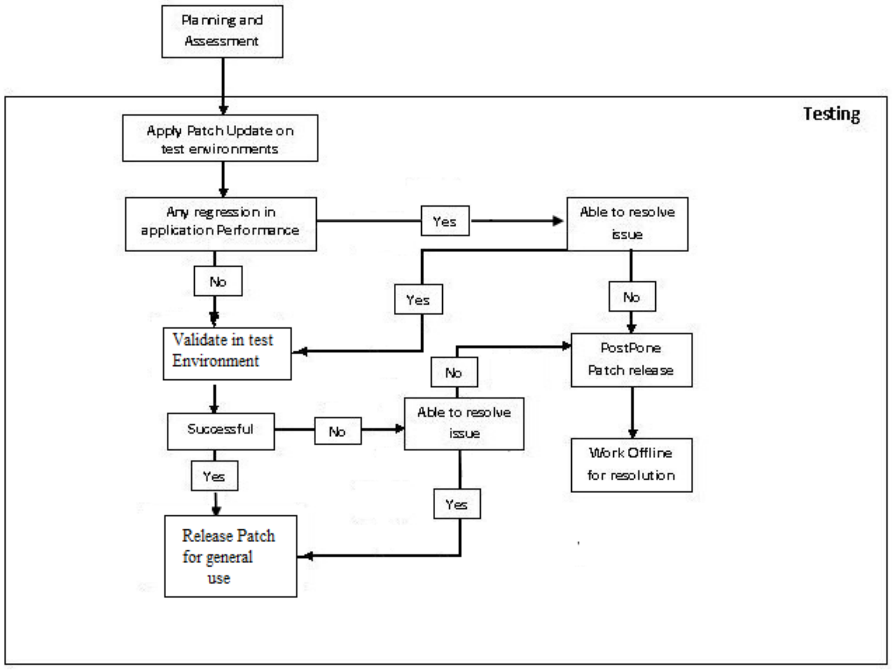
Planning and Assessment
Testing
Apply Patch Update on test environments
Any regression in application Performance
Yes
Able to resolve issue
No
Yes
No
Validate in test Environment
PostPone Patch release
No
Successful
No
Able to resolve issue
Work Offline for resolution
Yes
Yes
Release Patch for general use

Figure 1. Application update/maintenance process in test environment

Figure1 Shows flow-chart for the application update/maintenance process in the test phases illustrating several steps and validations involved. In the Planning and Assessment phase of testing the new patch any regression in application performance is compared and recorded. If there is no regression noticed, the patch is flagged as successful and processed for release to the customers. However, if regression is noticed the development team may work to fix the issue before release of the patch. This process potentially involves several iterations. Typically, after the patch is available internally the maintenance team tests the patch on a production environment which is designed to mimic actual customer environment. The patch is typically revalidated and then released externally. The challenge with this approach is that the diversity of customer environments cannot be reproduced successfully and hence all potential problems cannot be screened beforehand.

2.1 Application Baselining

A Baseline [3] is a benchmark that is used as a foundation for measuring or comparing current and past values. Some real-word examples of baselining include network performance management, scope baseline for quality planning, project management and medical science etc. Network Baselining McKellar [1996] is a commonly used technique for analyzing computer network performance. It involves measuring and rating the performance of a network in real-time. Once a network baseline has been established, it is then used to determine both present and future network upgrade needs as well as assist in making changes to ensure that the current network is optimized for peak performance.

Baselining a software application is required to identify key performance indicators in terms of the quantum of system resource usage and their temporal relationships. Application Baselining for determining run-time software regression is novel and a unique contribution of this paper. Through application baselining the installed application image (static) and resource usage and behavior (dynamic) of the application are monitored. The static profile includes the number of files installed, their checksum and size in bytes etc. Changes in the static profile of the installed application are useful in correlating with performance issues in specific software updates. The dynamic profile of the application includes dynamic resource usage such as CPU, Memory and I/O as a function of time. Thus, by matching run-time performance with past performance profile for the same time period trends either regressive or progressive can be determined and corrective interventions if any can be initiated.

