Received: 27 October 2017 | Revised: 3 May 2019 | Accepted: 7 May 2019

DOI: 10.1002/spe.2708

RESEARCH ARTICLE

WILEY

Framework for cloud-based software test data generation service

Priyanka Chawla[1] | **Inderveer Chana**[2] | **Ajay Rana**[3]

[1]School of Computer Science and Engineering, Lovely Professional University, Phagwara, India

[2]Department of Computer Science and Engineering, Thapar University, Patiala, India

[3]Department of Computer Science and Engineering, Amity University, Noida, India

Correspondence
Priyanka Chawla, School of Computer Science and Engineering, Lovely Professional University, Phagwara-144 411, India.
Email: priyankamatrix@gmail.com

Summary

This paper presents the framework of cloud-based software test data generation service (CSTS) that caters to cost-effective test data generation service in a cloud environment. In contrast to existing conventional or cloud-based testing frameworks, CSTS has a number of unique benefits. First, CSTS is designed to facilitate test data generation in minimum time and cost. Second, unlike existing frameworks which mandates clients to opt for resources to test their jobs, CSTS guides customer for selecting best cluster configuration in order to minimize the cost. While the existing models do not provide any solution for trust establishment in cloud computing services, CSTS delivers it by implementing security mechanism with the provision of role based access control. The security mechanism proposed in this paper ensures the protection of data and code of different users. Third, CSTS provides a mathematical pricing model to fulfill the expectations of customers and also to maximize the net profit of service providers. Cloud service request model has also been designed that postulates service level agreements between customers and service providers. We have evaluated, compared, and analyzed our framework and have found that it outperforms other existing cloud-based frameworks.

KEYWORDS

cloud testing, cloud-based testing service, prediction and profiling, pricing model, security model, software testing, testing service framework, trust

1 | INTRODUCTION

Cloud computing is a popular IT paradigm that provides shared pool of configurable computing resources that can be conveniently obtained and relinquished as required from any computing device such as laptop, tablet, mobile with minimal management effort, or support from service provider.[1] It provides an opportunity to software companies to focus on innovation rather than on IT functions of procurement and maintenance that helps in saving higher financial cost. Furthermore, pay-per-use model of cloud computing increases flexibility and reduces cost incurred in software development. It supports agile software development process that helps in shortening the software development life cycle, whereas conventional computing mode does support agile model of software development, but it is very complicated because any replication or change in application is very complex that should be administered throughout from beginning to end.

Software testing is the most important phase of software development and software need to be tested for all possible combination of hardware and software configurations; then, only software testing is said to be completely and adequately performed. Cloud computing has the potential to offer virtualized commodity hardware that provides unlimited storage in a cost-effective manner. Hence, software testing can be carried out very effectively and efficiently over cloud. Online test

labs available on cloud can help users to achieve quality attributes such as availability, reliability, security, performance, scalability, and elasticity.

Cloud-based software testing has the capability to offer high-quality testing at reduced cost and time with no upfront cost. It relieves the software companies from the burden of owning, reserving, and maintaining private test labs. The resource requirements can be elastically provisioned and deprovisioned depending upon the complexity and size of testing jobs. Various software industries like SOASTA, Microsoft, Rackspace, Sogeti, IBM, CloudTesting, Wipro, and HP have worked in this direction and provided various cloud-based testing models, but still very limited academic work is available on this subject. As per our literature review, very few work has been done that utilizes cloud computing infrastructure for automated software test data generation by users.

In this paper, we have presented cloud-based software test data generation service (CSTS) framework that provides test data generation service for the submitted software at optimized cost and time. In our previous work, Pareto-optimal-based test data generation framework was devised that facilitated efficient generation of test data with better coverage and fault detection capability.[2] This work is further extended by providing a job profiling and prediction mechanism that relieves customers in making important decisions of selecting appropriate cluster configuration to run a specific job. Additionally, we have implemented transparent mathematical pricing model and trusted third party (TTP)–based encryption technique that ensure secure access of data, this helps in enhancing the trust of clients.

Our proposed framework is generic enough to use any version of Apache Hadoop MapReduce to generate test data at the back-end. The main aim of the proposed framework is to provide test data generation service at optimum cost and time to service consumers. The cost of service depends on computational resources like CPU and memory used to perform a given task. By utilizing the resources effectively, service providers can offer competitive prices to the service consumers. To achieve this goal, we have used Yet-Another-Resource-Negotiator (YARN) framework that manages and schedules resources rationally and leads to reduction in the cost. Another very important feature of CSTS is that it guides users in the selection process of cluster and MapReduce configuration parameters such as number of virtual machines (VMs), type of VMs, number of mappers, and reducers per VM, etc. This helps the service providers to provide service to consumers at a reduced cost, whereas the existing cloud-based testing frameworks depend on the customer's specifications and, due to wrong selection of cluster and mapper configuration parameters, it leads to poor resource utilization, which results in higher cost.

Cloud computing services face significant obstructions in acceptance due to lack of trust among potential users. To build and sustain trust, security mechanism has been implemented and role based access control has also been provided. The proposed security mechanism ensures protection of data and code of different users that will certainly build confidence of users in using the service.

In order to become popular among potential users, cost of service should be minimized and satisfaction level of customers should be maximized. To address this challenge, mathematical pricing model has been designed to fulfill the expectation of customers and also to maximize the net profit of service providers. Cloud service request model has also been proposed that postulates service level agreements (SLA) between customers and service providers.

The primary contributions of our work are highlighted as follows:

1. Proposed the framework of CSTS that facilitates test data delivery in minimum time and cost.
2. Implementation of prompt mechanism that helps customers to select best cluster configuration.
3. Proposed a security model that can be applied in various layers. Provision of role based access control has also been implemented. Trusted third party security support is used to assure that only registered users are allowed to access the data.
4. Proposed a mathematical pricing model that fulfills the expectations of customers and maximizes the net profit of service providers. The cloud service request model has also been designed that postulates SLA between customers and service providers.
5. Experiments have been carried out to investigate the effectiveness of the proposed framework. The effectiveness of the proposed framework is assessed w.r.t cost and time by executing it in the cloud environment and at the same time ensures trust and privacy of user data.
6. Comparison of the proposed framework with other existing cloud-based testing frameworks.

This paper is structured as follows. Section 2 introduces Apache Hadoop YARN, Section 3 formulates the mathematical model of the proposed work. Section 4 describes the system model of the proposed framework. Section 5 provides the architecture overview. Section 6 includes evaluation and discussion. Section 7 discusses related work and limitations of the existing research work and Section 8 summarizes this paper.

2 | APACHE HADOOP YARN

"Yet-Another-Resource-Negotiator" or MapReduce2 is the latest version of MapReduce launched by Hadoop 2.0. This framework not only supports MapReduce processing model but also enables the execution of numerous other distributed processing application frameworks such as MPI and graph processing. The YARN layer lies above Hadoop distributed file system (HDFS) and below MapReduce processing layers. The resource management tasks are taken care by YARN layers and data processing task is carried out by processing application frameworks such as MapReduce, MPI, and graph processing. As a result, YARN is able to support clusters of large sizes more efficiently with increased scalability and agility. The resource manager optimizes cluster utilization by offering capacity scheduling, fairness scheduling, and SLAs.

The components of YARN are defined as follows.

1. **Resource Manager**: Resource manager is the main entity of the YARN cluster and acts as a master to all node managers under it. It is responsible for allocation of resources to node managers and application masters (AMs). It also monitors applications running on node managers.
2. **Application Master**: An AM is an instance of framework specific library and it works with node managers and resource managers to execute and monitor the containers with proper record of the resource consumption. It manages and monitors the execution of application and also negotiates for resource requirements from resource manager.
3. **Node Manager**: Node manager allocates resources to applications in the form of containers which represent CPU, memory, bandwidth, etc. It manages and monitors the life cycle of containers and keeps track of the usage and vigor of resources present on each node.
4. **Container**: Container is an abstract entity that represents amount of resources allocated to a specific application on a particular node. It is launched by AM on making resource request to the resource manager by invoking 'container launch' specification API. The YARN container launch specification permits AM to collaborate with node manager and to launch containers for different types of applications.

