Load Testing Strategy Review When Transitioning to Cloud

Tanvi Dharmarha, Ajay Jain

Abstract - The core objective of testing is to certify the product to a quality level at which the application is ready for releasing to the end customers. Apart from functional parameters, there are many other key parameters, especially operational parameters, which play a major role in deciding how the testing is performed. This paper focusses on reviewing the strategy for load testing and changes that a testing team undergoes when transitioning their in-house infrastructure to the cloud. Further to this, the paper also talks about the advantages and efficiencies for the testing team, when shifting to cloud.

Index Terms: Cloud Testing, Infrastructure, Load Testing, Testing Efficiency,

I.INTRODUCTION

A key customer-centric goal of a testing team is, to take the product quality to a level where there are minimum possible bugs in the product. While this goal is very important to enhance user experience and bring in more goodwill and revenue to the company, another goal for testing teams is to optimize the testing cost and increase the efficiency of the overall system. A major investment goes into setting up the test infrastructure and every step in the setup has an associated time and financial cost. It is important for a testing team to evaluate the best and the most efficient testing infrastructure.

Of all the types of and variances in testing, Load testing specifically requires a robust and cost-effective solution and infrastructure to support testing. Traditionally, software vendors were setting up satisfactory in-house infrastructure for load testing.

This requires the following investments:

- Cost: Infrastructure like virtual machines, servers, and test machines. If there are multiple geographies involved, test machines in different regions to simulate real time load.
- Time: Dedicated team and experienced people to set up the infrastructure.

There is also an uncertainty of infrastructure lying unused for lack of future similar Load testing projects.

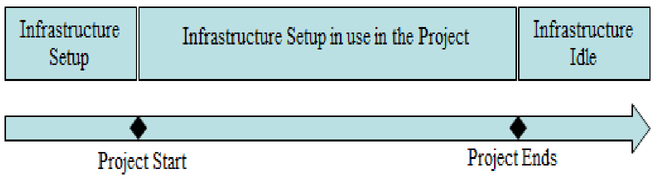

Figure 1: State of testing setup infrastructure

Manuscript published on 28 February 2014.
***Correspondence Author(s)**
Tanvi Dharmarha, Lead Software Engineer, Adobe Systems, Noida, India.
Ajay Jain, Quality Engineering Manager, Adobe Systems, Noida, India.

II.EFFICIENCY IMPROVEMENT IN A TEST SETUP

It is important for the testing team to explore and identify strategies and options to improve the system's efficiency.

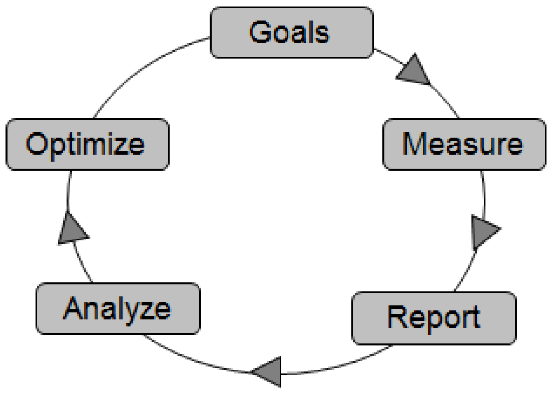

Figure 2: Efficiency Generation Cycle

The following are the key phases of an efficiency improvement plan:

- Setting up test efficiency goals.
- Measuring multiple relevant parameters on the identified goals.
- Reporting relevant metrics in a timely manner.
- Analyzing the metrics in an unbiased and ethical way.
- Concluding the metrics results with learnings from previous release or cycle and adjusting the goals to further improve and optimize.