The framework thus employs application performance baselining to determine performance regression as an outcome of software maintenance processes throughout the product lifecycle.

2.2 Related Work

In Zheng et al. [2014] propose an automatic framework for detection and characterization of performance degradation in software systems and have validated the framework for web-servers specifically. The proposed theoretical model focuses on determining software aging or degradation in terms of response time, QoS and resource utilization. However, this model does not account for performance degradation as a result of intermediate software maintenance updates, relying on a black-box approach. In contrast, the proposed scheme uses a white-box approach to correlate software updates to observed performance degradation in diverse customer environments at considerable scale.
Cito et al. [2014] identify root of performance degradation in web-based applications using Changepoint Analysis. They have created a simulation model based on the taxonomy of root causes in server performance degradation by continually collecting data in order to observe and track changes in desired metrics, e.g., service response time. This enables detecting anomalies, identifying patterns, ensuring service reliability, measuring performance changes after new software releases, or discovering performance degradation. However, the proposed framework uses a single point of reference for web-servers and is not applicable to software applications which operate in diverse real-time customer environments as [4].
In Jin et al. [2012] classify performance bugs in five real-world software application suites and develop a framework to classify such defects. This aids development and maintenance teams in detecting such defects early, fixing them or even avoiding them in future. However, undertaking such a comprehensive study is not feasible for all applications. Secondly, given the complexity involved in maintenance of large applications by geographically distributed teams, avoiding performance bugs altogether remains unrealistic. Having maintenance teams receive real-time feedback at run-time based on previously benchmarked performance baselines is practical and allows for quick correlation between specific software updates and degradation in specific application modules/processes.

[3]http://www.cisco.com/c/en/us/support/docs/availability/high-availability/15112-HAS-baseline.html
[4]http://www8.hp.com/in/en/software-solutions/network-node-manager-i-network-management-software

The Walker et al. [1998] describes a method and computer product for facilitating regression testing during the development and other life cycle phases of a software application comprised of transactions. The test data is in a robust functional description of the transaction such that physical modifications to the transaction during software development preserve the viability of the test data for execution in the modified transaction. However, this strategy is manual in nature and implemented before the product is released. Thus, regression analysis is performed in the pre-release test environment and not in the actual run-time environment during the life-cycle of the deployed software.
In Gupta [2012] authors describe a collaborative software maintenance framework which automatically builds expert knowledge aiding in maintenance of complex software and claim that such a system would contain regression by leveraging the knowledge of experts in the team. However, the effectiveness of the framework is not tested in real-world scenarios.
The Lane [2000] describes a method for regression and verification of hardware, network, and/or software technology platforms to deliver acceptance status details. This method variously prepares instructions and components to support unique business customer environments and manages the regression verification of these environments. Errors, deviations, and recommendations for improvement, with full regression capabilities, are reported to the business user customer. The drawback of this strategy is also the lack of real-time monitoring of the software application and the hardware and operating environment.
The proposed framework is based on the Indian patent application Gupta and Shamim [] filed by the authors. It alleviates the shortcomings mentioned above by ensuring that the application vendor can better track performance of the deployed application in the run-time environment for an individual customer (s) spanning over different time periods of software development and up-gradation. Thus, the proposed invention ensures continuous and comprehensive performance monitoring for real-time regression analysis.

3. SYSTEM MODEL

This research paper proposes a framework for creating a baseline which provides an indication of the performance of any software application over its entire lifecycle. This is done through real-time monitoring of the behavioral pattern of its critical parameters and storing them for future reference. Comparison with existing performance baselines can be used to detect and manage regression in the software application introduced due to changes in the application environment or as a result of the software maintenance process. By keeping an audit trail of the changes made to the application and its environment, their impact on application performance can be determined. It also provides an indication of the quality and effectiveness of the software maintenance process followed by the application development vendor. Through the proposed invention, the application vendor can better track performance of the deployed application in the run-time environment for an individual customer. Insights gained from the proposed framework help the application vendor in making informed decisions to improve quality of software maintenance process and manage changes which impact application performance adversely.
Figure2 presents the overall schematic of the proposed application baselining framework which consists of additional monitoring modules installed at the customer location, which provide real-time monitoring capabilities for the installed application. Any deviations from the known normal performance parameters pertaining to the application are intimated through events to the central server maintained by the application vendor. Based on the quantum of variation the events are categorized as critical, major, minor and appropriate automated actions can be initiated in case the application performance is impacted due to recent software maintenance updates.