3 | CLOUD-BASED SOFTWARE TEST DATA GENERATION SERVICE: PROBLEM FORMULATION

A client submits his testing job for a particular software under test (SUT) that can be represented in the form of tuple $(u_i, g_i, tt_i, d_j, b_j, n_j, t_j)$, where u_i represents jar file of SUT, g_i is the test goal, tt_i represents the testing type, d_j is the deadline to submit test report of the submitted job, b_j is the maximum budget quoted by client, n_j is the number of VMs the client can afford, and t_j represents the type of VMs. For simplicity, it can be assumed that clients have capability to specify their processing requirement. However, we have devised the mechanism of profiling and prediction that prompts the users or clients to select an optimal Hadoop cluster configuration (HCC). However, the final decision of cluster configuration selection is taken by clients only. This provision has been made to increase the transparency and building trust among users or clients. The proposed framework aims the following.

1. **Maximization of profit**: The profit of providers is computed by finding the difference between quoted price and total cost (TC) incurred. This can be represented mathematically as follows:

$$\textbf{Maximize}\ (\text{Profit}) = QP - TC. \tag{1}$$

   From the aforementioned equation, the profit of the provider can only be maximized when TC is minimized and the difference between quoted price and TC is as large as possible. This infers that, if cost incurred to process a request is minimized without compromising the quality of service, then it leads to client's cost benefits. It has been assumed that provider specified a minimum expected return and provider accepts job only if his investment return is greater than or equal to minimum expected return.
2. **Profile and predict job's performance**: The job profile provides information regarding probable execution time, data flow, and usage of resources. The completion time of the job needs to be estimated to determine the possibility of achieving the deadline. CProfiler aims to profile and predict the job's performance and is specified in terms of five parameters, ie, I represents input parameters, SD represents sample data, T_{vm} denotes types of VM used, D represents Data consumed or generated during map and reduce phases, and T represents time of completion for map and reduce tasks
   $\text{CPROFILER}(I, SD, T_{vm}, D, T)$.

3. **Minimization of cost to clients**: The cost to client can only be minimized when provider is able to perform the job by effectively using the resources. Then only service providers would be in a position to quote lesser price. Service provider should be able to provide service at minimum cost without compromising quality of service or minimum expected return. Therefore, an algorithm has been designed that effectively completes the job in the desired deadline with lesser number of VMs. The job will be accepted only if investment return is greater than or equal to minimum investment return. The resource usage cost is calculated as the product of the number of physical servers required to host the virtual cluster and running time of the job
   Cost of job = Running time of job X Number of physical servers required to host the cluster.
4. **Minimization of job completion time**: Job completion time is sum of the time spent while processing the job (Time_R), data transfer time (Time_{DT}), time spent to deliver test report to the client (Time_D), and waiting time spent during payment to gateway service (Time_{WT}). Hence, $(J_{CT})_j = (\text{Time}_R) + (\text{Time}_{DT}) + (\text{Time}_D) + (\text{Time}_{WT})$.
   The cost of job can be reduced by minimizing the job completion time
   $(J_{CT})_j$, ie, $(J_{CT})_j < d_j$, where d_j represents deadline for job completion.

The cost of the job can be reduced by minimizing the required number and types of VMs. To help the client in making this smart decision, the profiling and prediction module has been incorporated in the proposed framework.

4 | SYSTEM MODEL

In this section, an overview of high-level architecture of the proposed framework has been presented. The main objective of the proposed service framework, CSTS, is to facilitate trustworthy and cost-effective test data generation service. As depicted in Figure 1, the system model comprises of application and platform layer functions. At the platform layer, Apache Hadoop YARN is installed to form a cluster. The application layer function comprises of client requests for testing the software. It starts with submission of SUT, test goal, testing type, and quality of service (QoS) parameters such as deadline, budget, etc. Inputs from clients are converted into a test description file (TDF) and submitted to optimizer for further processing. The job is accepted for test data generation only if software as a service (SaaS) provider is able to make sufficient profit. Acceptance approval is taken from admittance algorithm and thereafter accepted request is submitted to the starter and the TDF is saved to the HDFS. The starter invokes the optimizer which computes test data of the submitted software by forking into several MapReduce jobs. The MapReduce jobs are sent to YARN in the platform layer where all tasks pertaining to scheduling and actual execution on the nodes are performed. The monitor module keeps track of resources usage while job execution and saves it as a separate file on the HDFS. This file is used by cost evaluator for computation of cost and bill generation.

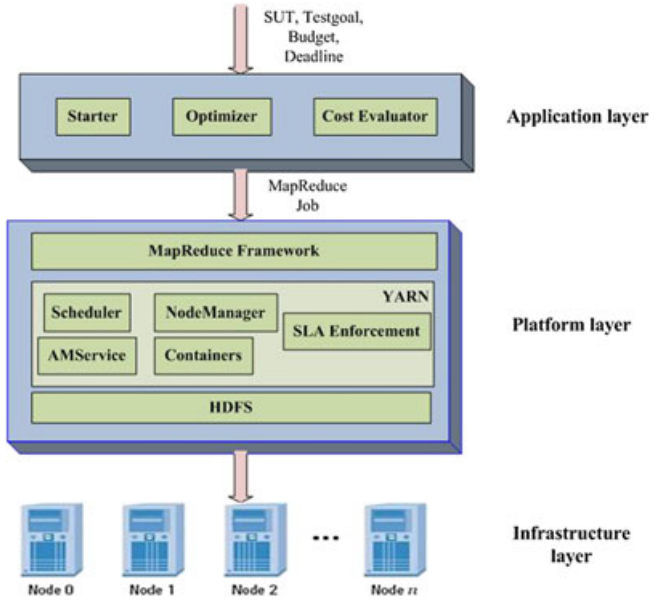

FIGURE 1 High-level architecture of proposed framework. HDFS, Hadoop distributed file system; SLA, service level agreement; SUT, software under test; YARN, Yet-Another-Resource-Negotiator [Colour figure can be viewed at wileyonlinelibrary.com]

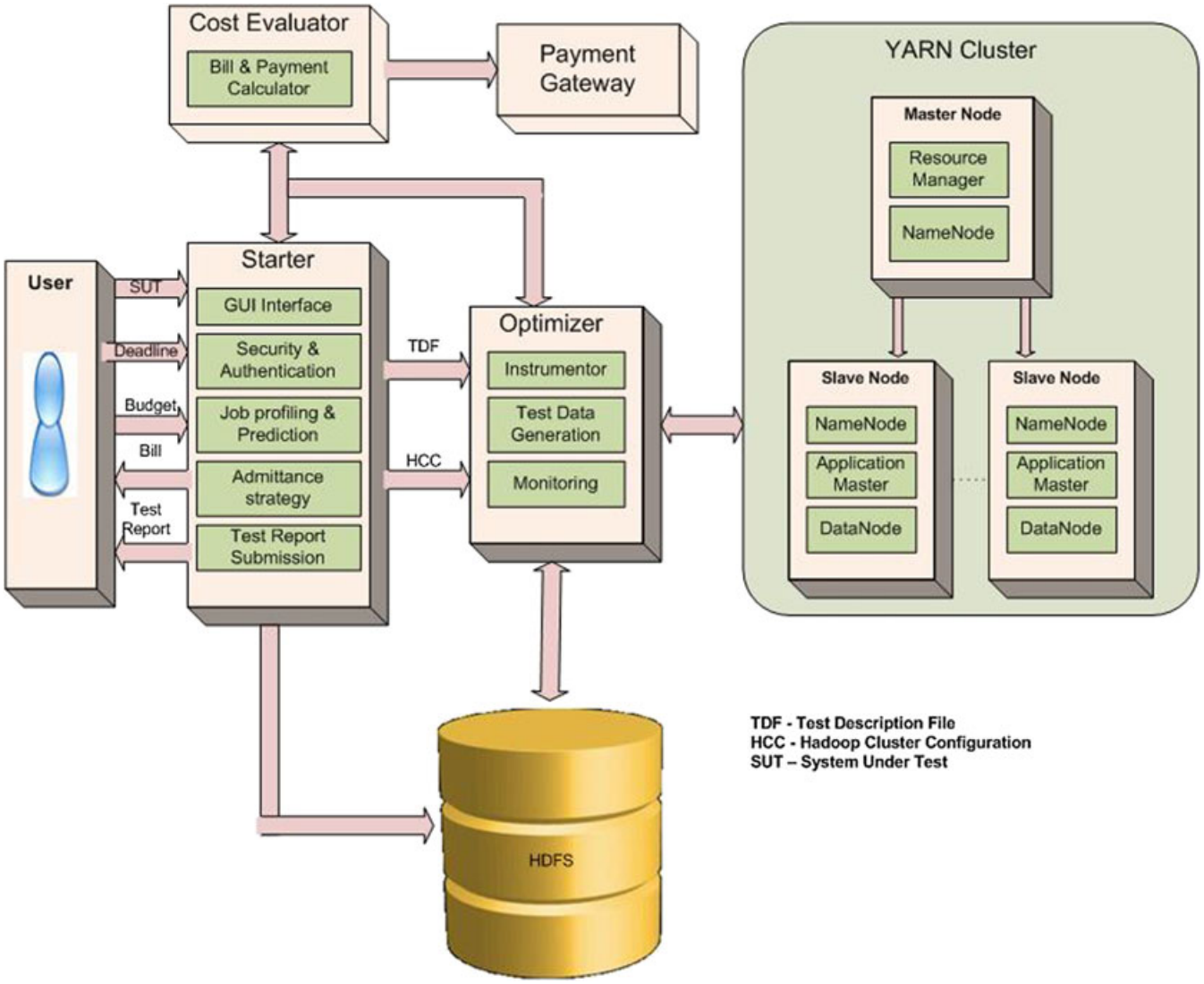

FIGURE 2 System architecture of proposed framework [Colour figure can be viewed at wileyonlinelibrary.com]

5 | ARCHITECTURE OVERVIEW

The architecture of proposed framework is based on master/slave communication model. Figure 2 shows the system level architecture and its basic components, ie, starter, optimizer, cost evaluator, and YARN cluster (YARN master and YARN slave nodes).