III.IN-HOUSE LOAD TESTIING INFRA STRUCTURE ARCHITECTURE

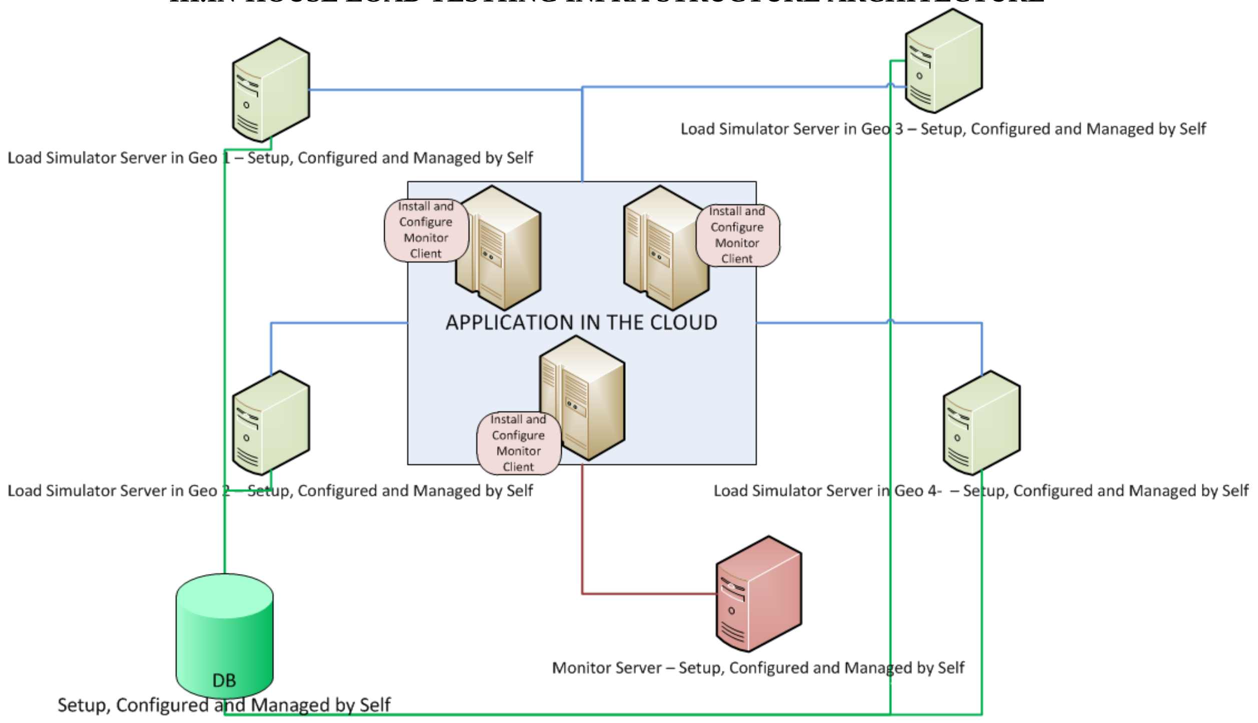
Load Simulator Server in Geo 1 – Setup, Configured and Managed by Self
Load Simulator Server in Geo 3 – Setup, Configured and Managed by Self
Install and Configure Monitor Client
Install and Configure Monitor Client
APPLICATION IN THE CLOUD
Install and Configure Monitor Client
Load Simulator Server in Geo 2 – Setup, Configured and Managed by Self
Load Simulator Server in Geo 4- – Setup, Configured and Managed by Self
DB
Setup, Configured and Managed by Self
Monitor Server – Setup, Configured and Managed by Self

Figure 3: In-house load testing infrastructure architecture

- Load Simulator Server in different Geographies: Installed and setup server in different locations like US West, US East, Europe and Asia to trigger requests to the Application.
- DB: Installed and setup DB server to persist server requests and responses in order to generate reports.
- Monitoring servers: Installed and configured performance monitoring servers to monitor network traffic, responsiveness of services, CPU utilization, disk space, and other network hardware.
- Monitor Clients: Installed and configured on all Application servers to pass the performance monitoring information to monitor server.

When transitioning the Load testing setup to the Cloud, the detailed architecture is as shown next.