3.1 Server Side Components

3.1.1 *Listener:.* The listener is responsible for receiving all events from the customer-side baseline agent. It forwards all received events to the action manager for initiating an appropriate response if required.

3.1.2 *Action Manager:.* The action manager is responsible for initiating corrective actions in case performance degradation is detected. It involves roll back of the patch incase application performance degradation is detected and classified at a major or critical level subsequent to patch updates. The classification of events is based upon the degree of variation observed from the stored normal baselines for the application module(s) in question and based on a set of pre-defined rules. It also raises a trouble ticket for the event and notifying the maintenance team and the system administrator (customer-side) with relevant details. The patch rollback process can be fully automated or require the intervention of the system administrator.

3.1.3 *Baseline Database:.* The baseline database holds a repository of previous normal baseline values for the application in the customers run-time environment. These baselines are measured against key application performance indicators like CPU usage, memory utilization, network performance etc. which are unique to the customer environment and can be expected to vary over time. Performance parameters are in turn dependent upon the hardware configuration deployed by the customer, the operating system, the operating system patch level, the load on the system, the size of the application database, the duration for which the application is active, the application patch level and configuration parameters for both the system and the application. The baseline values help in correlating observed application performance with changes in the system environment and application updates.

3.1.4 *Patch DB:.* The patch database is the repository of all production patches released by the maintenance team for different application versions. Whenever it receives a new patch for production deployment release from the maintenance team, it notifies the action manager about the new release.

3.2 Customer Side Components

3.2.1 *Listener (client):.* Should have root privileges and is responsible for installing and uninstalling patch updates received from the server side Action Manager after approval from the designated human administrator.

3.2.2 *Local baseline Store:.* This is a repository of previous normal baseline values for the client runtime environment for pre and post patch update application performance analysis. The baseline values need to be updated since normal behavior for an application can change over its lifecycle depending upon application upgrades, new components added, feature enhancements and modifications. Baselines for varying time durations are part of the store. Performance measurements over one month for which no action is initiated by the human managers or by the server side action manager are taken to be new baseline values, since these would indicate expected application behavior and performance.

3.2.3 *Baseline Agent.* This agent is responsible for measuring current Application performance and comparing with previous baseline values from the local baseline store and communicating the values to the listener at the server end. The performance counters used by the baseline agent for an application are:

—Current CPU Utilization of the Application (% Processor Time counter).

—Current CPU Utilization of the Application (% Processor Time counter).

—Current Memory Utilization of the Application (Available Bytes and Pages/sec counters).

—Current Network Utilization of the Application (Bytes Received, Bytes Sent, Connections Established, Datagrams Received, Datagrams Sent counters).

—Current Disk Utilization of the Application (Disk Reads/sec, Current Disk Queue Length,

In addition, the baseline agent also keeps track of the static installed image of the application in terms of number of files installed, their associated checksum, new files introduced, configuration