5.1 | Starter

It is a front end component that accepts SUT, budget, and time deadline from the client as an input. The constituent modules are GUI interface, security and authentication, job profiling and prediction, admittance strategy, and test report submission.

1. **GUI Interface:** It provides the graphical user interface where user can login and once authenticated can make request to test SUT through the web browser.
2. **Security and Authentication:** The primary objective of the proposed security model is to ensure the following:
   (a) Role-based access control for authorization.
   (b) Authentication by using symmetric cryptosystem.

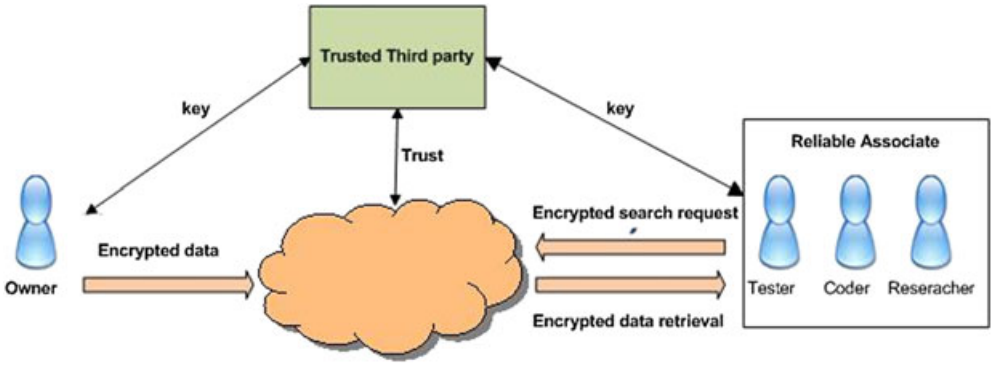

FIGURE 3 Security model of the proposed framework[Colour figure can be viewed at wileyonlinelibrary.com]

(c) TTP-based mechanism that facilitates the following:

- Encryption of uploaded code and data by using symmetric key algorithm (eg, advanced encryption standard (AES)).
- Secure exchange of secret key (SK) by using public key symmetric algorithm (eg, Rivest-Shamir-Adleman (RSA)).
- Communication with reliable associate (RA) on behalf of service provider or user.

The security challenges of cloud-based testing framework have been addressed by incorporating symmetric cryptosystem for authentication and a role based access control for authorization. Security mechanisms have been implemented at multiple stages by provision of role based access control for protection assurance of code and data of clients. Role-based access control is used since it is a unified model that supports real-world access control requirements for large scale authorizations.[3] We have defined two modes of client roles, ie, owner/user and RA. A client who has submitted a job for test data generation is treated as an owner/user. The owner may share test data with other groups of community such as testers, developers, and research groups for consultation on specific issues. These groups of people are termed as RA. The process of creation, modification, and removal of RA is named as MngRA. Reliable associate is further divided into tester (RA_T), developer (RA_D), and researchers (RA_R) classes based on the permissions granted to them. The tester is given the permissions of editing the code uploads, developer is allowed to view test data, and researchers are allowed to download the strategy of test data generation and view the test data. Identification of a client is carried out by his user name and password. Since data security is the foremost apprehension of customers using cloud services and this may be due to lack of trust between service provider and the customers,[4] therefore, to enhance the trust relationship between the parties, TTP security support has been used to assure that only registered and legitimate users are allowed to access the data.[5] The authenticated users are allowed to access the resources based upon authorization roles associated with user id. The user possessing the role of owner can create a number of RA and assign different authorizations to them labeled as (RA_T, RA_D, RA_R). The model is shown in Figure 3.
Encryption of user's code can either be performed by the user itself or this task can be automated by calling encryption and decryption algorithm for each write and read request of a code file or SUT on HDFS. In our proposed framework, we have designated this task to starter module of the proposed framework. The scheme of proposed security model that ensures protection of SUT or code is shown in Figure 4 and illustrated as follows.

(a) The submitted SUT is encrypted by using the AES. We have adopted 128-bit AES to perform user-side encryption.[6] Advanced encryption standard is one of the most popular block cipher algorithm and suitable to handle HDFS blocks. The submitted file or SUT will be divided into fixed size multiple blocks $b_1, b_2....b_n$, where the size of each block can be up to 16 bytes. Each block of data is processed through ten recurring sequence of encryption iterations before the final encrypted file is produced. During each iteration of encryption, four distinctive operations are performed on block of data b_i using a unique key k_i. Initially, a byte-to-byte substitution (S_i) is performed on the block b_i using a table called as substitution box. In the second step, row-by-row permutation function (P_i) is performed. This is followed by another round of data substitution (S_j) and in the fourth step, bitwise exclusive or (XOR) operation is performed between current block and its corresponding key. The output can be represented as $RS_i = l_i \oplus (g_i, S_j(g_i, P_i(g_i, S_i)))$, where RS_i represents the output of i^{th} recurring sequence and l_i is the encryption key for data block b_i.
The aforementioned step has been shown as (a) in Figure 4.

FIGURE 4 Exchange of messages between stakeholders to ensure data confidentiality. AES, advanced encryption standard; HDFS, Hadoop distributed file system; SUT, software under test

(b) Once the SUT is encrypted, the cipher text is transmitted to HDFS over a secure communication channel. The aforementioned step has been shown as (*b*) in Figure 4.

(c) The SK is encrypted by using RSA algorithm for secure transmission to TTP. This has been achieved in two steps. In the first step, SK is encrypted by using the public key of TTP(P_{TTP}). Hence, encrypted copy of secret key $C_{SK} = E(P_{TTP}, SK)$.
In the second step, digital signature is produced by using the private key of user ($R \rightarrow user$) so that it can be authenticated by the TTP
$DS = E_{R \rightarrow user}(E(P_{TTP}, SK))$.
The proposed service framework would generate different keys for different users and these keys would be transmitted securely to TTP.
The above aforementioned has been shown as ($c1$), ($c2$), and ($c3$), respectively, in Figure 4.

(d) Upon request from RA for access to a particular SUT and its associated test data or report, TTP verifies the identity of RA by performing decryption such that $D_{P \rightarrow RA}(E_{R \rightarrow RA}(MD)) = H(d_i)$, where H is a cryptographic hash function

implemented on a small block of data d_i. Once the identity of RA is verified, corresponding SK of user U_i is retrieved and sent to the corresponding RA by using RSA encryption algorithm such that $(E_{P\rightarrow RA}(SK_{user}))$.
The aforementioned step has been shown as $(d1)$, $(d2)$, and $(d3)$, respectively, in Figure 4.

The benefits of the proposed security scheme are as follows.

(a) Reduction in time and cost: The encryption of data and code has been done by using symmetric key algorithm (AES), which is comparatively easier to apply than asymmetric key algorithms. The symmetric key algorithms are usually fast and help in the reduction of time. Moreover, cipher text produced after the application of symmetric key algorithm is same as that of input text whereas the size of cipher text produced after applying asymmetric key algorithm is larger than the input text. Thus, it helped us to achieve reduction in storage space.
(b) The use of RSA algorithm has helped us in secure exchange of SK among the proposed framework, TTP, and RA. Hence, benefits of both symmetric and asymmetric algorithms have been incorporated in the proposed scheme.