IV.CLOUD LOAD TESTING INFRASTRUCTURE ARCHITECTURE

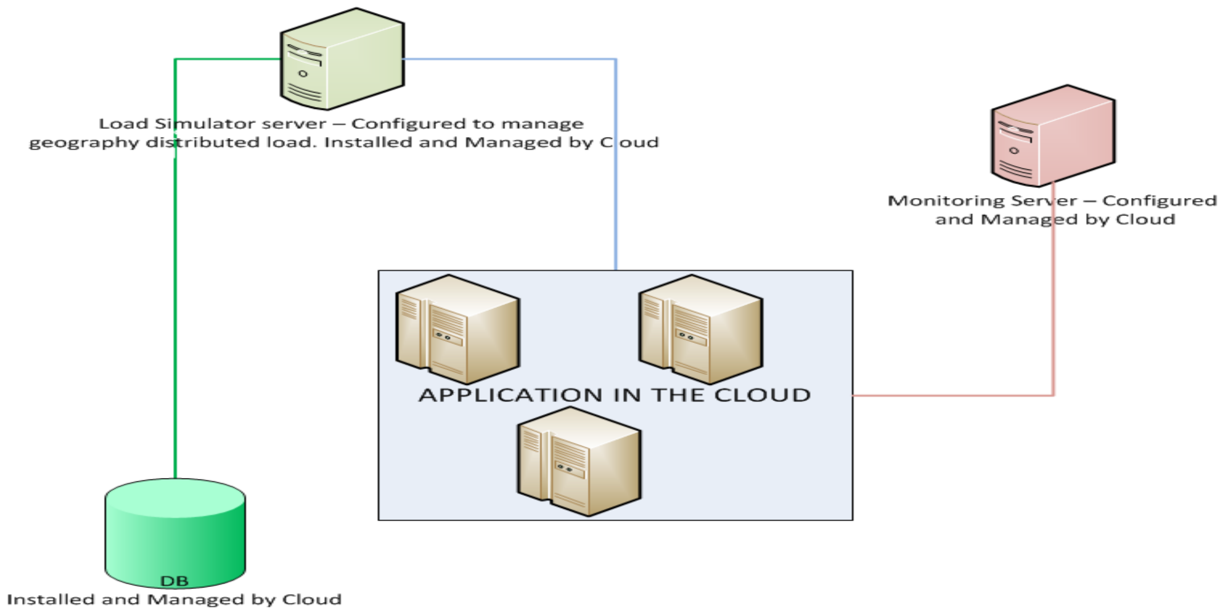
Load Simulator server – Configured to manage geography distributed load. Installed and Managed by Cloud
Monitoring Server – Configured and Managed by Cloud
APPLICATION IN THE CLOUD
DB
Installed and Managed by Cloud

Figure 4: Cloud Based Load Testing Infrastructure

- Load Simulator Server: Setup and managed by cloud. Tester can initiate load from multiple geographies at the click of a button. Post use, simulator servers can be terminated easily

- DB: Installed, Configured and Managed by the cloud provider and database connection is opaque to tester. Testers can simply draw reports without having the need to connect to the database.
- Monitoring servers: Managed by the cloud provider. Also monitoring clients are preinstalled in application by the Cloud provider.

V.COST BENEFIT ANALYSIS

The average cost of a server is around $1000, so the total cost of servers is 1000 *6 (4 Load simulator servers, 1 DB or result server, 1 Monitor server) which amounts to $6000. About 15 days goes into planning, designing and setting up the load test infrastructure, so cost of IT person is roughly half month salary which approximates to $4000. For cloud, the pricing depends on the type of instance and usage rates. For large instances, with usage rate of 0.25 $/hour and usage duration of 2 hours/day, the average monthly cost amounts to 0.25*2*22 i.e. $11. For 6 servers in use for 6 months, the total usage cost is 6*6*11=$398.

Cost	In-house	Cloud
Setup: **Cost of Servers** **Cost of IT person setting up**	 6000 4000	No setup costs
Usage:	Not much	398
Total	**10000**	**398**

Table 1: Cost comparison between in-house setup and Cloud setup

It is clear from above that architecting the load in cloud hugely saves on cost as well as time.
To optimize Load testing, one must understand the correct load distribution.
This involves monitoring:

- Load generated through real time system usage from the field, that is, from the end-users and the customers.
- Load simulated through dummy data during a test simulation.

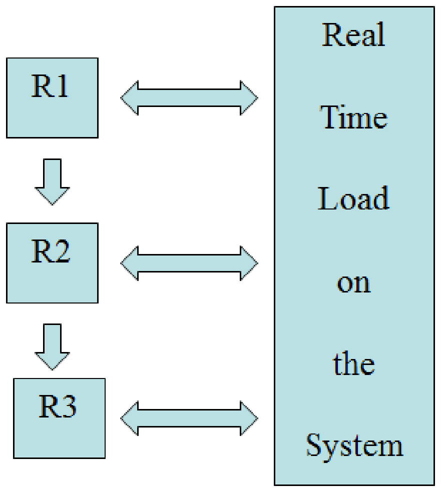

Figure 5: Loopback real time field data to test setup

In the figure above R1, R2, and R3 are the three prerelease phases of a single project cycle. R1 is released to the field for prerelease and real time load data originating from the field is monitored. The data is fed back to the testing team, which is now working on R2 phase of the project. This loop back from the field helps testing team improve their load testing data set, by matching it to the real-world load. This process is then adopted for next prerelease, that is, R3 version of the product.

VI.80-20 RULE

This rule is very effective in prioritizing when a lot of data is to be evaluated.