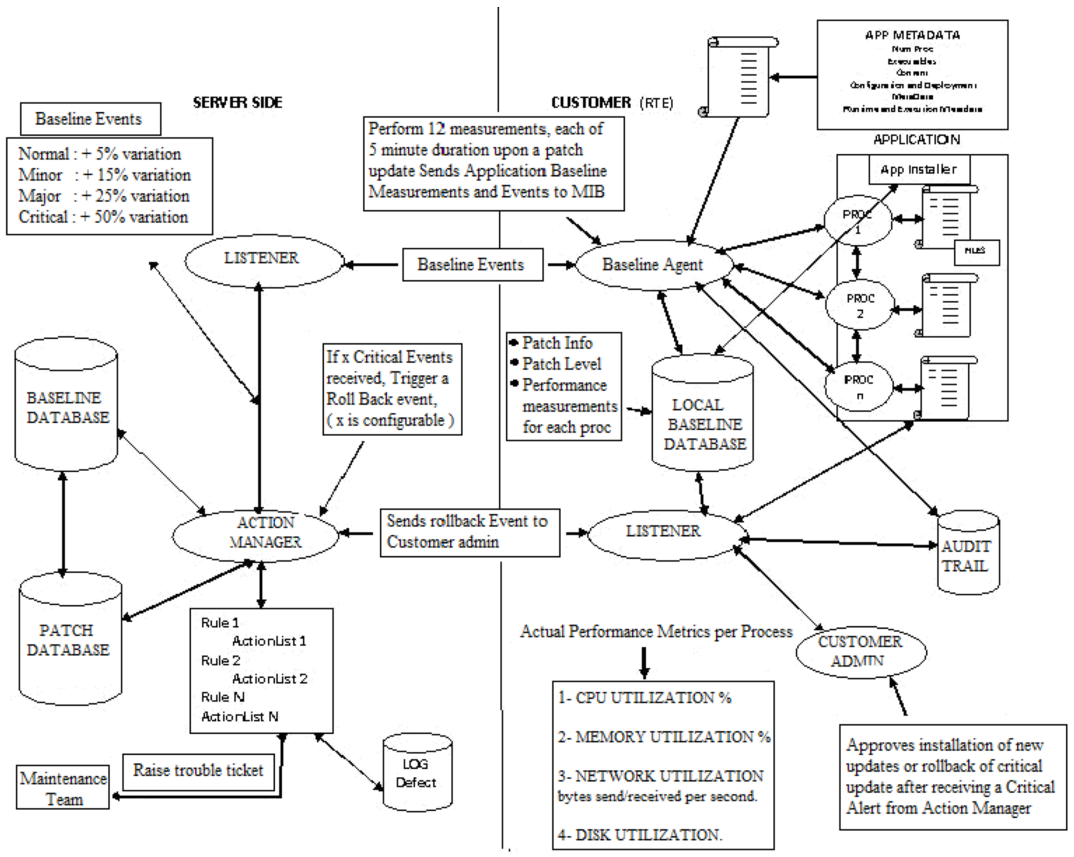
SERVER SIDE
CUSTOMER (RTE)
Baseline Events
Normal : + 5% variation
Minor : + 15% variation
Major : + 25% variation
Critical : + 50% variation
LISTENER
Baseline Events
Baseline Agent
Perform 12 measurements, each of 5 minute duration upon a patch update Sends Application Baseline Measurements and Events to MIB
APP METADATA
APPLICATION
App Installer
PROC 1
PROC 2
PROC n
FILES
BASELINE DATABASE
If x Critical Events received, Trigger a Roll Back event, (x is configurable)
Patch Info
Patch Level
Performance measurements for each proc
LOCAL BASELINE DATABASE
ACTION MANAGER
Sends rollback Event to Customer admin
LISTENER
AUDIT TRAIL
PATCH DATABASE
Rule 1
ActionList 1
Rule 2
ActionList 2
Rule N
ActionList N
Actual Performance Metrics per Process
CUSTOMER ADMIN
1- CPU UTILIZATION %
2- MEMORY UTILIZATION %
3- NETWORK UTILIZATION bytes send/received per second.
4- DISK UTILIZATION.
Approves installation of new updates or rollback of critical update after receiving a Critical Alert from Action Manager
Maintenance Team
Raise trouble ticket
LOG Defect

Figure 2. Schematic of the Proposed Framework

parameters changed so that the changes to the application and its environment can be correlated to performance.

3.3 Sequence of Operations

The Sequence of Operations of the proposed framework is described below:

(1) Software and monitoring framework installation at customer location. This is the setup phase in which the customer installs the software from the vendor. As part of the installation package the monitoring framework including the Baseline Agent (BA) and associated files also gets installed.

(2) Establishing connection between Baseline Agent (customer) and Listener (Server side). The baseline agent is hardcoded with the address of the Listener so that it can establish a preliminary connection after installation.