3. **Job Profiling and Prediction:** This module is responsible for suggesting customers to select an appropriate cluster configuration that helps in achieving timely completion of jobs within specified performance and cost constraints. As per our knowledge, in the present state of the art, users are required to specify the Hadoop cluster configuration as per the size of the job.[7,8] The user specifies values for all configuration parameters. For the parameters whose values are not specified by the user, system administrator uses the default values. These parameters have big impact on performance and cost of the job. The burden and complexity of this process can be overcome by job profiling and prediction and, thus, we have designed a module named as CProfiler to carry out this task. It is somewhat similar to the work proposed by various eminent researchers.[8-10] Our approach is different in the way as explained as follows.

CProfiler uses dynamic instrumentation to collect the job profiles. A MapReduce job profile collects some unique aspects of data flow or cost during job execution at the task level or phase level within tasks. A profile is compact representation of job execution that captures information both at task and subtask levels. The job profile provides information regarding execution time, data flow, and usage of resources. CProfiler functions in three parts.

(a) First part collects information about operation of MapReduce jobs and creates an analytical expression.
(b) Second part profiles the tenant job with sample data and produces the parameters related to the job and infrastructure.
(c) Third part estimates the job completion time.

The MapReduce jobs are executed in three phases, namely, map, transfer, and reduce. Normally, these three phases execute in sequential manner for a job. The completion time for a job can be estimated by summation of time required to compete these three phases

$$T_i = T_M + T_t + T_R, \tag{2}$$

where T_M, T_t, and T_M are the time required to complete the map phase, transfer phase, and reduce phase, respectively; then, let N_M represent the number of cycles required to execute the tasks of map phase, and then T_M can be measured as

$$T_M = N_M * (D_M \div PB_M), \tag{3}$$

where D_M is the data consumed during execution of map phase and PB_M is phase bandwidth (PB) of map phase.
In the same manner, we can calculate T_t and T_R.
Given a job J, input data size S_I, sample data size SMP_i, and network bandwidth between VMs NT, CProfiler calculates the estimated completion time of the job
$CProfile(S_j, VM_T, SMP_i, NT, J, D_M, D_R, T_M, T_R) \rightarrow T_j$,
where VM_T is type of VMs; D_M and D_R are data consumed or generated during map and reduce tasks.
T_M, T_R represents completion time for map and reduce tasks.
The PB of individual phases is determined by using the similar analytical model as proposed in other works.[8-10] It works by computing following three parameters:

- PB;
- data consumed;
- waves.

CProfiler profiles the job by executing it on a single machine with a sample of input data. It calculates the amount of data consumed and generated during each phase and also the execution time for each task. The log files of this process

are used to collect the information that leads to determine the PB. Afterwards, based on this information, it calculates the data consumed and estimated job completion time.

(a) Phase bandwidth: Phase bandwidth can be calculated as follows:
$PB_R = (IO_B, D_R/t)_{\min}$.
During transfer phase, reduce tasks performs read operations on intermediate data and write operations on the disk. Afterwards, data is read from disk and finally stored in memory before it is consumed by reduce phase. Let NB be bandwidth of network; the transfer PB can be calculated by inversing the summation of inverse of disk I/O bandwidth and minimum of network bandwidth NB and disk I/O bandwidth
$Transfer_B = IO_B * (IO_B, NB)_{\min} \div IO_B + (IO_B, NB)_{\min}$.

(b) Data consumed:
For a job j with M map tasks and R reduce tasks, we take size of input S_i,
data consumed by each map task $= S_i \div M$ bytes, and
data consumed by each reduce task $= S_i \div (A_M * R)$ bytes,
where A_M is average number of records output by map tasks per input second.
Data generated by each reduce task $= S_i \div (A_M * R * R)$ bytes,
where A_R = average number of records output by reduce tasks per input record

(c) Waves (N_M): For a job using x VMs with M_s map slots per VM, maximum number of mappers $= x \times M_s$,
map tasks execute in $M \div (x \times M_s)$ waves,
and reduce tasks execute in $R \div (x \times R_s)$ waves,
where R_s is number of reduce slots per VM.
Now, for map phase, $T_M = M \div (x \times M_s) \times ((S_i \div M \div PB_M)$
In the same manner, T_R and T_t can be calculated.
The estimated job completion time is evaluated using the following:
$T_j = T_M + T_R + T_t$

$$\begin{aligned} T_j = {} & [M \div (x \times M_s) \times ((S_i \div M) \div PB_M)] \\ & + [R \div (x \times R_s) \times ((S_i \div (A_M \times R)) \div PB_T)] \\ & + [R \div (N \times R_s) \times ((S_i \div (A_M \times A_R \times R) \div PB_R)] \,. \end{aligned} \tag{4}$$

4. **Admittance Strategy**: The job is submitted to optimizer only if it is approved by admittance module of the starter. The job is admitted only if service provider is able to make profit by executing the job. The flowchart of admittance strategy is shown in Figure 5. After the admission of job, TDF is stored on HDFS and the optimizer is invoked for test data generation.
5. **Test Report Delivery:** The generated test suite is delivered to the client in the form of JUnit test file after receiving payment.

5.2 | Optimizer

This component is responsible for computing test data of submitted application. Test data generation (TDG) strategy has been implemented by using the computational intelligence of genetic and particle swarm optimization algorithm. The test data generation process has been parallelized by distributing the task among multiple mappers. The mapper module performs the tasks in parallel pertaining to test data generation for multiple class files in a jar file. The strategy has been explained in detail in our previous work.[2] It also keeps track of number and type of resources utilized during the generation of test suites and its output acts as input to the cost evaluator.

5.3 | Cost evaluator

The proposed framework is meant to operate in both public cloud infrastructures and private clusters. Unlike owning a private cluster, the economical way is to acquire the required resources on pay-as-you-go subscriptions. The resources represent computing machines and storage space. The incurred cost will be much lesser than the cost of acquiring and operating a private cluster of the same size.

The cost evaluator evaluates the cost of test data generation based on four parameters, namely, test goal, cost budget, size of SUT, and testing type. The test goal defines the stopping criteria of TDG. The cost budget indicates the upper limit of the chargeable amount the client could afford. The testing type represents the alternative choices for testing levels such

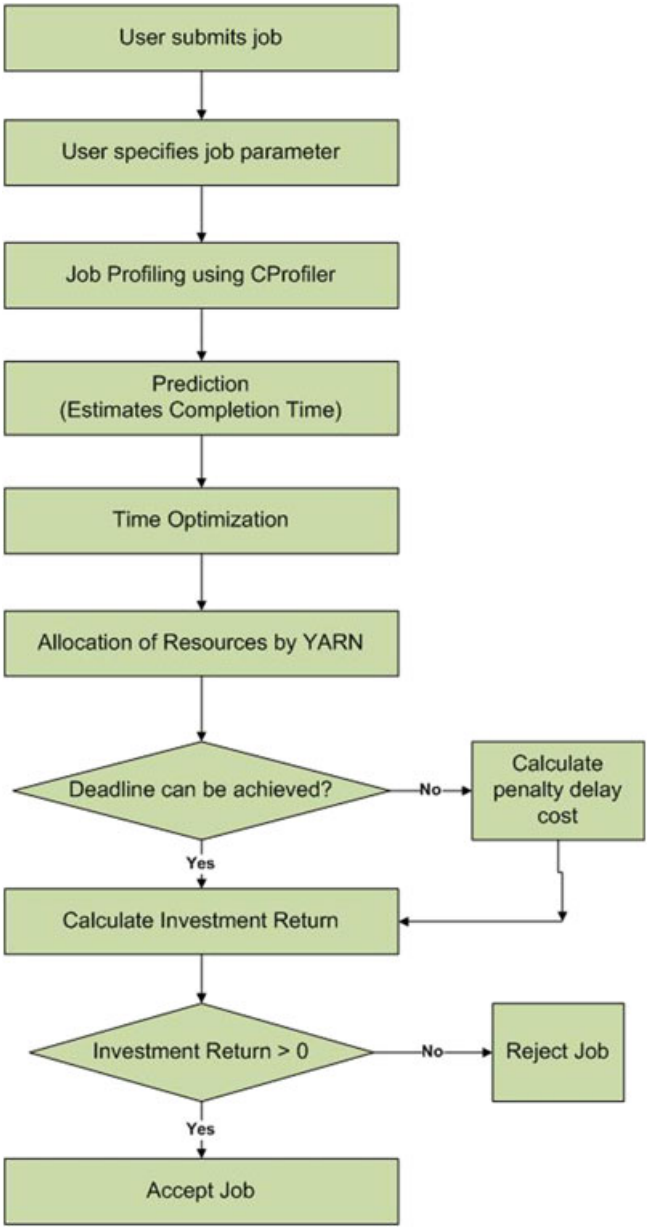

FIGURE 5 Flowchart of admittance strategy. YARN, Yet-Another-Resource-Negotiator [Colour figure can be viewed at wileyonlinelibrary.com]

as unit testing, system testing, etc. Presently, this work offers unit testing to the client. In the future, other testing levels may also be offered to the clients in the form of various choices.