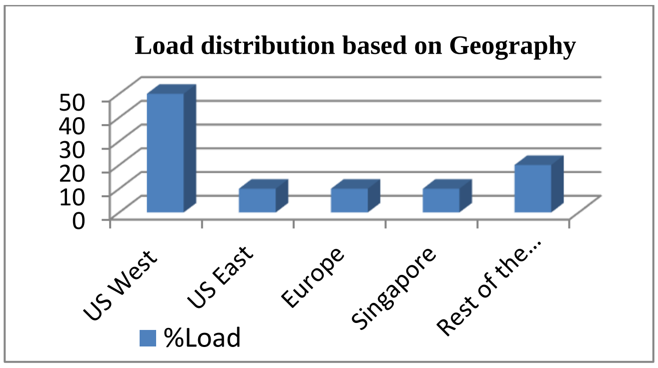

Figure 6: Real time load data spread based on Geography

A charting of real-world load, originating from different geographies, is evaluated and is charted to identify the geographical spread of the load.
The graph above indicates that 20% of the regions are generating 80% of the load. In order to load test the application, we generate load on these regions to pinpoint performance bottlenecks. On the remaining 20%, we run one round of sanity tests to increase functional coverage.
Load is injected in the application similar to the actual website traffic. If there is higher traffic from one geographical region, we inject higher load from that region to approximate real-time scenario, as much as possible.

VII.CONTROL CHARTS

Manipulating load across geographies through manual intervention is not an effective technique. An automated method is required, which can toggle load values between geographies. Control chart helps in constantly and continuously monitoring the variance between the real-world load originating from the field and the load simulated in the test setup.
A suggested statistical approach is to use automation to control and monitor the variance limits by using control charts. Control charts are effective in tracking and monitoring the stability of the data.
The following are the three key parameters of Control charts:

- Mean: Statistical value defining the average of all data points. In this case, as the goal is to spread multiple load values for a specific geography, we keep 100% of the real-time field load as the mean value. In the following graph, the green line denotes mean.

- Upper Control Limit (UCL): Statistical value defining the maximum value (or limit) that a tester wants to specify of the variation. In the following graph, this value represented by red line at 106. It means the tester takes an action when the simulated load is 106% of the field load.
- Lower Control Limit (LCL): Statistical value defining the minimum value (or limit) that a tester wants to specify the variation between real-time load from the field versus the simulated load. In the following graph, it is represented by red line at 96.

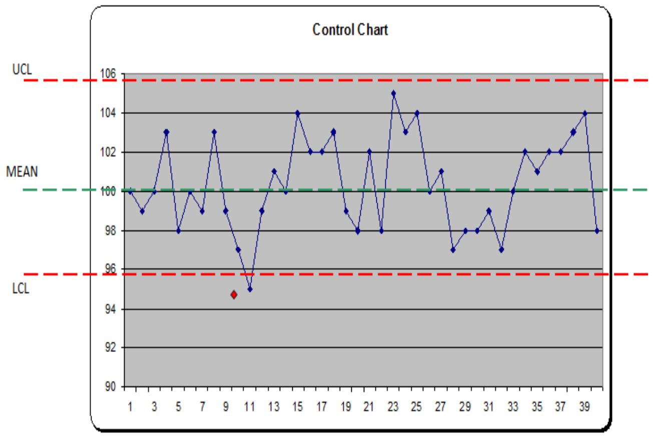

Figure 7: Control Chart

Keeping the data checks between these three statistical values helps load tester in continuously receiving the information if the simulated load in the test setup is lower than, higher than, or at par with the load in the field. This maintains the load testing within the best data set specifications. Being in the cloud, this really helps in leveraging the cloud capabilities and dynamic manipulation of the load test data set. Also, since Control charts are automated, a tester can reset the threshold values for the three key parameters depending on the real-time testing need.
For example, in order to simulate peak load scenarios, the tester can setup UCL value relatively higher or equivalent to the real time load (also, we expect real time load value itself to go up during peak load sessions). Similarly, the tester can setup lower LCL values for simulating very less load taking care of dry scenarios.
System can also be configured to generate specific alarms in case any of the UCL/LCL limits are crossed. These on the spot and dynamic data driven alarms will help tester take corrective action to bring back the system to desired state.