(3) Action manager checks for any available patches to ensure customer software is concurrent with latest patch levels. During the first connection, the listener on the server side checks whether the latest patches are installed at the customer-site based on the version information provided by the BA. If the latest patches are not found, the action manager pushes the patches to the customer site, where they are installed automatically and the human administrator at the customer location notified.

(4) Baseline agent begins run-time monitoring of application which include
—monitoring all processes and sub-processes
—monitoring performance parameters cpu, disk, memory, network etc.

—monitoring all the installed files, directories and sub-directories installed as part of the application package
—monitoring system parameters like memory, cpu, disk and other vital parameters content...

(5) Creation of first performance baselines (duration: startup, 24 hours, 1 week,1 month, 1year) for the running software for each process and sub-process.
(6) Run-time variance measurement of actual performance vs. baseline performance.
(7) If variance is detected, BA generates baseline threshold events.
(8) Listener receives and forwards to action manager after classifying events as normal, minor, major and critical based on
(9) Action manager logs event in baseline DB, checks patch DB for the customer, executes rules, raises trouble-ticket and may affect a rollback at the customer-site to previous known stable patch level.
(10) Rollback event received and executed by the listener at customer-site, notification to customer-site admin and application rolled back and restarted.
(11) Event logged in customer-site audit trail which keeps record of all changes to environment and application.

Table-1 list out the important fields in the baseline event as generate by client baseline agent.

Field	**Description**
x-name	Name of Application
date	Date of Event
time	Time the Event occurred
time-period	Time duration for which baseline is violated (statup, 5 min,1 hour,24 hours etc.)
hw-desc	Description of the Hardware
sys-param-var	List including OS, patch-level and environment variables
x-type	Type of Event
x-category	Event Category
x-desc	Event Description
x-ctx	Event-dependent context information
x-plevel	Current Application Patch Level
x-pid	Application sub-process ID for which baseline violation observed
x-cpu-load	Current measured Application CPU Utilization %
x-mem-load	Current measured Application Memory Utilization %
x-disk-load	Current measured Application Disk Utilization %
x-r-bytes	Current measured Application Network Utilization (bytes sent/received per second).
x-var	Variable list with ¡name, value¿ pairs for the baseline parameters e.g.<cpu, 90>, <mem, 86>, <disk, 67> etc. the number of parameters can be variable depending upon the type of event.

Table I: Event details sent by the baseline agent to the server-side listener

The baseline agent is capable of measuring several performance counters based on utilization of CPU, Memory, Network and Disk besides system and environmental parameters. Two listener modules each at the vendor and customer location for receiving messages from either side and interacting with the respective Action Manager (server side) and Baseline Agent (customer side). All events at the customer side are recorded at the audit trail database repository and notified to the human administrator. Similarly all baseline violation events received at the server side are recorded in the defect database repository and notified to the maintenance team for further analysis and rectification.

The proposed framework thus addresses the following issues:

—Enable automated collection of real-time information from the customer environment (system configuration, OS version and patch levels, application version and patch levels, system parameters etc.) reducing the time to collect run-time information for the software maintenance team.

—The run-time monitoring of the deployed application and creation of performance baselines is based on a host of system, environment and application-specific parameters, creating a holistic correlation between software performance and its environment.

—Continuous updating of performance baselines based on the performance evolution of the application to reflect new normal states of performance.

—Automated context-specific analysis of application performance by comparing current performance levels with the known normal performance baseline for the application.

—Tracking of system and application updates and establishing their correlation with application performance.

—Initiate automated corrective action at the customer-site to mitigate any adverse impact on application performance due to software upgrades.

—Improve effectiveness of the software maintenance process.

—Helps to improve Quality-of-Service provided to individual customers even with a large customer-base.

4. CONCLUSION AND FUTURE WORK

This research paper presents a novel framework based on application performance baselining to effectively detect and manage regression in s performance throughout its lifecycle and aid the software maintenance team in effectively dealing with run-time regression. The framework also enables in quick detection of performance issues in the customer run-time environments providing a correlation between application performance degradation and applied software updates. This helps the software maintenance team an aid in effective root-cause analysis leading to potentially quicker resolution of customer issues and improved quality-of-service in terms of software maintenance. The framework therefore addresses a valid real-world issue which typically affects all software maintenance teams involved in maintaining legacy products with a large installed base spanning several generation and versions.