The number and type of resources depend upon the size of the SUT, test goal and cost budget, and the number and type of resources required is estimated based on the values of these inputs. For branch coverage, the client will be charged a telescoping amount $x for each percentage point of coverage. For crash points, $y will be charged for each crash inducing defects. Both $x and $y is proportional to the size of the SUT

$$\text{Bill} = n_1 * x + n_2 * y, \tag{5}$$

where n_1 denotes the percentage coverage of SUT, n_2 denotes the number of faults detected, and

Bill <= Cost Budget.

Pricing model

Pricing is the process whereby a service provider fixes price at which it will provide its services to the users. Pricing approach can be either fixed or dynamic. The factors that affect the pricing in cloud computing are as follows[11-14]:

- initial cost,
- lease period,
- QoS,
- types of resources,
- maintenance cost,
- comparison with other service providers, and
- supply and demand of services.

A customer evaluates the pricing model mainly by pricing, QoS, and usage type. The pricing can be fixed regardless of volume. In the fixed price plus per unit charges, the customer pays a fixed price plus a unit rate. In assured purchase volume plus per unit price rate, customer pays fixed price for certain quantity and pays per unit if usage exceeds the limit. In per unit rate with a ceiling, customer pays per unit rate with a certain limit. In per unit price, customers pay per unit price of resources. The service provider with high level of QoS (such as availability, security, privacy, scalability, integrity, etc) attracts an increased number of loyal customers. The usage period defines the time period during which a customer can utilize the services as per SLA between the two parties. The usage period can be perpetual or by subscription or pay-per-use.

The most common pricing model used by service providers is pay-per-use model, in which user is charged a fixed price for each hour VM usage. In recent years, cloud computing has driven shift in the computing paradigm, which allows service providers at different layers (application, platform, and infrastructure) to offer computing services on demand and pay as per the usage. Nowadays, customers using the SaaS model need not to worry about purchasing a license from a software company for installing, upgrading, and maintaining the software. Rather, customer buys services on rent from the service providers and relies on them for the acceptable QoS requirements, upgradations, and maintenance. To fulfill the expectations of customers and maximize the net profit of SaaS providers, cloud service request model needs to be designed in such a way that it postulates SLAs between clients and service providers. Service level agreement defines the officially authorized agreement between a service provider and a user that states quality of service and its analogous proceedings. It also states the response time, customer budget, and penalty in case of performance breakdown among other things. Considering the aforementioned explanation and various work related to design of profit model of service providers,[11-14] we have defined user service request of the proposed framework as follows:

$f(\text{USR})=f(B, T_{CL}, T_{DL}, \beta, TC)$.

The parameters to the $f(USR)$ are maximum price allocation (B), job completion time (T_{CL}), penalty delay time (T_{DL}), penalty rate (β), and TC Incurred (TC). The pricing model of our proposed framework is given as follows.

Let a new user submit a service request at submission time (arrival time) T_a to the service provider. The new user provides the following parameters:

- maximum price (budget) represented as B,
- deadline represented as T_{DL}, and
- penalty rate represented as β.

Let service provider provide x types of VM, where each VM type has P_{VM} price. The prices per GB charges for data transfer in and out by the provider are P_{in} and P_{out}, respectively.

Let D_{in} and D_{out} be the data-in and data-out required to process the user request. Let TC_i be TC incurred to provider while processing the user request on VM_i of type x. Then, the profit gain by the provider is defined as

$$\text{Profit} = B - TC_i. \tag{6}$$

The TC incurred by the provider for accepting the new request depends on the request processing cost (C_R), data transfer cost (C_{DT}), VM initiation cost (C_{VM}), and penalty delay cost (C_{PD}). Thus, TC (TC) is given by

$$TC = C_R + C_{DT} + C_{VM} + C_{PD}. \tag{7}$$

The request processing cost (C_R) is dependent on request processing time (T_R) and hourly price of VM (HP_{VM}).

Thus, (C_R) is given by the following:

$$C_R = T_R * HP_{VM}. \tag{8}$$

Data transfer cost can be described as summation of cost of both data-in and data-out

$$C_{DT} = D_{\text{in}} \times P_{\text{in}} + D_{\text{out}} \times P_{\text{out}}. \tag{9}$$

The VM initiation cost is dependent on the type of VM initiated. Let T_{VM} represent time taken for initiating VM

$$C_{VM} = P_{VM} \times T_{VM}. \tag{10}$$

The penalty delay cost (C_{PD}) represents the penalty levied to service provider for violating the SLA. It is dependent on penalty rate β and penalty delay time period (T_{PD}).

Let T_{CL} be the job completion time. The T_{PD} is defined as

$$T_{PD} = T_{CL} - T_{DL} \tag{11}$$

and

$$C_{PD} = T_{PD} \times \beta. \tag{12}$$

The job completion time for the new request to be processed on VM_i consists of VM initiation time (T_{VM}), request service processing time (T_R), data transfer time (T_{DT}), and penalty delay time (T_{PD})

$$T_{CL} = T_{VM} + T_R + T_{DT} + T_{PD}, \tag{13}$$

where data transfer time T_{DT} is summation of time taken to upload the input (T_{in}) and download the output (T_{out})

$$T_{DT} = T_{\text{in}} + T_{\text{out}}. \tag{14}$$

The investment return (*IR*) to accept new user request per hour on a particular VM_i is calculated based on profit and time (T_{CL})

$$IR_i = \text{Profit}_i \div T_{CL}. \tag{15}$$

The (*IR*) can be maximized by maximizing profit, which, in turn, can be achieved by minimizing *TC*. Minimization of *TC* is subject to following constraints:

1. $T_{PD} \to 0$, which can be achieved by job profiler by optimizing the time required for job completion.
2. C_{DT} can be minimized either by storing local data on HDFS or by using *S*3 bucket whichever is nearer to the location of Hadoop cluster. Hence, *TC* primarily depends upon C_R and C_{VM}.
   Therefore,
$$TC_i = \sum_{i=1}^{l} \sum_{j=1}^{m} R_j \times HP_{VMj} + P_{VMi} \times T_{VMi}. \tag{16}$$
3. Minimize job completion time: The cost of job can be reduced by minimizing the required number and types of VM_i. To help the client in making this smart decision, the profiling, and prediction module has been inculcated in our proposed framework.

The job will be accepted only if *IR* is greater than or equal to minimum investment return. If the job is accepted, it will be executed as per the sequence defined in the proposed framework. After completion of test data generation task, cost evaluator generates bill according to the pricing model explained above. The client pays the bill through payment gateway and, after successful payment, payment gateway sends confirmation message to the starter module, which then get test suites from HDFS and submits to the client. The method used to generate bill (cost to customers) using the proposed pricing model is explained as follows with the help of working example.

Working example: using pricing model to generate the actual bill

Suppose a user submits a service request with following parameters:

1. Maximum Price (Budget) B = \$1000;
2. $T_{DL} = 60$ min;

3. Penalty Rate = 10;
4. Job Size = $4 \times 1\,000\,000$ MI.

Let user select the cluster configuration of Amazon *EC*2 with two VMs after getting the prompt from the service provider. Suppose the cluster configuration is 1933*MIPS*, \$0.12 per hour, Ubuntu 12.04, 4*GB* RAM, 2*ECU*, and 160*GB* disk space.

After the selection of number and types of VM, estimated profit is calculated as follows:

1. Service Request Processing Time $T_R = (4 \times 1\,000\,000) \div 1933$.
   Hence, $(T_R) = 34.4$ min.
   Therefore, $C_R = T_R \times HP_{VM}C_R = 7.2 \times 34.4$.
   Hence, $C_R =$ \$247.68 per VM.
2. $C_{VM} = P_{VM} \times T_{VM}$.
   Hence, $C_{VM} = 7.2 \times 5$ (Mean value).
   Hence, $C_{VM} =$ \$36 per VM.
3. C_{DT} is minimized by using local storage or using *S*3 bucket.
   Therefore, $C_{DT} =$ \$20 per VM.
4. $T_{DT} = 0$, as we profile and predict the job to achieve the deadline.