Study Parameters	R1	R2	R3
How many Users logging to the system	123m	98m	108m
How many New Users?	45m	50m	63m
How many on new feature-set that is pre-released?	42m	48m	59m
Which Geography is running with Max users?	US West	US West	US West
Differentials with respect to previous data in said geography?	-3m	+4m	+8m

Table 2: Control Chart Data comparison

A comparative data always helps in planning out the test strategy, data set for the upcoming release (or prerelease) version of the product/application.
Another key area that helps in adding efficiency in testing cycle is the turnaround time in debugging and isolating the bug. Lesser the turnaround time, faster will be the bug fixes.

VIII.REAL TIME ERROR ANALYSIS

Load Test tool/environment should be such that it provides real time load and error analysis. Any communication failure, request rejection or validation error should be available in real time so as to easy the process of bug isolation and eliminate performance bottlenecks. Application and infrastructure logs are authored in order to capture all relevant information of each entry point and exit point and all relevant test data used in the application testing. System monitoring tools are also recommended to use in the entire load testing process, covering CPU, Memory usage of the machine under test.
We encountered a situation where in 85-90% of the load requests were resulting in Bad Gateway error. Bad gateway error is an HTTP 502 error which means that one server had received an invalid response or did not receive a timely response from another server. The server hierarchy was as follows

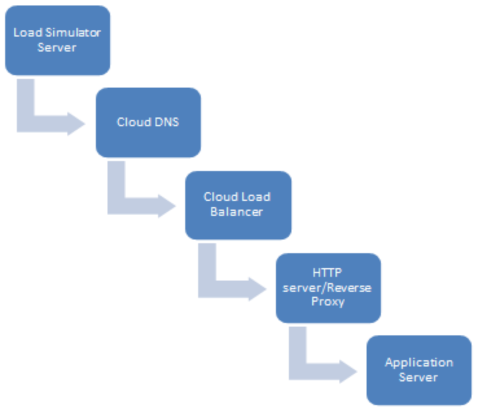

Figure 8: Server Hierarchy

The gateway error could have stemmed anywhere in this server to server communication. Load test scripts had to be updated to print HTTP headers to determine cause of failure. Following was the methodology followed.
Application Load handling capacity: 300 request/sec

S. No.	Activity Done	Result	Fix
Round 1	Simulated load of 1000 requests distributed across 4 servers	857 requests failed due to Bad Gateway	NA
Round 2	Modified script to print HTTP header in case of failure and simulated load of 500 requests distributed across 4 servers.	Bad Gateway errors begin to occur in the first few seconds. Requests were getting rejected at the Cloud Load Balancer as it was not configured to handle a load more than 150 req/sec.	Configured the load balancer to handle 500req/sec
Round 3	Simulated load of 500 request distributed across 4 servers	Entire Traffic forwarded from Load Balancer to Reverse Proxy and Bad Gateway error occurs at Reverse Proxy. Requests are not relayed to the application server.	Optimized the reverse proxy configuration files by increasing the number of worker_connections.
Round 4	Simulated load of 500 request distributed across 4 servers	Bad Gateway error still at reverse proxy but the error percentage decreased from 85 to 20%.	Optimize the ulimits, i.e. the user limits, a builds in shell utility to set restrictions on resource usage.
Round 5	Simulated load of 500 request distributed across 4 servers	All requests relayed successfully to the application server and servers also auto scaled to handle more load.	NA

IX.SCRIPT BASED LOAD SIMULATION

In the diagram above, a crawler script fetches data from the staging database (pre –production environment) and saves it in csv files. Cell values from csv files are read by the load test scripts in a loop and fed to the HTTP message (get/post). Such complete requests are injected to the application via load simulation servers that are distributed across different geographies. Parallel load is simulated on the cloud and the results of the load test run are also saved on the cloud so that they are accessible from anywhere, anytime.

a) Pseudo code

```
Step 1: Sql export 100 user accounts, 100 product
keys.
Step 2: export user_ accounts.sql to
user_accounts.csv, product_key.sql to
product_key.csv
Step 3: For each (cell in user_accounts.csv)
For each (cell in product_key.csv)
        While (Actual load simulation < load peak
value for Geo1)
        Execute HTTP request
        Save result on cloud
        Actual load_simulation ++
```

Same Step 3 is executed for Geo 2 and Geo 3 in parallel with different test data.