Future work needs to focus on establishing specific scenarios for updating the performance baselines. Defining application performance parameters for diverse run-time environments remains challenging and may vary over periods of time as the software evolves. Hence, a fool-proof mechanism to adopt new baselines may require a combination of historical analysis coupled with human intervention to define acceptable performance bounds for the application for subsequent software updates. Testing in a real-world scenario is a must for establishing the effectiveness of the proposed framework and will require customer buy-in to implement run-time monitoring agents to monitor application performance. Also, some further validation on the frequency of application performance monitoring shall also be required. Frequent monitoring shall consume additional compute resources and keeping these to a required minimum shall also be a key requirement for the successful implementation of the proposed framework.

References

Cito, J., Suljoti, D., Leitner, P., and Dustdar, S. 2014. Identifying root causes of web performance degradation using changepoint analysis. *Series Lecture Notes in Computer Science Vol.8541*, pp.181–199.

Gupta, A. 2012. Practitioner-oriented collaborative and cooperative software maintenance. *International Journal of Computer Science: Theory, Technology and Applications Vol.1,* No.1.

Gupta, A. and Shamim, S. Method of regression analysis for software maintenance throughout its lifecycle. Patent Application 3359/DEL/2013.

Jin, G., Song, L., Shi, X., Scherpelz, J., and Lu, S. 2012. Understanding and detecting real-world performance bugs. In *33rd ACM SIGPLAN Conference on Programming Language Design and Implementation.* pp.77–88.

Lane, H. 2000. Technology regression and verification acceptance method. US Patent No: 6,269,457.

McKellar, B. 1996. Network baselining, part i: Understanding the past to predict the future. WG Sales.

Walker, J. L., Diab, S., and Slovik, A. 1998. Method for defining durable data for regression testing. US Patent No: 6,061,643.

Zheng, P., Qi, Y., Zhou, Y., Chen, P., Zhan, J., and Lyu, M. 2014. An automatic framework for detecting and characterizing performance degradation of software systems. *IEEE Transactions on Reliability Vol.63,* No.4.

Dr. Ankur Gupta is the Joint Director at the Model Istitute of Engineering and Technology, Jammu, besides being a Professor at the Dept. of Computer Science and Engineering. Prior to joining MIET, he worked as a Team Leader at Hewlett-Packard, India Software Operations, Bangalore for 7 years. He has a wide range of software design and development experience in e-commerce and network management domains. He has 11 patents pending at the US and Indian Patents and Trademarks Office and over 40 published research papers in international journals/conferences. Prof. Gupta pursued his engineering education at BITS, Pilani from where he has a MS, Software Systems and B.E (Hons), Computer Science degrees. He completed his PhD from National Institute of Technology, India. His research interests are in cloud computing, P2P networks, network management and software engineering. He is the founding Managing Editor of the International Journal of Next-Generation Computing (IJNGC) and a recipient of the AICTE Career Award for Young Teachers. He is a senior member of the ACM, senior member IEEE and life member of the CSI.

Ms Veena Tripathi is an Assistant Professor in Model Institute of Engineering and Technology, Jammu (India). She has done her MS in Software Engineering from Dipartimento di Informatica e Telecomunicazioni University Of Trento, Italy(www.dit.unitn.it). She has completed her Internship of MS in the field of Software Testing under guidance of Prof. Paollo Tonella and Angelo Susi in Computer Science Research center:FONDAZIONE BRUNO KESSLER Trento, Italy(www.fbk.eu). She has more than 8 years of teaching Experience. She has Worked On YAKSHA project (for online preparation of Medical Entrance Examination). She has worked on many research projects during her MS program. She has the membership of International Association of Engineers (IAENG). Her research interests are in Software Testing, Reverse Engineering, Software Maintenance and Cloud Computing.