As per equation, $TC = 247.68 + 247.68 + 36 + 36 + 20 + 20$.
Hence, $TC =$ \$606.68.
Let the minimum expected profit of service provider be 20% of the user budget. Min Expected Profit $= (1000 \times 20) \div 100$.
Min Expected Profit = \$200.
Min Expected Investment Return = *Profit* ÷ *deadline*.
Hence, Min Expected Investment Return = \$200 ÷ 1.
Min Expected Investment Return = \$200.
Expected Investment Return = $Profit \div T_{CL} = (1000 - 606.68) \div 0.866$.
Hence, Expected Investment Return = \$453.97,
where $T_{CL} = 52$ mins
$T_{CL} = 0.86$ hour.
Therefore, Expected Investment Return > Minimum expected investment return. As a result, the user job will be accepted.
$Bill = n_1 \times x + n_2 \times y$
Let $x =$ \$3
$y =$ \$.08 per crash point.
Therefore, $Bill = 90 \times 3 + 8000 \times .08$
Hence, $Bill =$ \$650.

5.4 | Yet-Another-Resource-Negotiator cluster

It constitutes YARN master and YARN slave nodes. The YARN master is invoked by optimizer module of our proposed framework for execution of MapReduce jobs to generate test data of SUT. The YARN slave node includes node manager, data node, and containers.

Yet-Another-Resource-Negotiator master

The YARN master is invoked by optimizer for execution of MapReduce jobs to generate test data of the submitted SUT. It manages several slave nodes and jobs executed over them. The two main components of YARN master are scheduler and AM service. Scheduler is responsible for monitoring and distribution of various resources (CPU, memory, and bandwidth) to slave nodes. It also handles remote procedure call interface with client such as application submission, application termination, and obtaining queue information and cluster statistics. An Application master is an instance of framework-specific library and it works with node managers and resource managers to execute and monitor containers and their resource consumption. It negotiates for resource containers required to execute specific jobs. The AM service keeps track of all applications, which are being executed on node manager, and maintains a list of live and dead AMs. It is also responsible for managing register/unregister requests from AMs, and allocation and deallocation of container requests from the AM.

Yet-Another-Resource-Negotiator slave

It includes node manager, data node, and containers. The node manager manages each and every slave node in YARN cluster and monitors the containers and their lifecycle. In addition, it keeps track of health and resource usages of YARN clients. 'Container' is an abstract entity that corresponds to resource elements (CPU, disk, network, etc) necessary for execution of applications. A resource request is made by an application for specific resources and is granted by scheduler in the form of container.

5.5 | Job execution flow

The job is said to be submitted when admittance module approves request of a client to test the software. The admittance module accepts job only if service provider gains profit by executing the job. The profit is calculated with the help of pricing model described in the previous section. Once it is approved, optimizer module invokes TDG to generate test data by creating MapReduce job and invoking several mappers and reducers on the slave machines in the YARN cluster. After that, the role of YARN cluster comes into the picture. The steps of job execution are summarized as follows and are depicted in Figure 6.

1. **Job Submission**: Client submits the job to starter by submitting SUT, test goal, testing type, and by stating its budget and time deadline.
2. **Job Id**: Client receives its Job ID.
3. **Save Job Data**: The job data is saved on HDFS or S3 bucket.
4. **TDF file generation**: Test description file (TDF) file is generated. It contains the details needed for test data generation. This file is given as an input to the optimizer.
5. **HHC file generation**: The HCC file is generated by prediction module of our proposed framework. It contains preferable HCC. It helps clients to intelligently choose an optimal HCC for test data generation at minimum cost and time.
6. **Start Optimizer**: Optimizer is started by initializing the population and generating random population. It is further accompanied by a call to the modules like instrumentor, TDG, and monitor.[12]

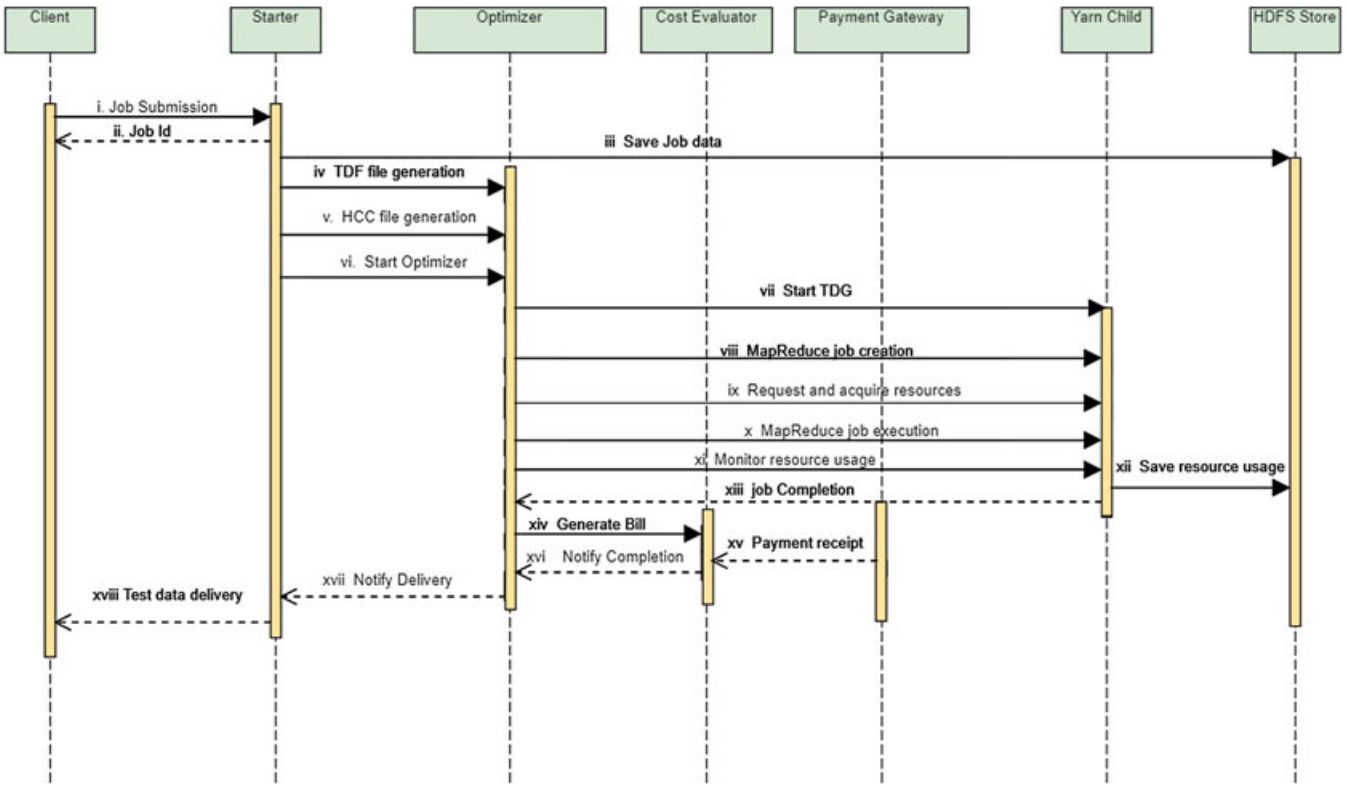

FIGURE 6 Sequence diagram of proposed framework. HCC, Hadoop cluster configuration; HDFS, Hadoop distributed file system; TDF, test description file; TDG, test data generation; YARN, Yet-Another-Resource-Negotiator [Colour figure can be viewed at wileyonlinelibrary.com]

7. **Start TDG**: Test data generation is started.
8. **MapReduce Job Creation**: MapReduce job is created by selecting Hadoop cluster intelligently as per the recommendations of profile and predict module of the proposed framework.
9. **Request and Acquire Resources**: The YARN cluster grants resources based on the request and the granted resources are acquired for test data generation.
10. **MapReduce Job Execution**: MapReduce job is executed on the acquired resources.
11. **Monitor Resource Usage**: Resource usage is monitored by the monitor module of optimizer, which instructs the cost evaluator to generate the bill.
12. **Save Resource Usage File**: Resource usage file is saved on the HDFS for further use and reference.
13. **Job Completion**: Job completion notification is sent by YARN cluster to the cost evaluator.
14. **Generate Bill**: Upon receiving the job completion notification, cost evaluator generates bill. The information of generated bill is sent to the payment gateway.
15. **Payment Receipt**: The payment gateway upon receiving the payment notifies the cost evaluator about it.
16. **Notify Completion**: Completion report is notified to the optimizer by the cost evaluator.
17. **Notify Delivery**: Optimizer instructs the starter for delivery of test report and JUnit file.
18. **Test Data Delivery**: Test data in the form of JUnit file is delivered to the client.