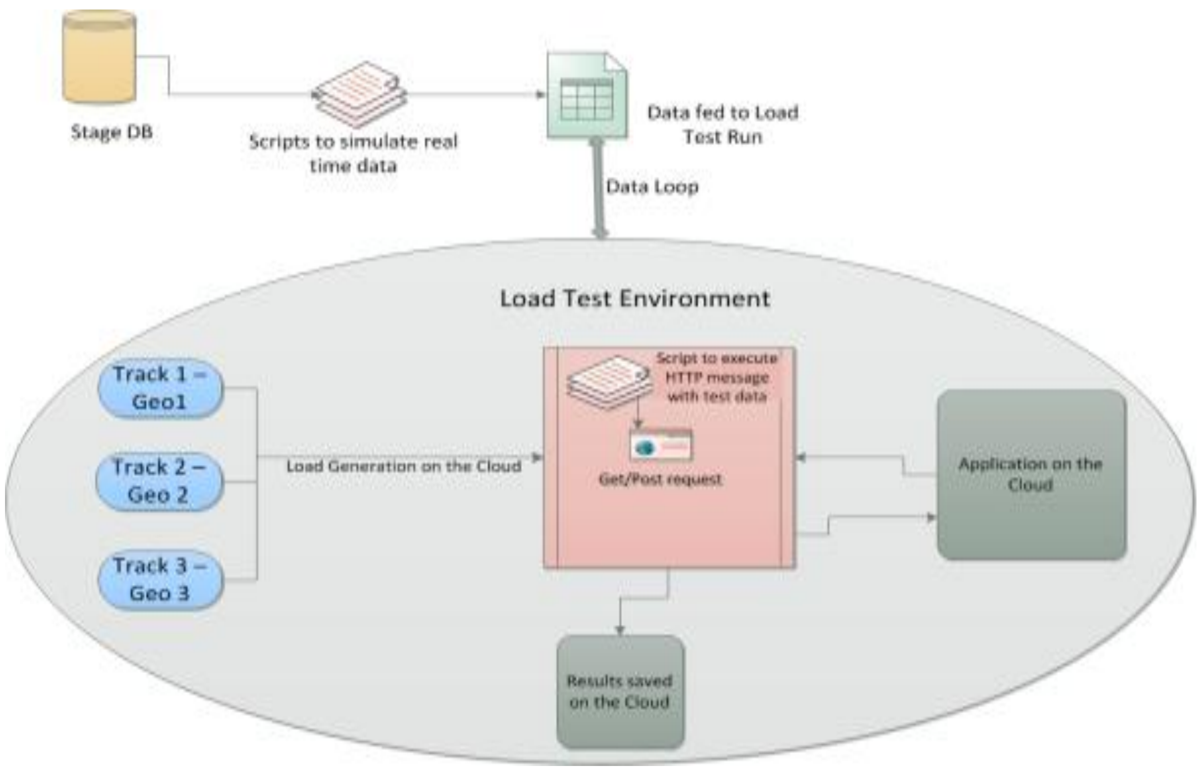
Stage DB
Scripts to simulate real time data
Data fed to Load Test Run
Data Loop
Load Test Environment
Track 1 – Geo1
Track 2 – Geo 2
Track 3 – Geo 3
Load Generation on the Cloud
Script to execute HTTP message with test data
Get/Post request
Application on the Cloud
Results saved on the Cloud

Figure 9: Load Simulation using Script

X.CONCLUSION AND KEY TAKEAWAYS

Through this study we observed there are significant benefits of Cloud infrastructure in comparison to in-house infrastructure. Further, we observed significant cost and time saving with Load testing Infrastructure transitioning to Cloud infrastructure. Some key points are listed here:

- Loop back of data from the field helps testing team improve their load testing data set, by matching it to the real-world load
- In case of large amounts of data, it is important to prioritize which type of data to target for load testing.
- Control charts are effective in tracking and monitoring the stability of the data.

We plan to do further work on this by designing a loop back mechanism between real time loads from the field with scripts that simulates load in test setup. Also, we plan conceptualizing profile based testing where we will identify methods through which automatic load profiles are generated based on the real time loads observed from the field.

XI.AUTHORS BIOGRAPHY

Tanvi Dharmarha is working as a Lead Software Engineer with Adobe Systems Inc and is leading the quality team for Anti-Piracy initiatives at Adobe. She holds an Engineering Degree in Informational Technology. Prior to Adobe she has worked with an online Trading Company, International Marketmakers Combination, based out of Amsterdam.

Ajay Jain is currently working with Adobe Systems Inc as Quality Engineering Manager responsible for managing multiple installer validation projects like Adobe Photoshop Installer and tools development. Ajay has published many technical papers with leading conferences and Journals across the world and also has a USPTO Patent to his credit. Ajay has done B.Tech from Delhi Institute of Technology, Delhi, India.