6 | EVALUATION AND DISCUSSION

In this section, we have analyzed the performance of the proposed framework over local cluster as well as on AWS cluster. The proposed framework is assessed w.r.t cost and time by executing it in the cloud environment, and also ensures trust and privacy of user data.

6.1 | Experimental setup

The performance evaluation of the framework has been carried out on a public cloud Hadoop cluster over Amazon elastic compute cloud (EC2)[15] as well as on a local Hadoop cluster.

Public Cloud: The experiments have been carried out by using Amazon EC2 compute instances. For the evaluation purpose, two different types of Amazon EC2 instances have been used. These instances vary in the number of vCPU, memory allocated to them, instance storage, and network performance. Ten nodes cluster have been formed in which m1.large instance has been made as master (namenode and resource manager), while other nine m1.medium instances acted as slaves (nodemanager and datanodes). Each node runs Ubuntu 12.04 and Apache Hadoop version 2.2.0. HDFS consists of a master (name node) and multiple slaves (data nodes).

Local Cluster Setup: We took 10 nodes in the cluster, each having the hardware configuration as Intel Core i7-4980HQ CPU as processor with four cores having 2.8 GHz processor speed, 8 GB of RAM, software configuration as Hadoop version 2.2.0, and Ubuntu 12.04 on the machines. One of the nodes is made the master, while other nine acted as slaves. The network speed is 1 Gbps and the nodes are connected through a single switch.

6.2 | Performance evaluation

Hadoop distributed file system can be described as scalable and fault-tolerant distributed file system that can be installed very easily and efficiently on low-price hardware. The architecture of HDFS is based on master/slave framework in which there is a single master node and arbitrary number of slave nodes. The master node is called as name node whose goal is to regulate file system namespace as well as file-access operation of multiple slave nodes. The slave node is named as data node and is responsible for the file storage and storage device management. HDFS utilizes a traditional hierarchical file structure in which files can be created and stored within directories. Files are bifurcated into blocks before storing it on data nodes so that a file can be accessed in parallel for read and write operations. The replication factor of each block is kept at three by default and mapping between blocks of a file and datanodes is being maintained by master node. MapReduce programming model has the ability to process several HDFS blocks in parallel. In our model, the input data (SUT, test data, etc) submitted by the user has been encrypted and decrypted before it is written and read from the HDFS. We use 128-bit AES encryption algorithm as it is most appropriate to handle HDFS blocks. There are various modes of operation for AES such as ECB, CBC, OFB, CFB, CTR, and XTS; we have selected AES ECB modes as it supports concurrent computation and is thus more suitable for distributed environment–based computing. In the next section, the

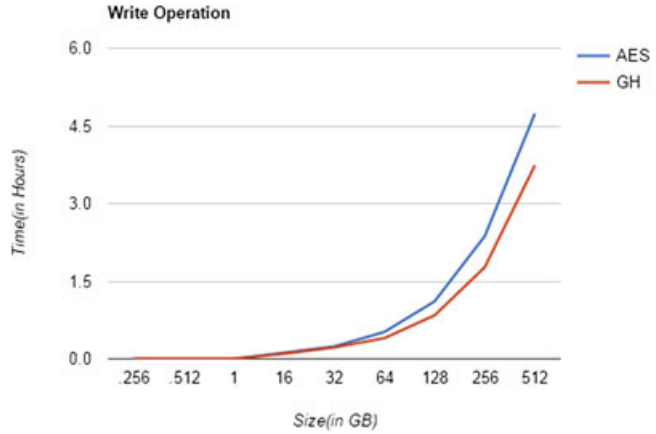

FIGURE 7 Time of advanced encryption standard (AES) versus generic Hadoop distributed file system (GH) in write operation [Colour figure can be viewed at wileyonlinelibrary.com]

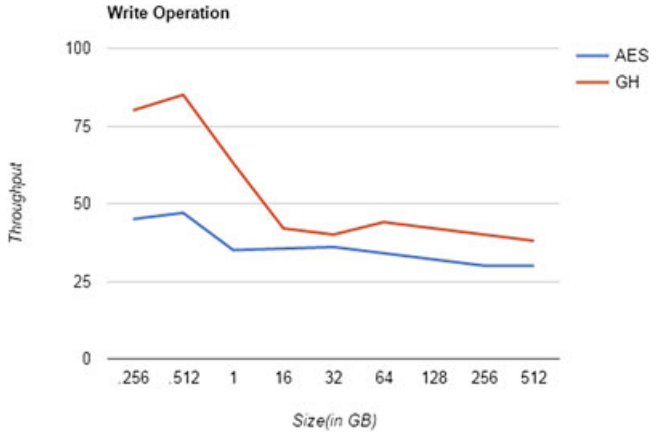

FIGURE 8 Write throughput of advanced encryption standard (AES) versus generic Hadoop distributed file system (GH) [Colour figure can be viewed at wileyonlinelibrary.com]

performance of encrypted HDFS and unencrypted HDFS (generic HDFS) has been compared for write and read operations w.r.t time and throughput.

6.2.1 | File write and read operation

The file is written in the form of encrypted chunked blocks to datanode by HDFS client. The blocks are encrypted by implementing AES encryption algorithm in Java. The HDFS client is also responsible for replication of encrypted blocks to other data nodes. Generic HDFS and encrypted HDFS have been compared w.r.t time, as shown in Figure 7. The size of the file has been varied from 256 MB to 512 GB. It clearly depicts the time taken to write a 512 GB file to HDFS is 224.56 minutes and 284.44 minutes has been taken to encrypt a file in HDFS, which exemplifies performance degradation of 27%. The throughput of writing files in generic HDFS is 38 Mbps, whereas throughput in case of AES encrypted HDFS is 30 Mbps. The same has been portrayed in Figure 8.

With the implemented algorithm, reading of encrypted file stored in the form of blocks in HDFS is carried out in parallel by map tasks at HDFS datanodes. The decryption of the encrypted file is also performed in parallel which helps in attaining better performance than file-write operations. Figures 9 and 10 show the performance of MapReduce jobs on unencrypted or encrypted HDFS. We observed 294.25 minutes was taken for unencrypted HDFS for 512 GB file, while 316.04 minutes for the encrypted HDFS. The overhead for decrypting file is 7% (maximum) in the case of 512 GB file.

The aforementioned experiments lead to the interpretation that, although there is degradation in performance while uploading encrypted files to HDFS, it helps in attaining trust and privacy of users data.

6.3 | Cost comparisons: local cluster versus AWS

In addition to aforementioned analysis, we also evaluated the cost required to set up our cloud-based framework locally in the lab and the equivalent system required with its associated cost.

To perform the experiments, we have selected the *t2.medium* AWS instance.[16] The price of t2.medium = \$0.052/hour. Hence, the cost of 10 AWS instances is

$$C_{aws} = \$0.52. \tag{17}$$

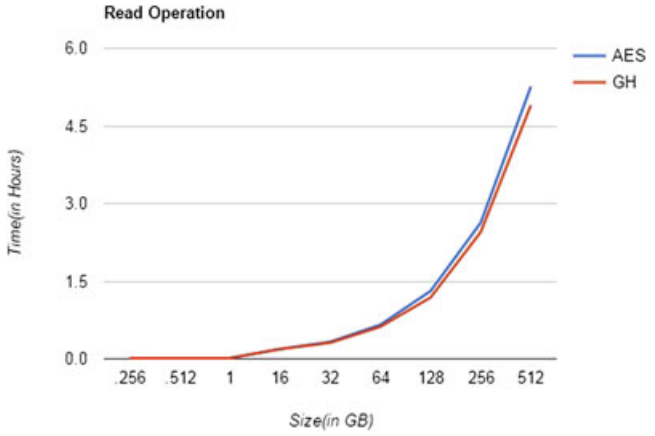

FIGURE 9 Time of advanced encryption system (AES) versus generic Hadoop distributed file system (GH) in read operation [Colour figure can be viewed at wileyonlinelibrary.com]

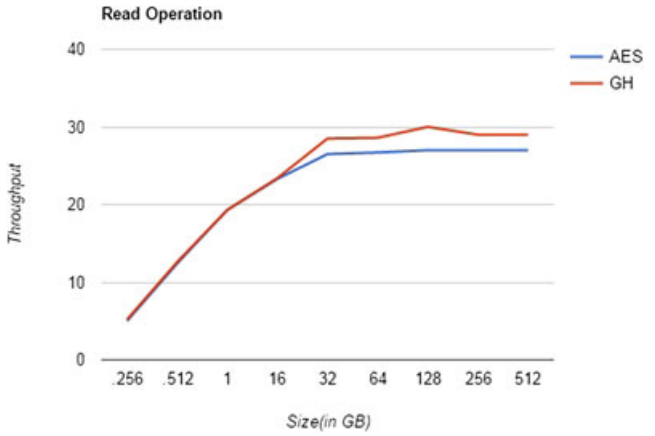

FIGURE 10 Read throughput of advanced encryption system (AES) versus generic Hadoop distributed file system (GH) [Colour figure can be viewed at wileyonlinelibrary.com]

In the case of local cluster setup, the cost of machine (C) is a factor of initial cost of machine (i_c), cost of maintenance (c_m), and cost of installations (c_i)

$$C = i_c + c_m + c_i. \tag{18}$$

Total cost of ten machines (TC) = $CX10$.

Cost of maintenance (c_m) depends upon cost of space (c_s), cost of electricity (c_e), cost of cleaning (c_c), cost of furniture (c_f), and cost of personnel (c_p)

$$c_m = c_s + c_e + c_c + c_f + c_p. \tag{19}$$

Cost of installations (c_i) is a function of cost of license (c_l) and cost of expertise of personnel ($c_e p$)

$$c_i = c_l + c_e p. \tag{20}$$

When we compare TC with the machine setup on AWS, we find that TC is strictly higher in many folds than the cost of 10 AWS instances. Hence, the cost of local cluster is much higher than the cost of AWS machines that are used over internet. Moreover, there are no difficulties in installations, procurement, and maintenance. The AWS machines are released after the completion of task. It allows considerable amount of savings.

6.4 | Time comparisons: local cluster versus AWS

In this section, we have compared the time requirements for the local cluster setup and AWS machines.

The total time for AWS cluster setup Time_{aws} is the sum of time for installations (T_i), time for cluster setup (T_{cs}), and time for performing the experiment (T_{pe})

$$\text{Time}_{aws} = T_i + T_{cs} + T_{pe}. \tag{21}$$

The time requirements for local cluster setup (Time_{lcs}) depend on the time for procuring machines(T_{pm}), time for installations on local cluster (T_{il}), time for local cluster setup T_{lcs}, and time for performing the experiment (T_{pe})

$$\text{Time}_{lcs} = T_{pm} + T_{il} + T_{lcs} + T_{pe}. \tag{22}$$

It can thus be easily proved that Time_{lcs} is greater than Time_{aws} as T_{pm} is usually at higher end as compared to the machine allocations on AWS.

From the aforementioned mathematical equations, it is clearly depicted that the cloud-based framework for test data generation is more efficient and cost effective than existing models.

7 | RELATED WORK

Several studies have been carried out that anticipate increase in demand of cloud computing in the upcoming years, which clearly reflects that there would be prospect of tremendous increase in utilization of infrastructure as a service over internet.[17,18] Software testing being one of the safest workload that can be put onto cloud as it does not involve any business sensitive data.[19] In spite of the aforementioned facts, very little work has been reported on cloud based software testing frameworks. Therefore, in this paper, we have designed a framework for cloud-based testing with stronger focus on QoS parameters such as fault detection capability, resource utilization, security, and costs.

In this section, related research work in the field of cloud-based testing has been exhibited. In our previous work,[20] we have categorized cloud-based testing models into seven categories based on the adopted research model. The existing cloud-based testing models have been analyzed and described in the work of Chana and Chawla.[21] In our previous work,[22] a hybrid particle swarm optimization (PSO)–genetic algorithm (GA)–based sequential automated TDG for object-oriented programs was presented. Later, in our previous work,[2] Apache Hadoop MapReduce and Pareto-optimal-based test data generation framework was devised that facilitated efficient generation of test data with better coverage and fault detection capability. The work presented in this paper is the extension of our previous work[2] to help customers in making important decisions such as selection of appropriate hardware configuration for the execution of specific job by providing a job profiling and prediction mechanism. Additionally, it provides secure access and transfer of data by implementing TTP based encryption mechanism. Transparent pricing model has also been provided to help the testing providers to charge customers as per the usage of services. The proposed framework also takes care of the profit interests of testing providers by only allowing the execution of the jobs that can generate good revenues for them. The effectiveness of the framework has been tested empirically w.r.t time and cost and results have been presented in Section 6. Some of the existing popular testing models is shown in the comparison Table 1 and is also described as follows.

D-Cloud is a cloud-based software testing environment used for testing of distributed systems.[23] This environment uses VMs with Eucalyptus private cloud environment. It facilitates fault tolerance testing by finding device faults with the help of VMs. If we compare it with our testing framework, we can say that tester can execute the test jobs designed by him, but test data will not be generated automatically without his intervention.

Virtualized-aware automated test service utilizes HP LoadRunner to generate load and automatically evaluates the performance of SAP R/3 system operating in a Xen-based environments.[24] It can however only supplements the service lifecycle management system and only supports the testing of compatible applications. Ganon and Zilbershtein[25] suggested performance testing framework for network management system, which facilities testing of distributed system containing VoIP private branch exchange networked through SIP. The authors have make use of emulation agents to write application-level test cases.[25] The authors have also illustrated that automatic execution of tests could be done in considerable less amount of time. On the other hand, our proposed framework generates test data automatically using soft computing technique. At the same time, it also uses Apache Hadoop MapReduce framework.

We also refereed to contribution made by Ciortea et al[26] and Bucur et al[27] toward Cloud9 development. They based their efforts on parallel symbolic testing, which calibrates to large clusters of machines. Inspired by the research of Ciortea et al,[27] Bucur et al[27] proposed a new symbolic environment model that includes all essential aspects of the POSIX interface like synchronization, networking, processes, threads, IPC, and file I/O. Besides implementing the load balancing for automated testing of UNIX utilities, the aforementioned authors have primarily focused on devising the parallel

TABLE 1 Comparison chart of existing cloud-based testing frameworks

Testing Framework	Testing Approach	Automated Test Data Generation	Parallelization Framework	Test Bed	SUT	Performance Adequacy Criteria	Test Adequacy Criteria	SLA	Pricing Model	Prediction Mechanism	Validation
Yeti	Automated Random Testing	Yes	Apache Hadoop	Amazon EC2	Java.lang	Speedup	Bugs detection	No	No	No	No
VATS	Performance Testing using HPLoadRunner	No / Test execution only	No	Xen	SAP/R3	Service Performance	—	No	No	No	No
D-Cloud	Fault Injection Testing	No / Test execution only	No	QEMU and Eucalyptus	Distributed Application	Cost; Time	—	No	No	No	No
Cloud-9	Symbolic Execution	Yes	No	Amazon EC2	UNIX utilities	Speedup	Line Coverage	No	No	No	No
GridUnit	Regression Testing	No/ Test execution only	No	Grid	JUnit Test case Execution	Speedup	No	No	No	No	No
Joshua	Regression Testing	No / Test execution only	No	Jini	JUnit Test case Execution	Speedup	No	No	No	No	No
Korat	Constraint Testing	Yes	Google MapReduce	No	Google Application	Object Graph Visualization	Constraint satisfaction	No	No	No	No
NMS	Performance Testing	No	No	No	Simulated networks	Scalability; Time	No	No	No	No	No
TaaS	Prototype of TaaS Over Cloud	No	No	Cloud	Web Application	Elasticity	No	No	No	No	No
HadoopUnit	Regression Testing	No/ Test execution only	Apache Hadoop	Hadoop Local Cluster	JUnit Test case Execution	Speedup	No	No	No	No	No
Linda et al	Unit Testing	Yes	Apache Hadoop	Hadoop Local Cluster	One Open Source Library	Speedup; Coverage	Line Coverage	No	No	No	No
CSTS Our Proposed Framework	Unit Testing	Yes	Apache Hadoop	Amazon EC2	Open Source Library file size varied from 256 MB to 512 GB	Speedup; secure effective resource Utilization, Scalability; %Increase in APFD score, %increase in coverage per time	Branch coverage; Fault detection	Yes	Yes	Yes	Yes

Abbreviations: EC2, elastic compute cloud; NMS, network management system; SLA, service level agreement; SUT, software under test; VATS, virtualized-aware automated test service; TaaS, Testing as a service.

version of the symbolic execution engine, whereas we have developed the automated TDG using Apache Hadoop MapReduce for object oriented software comprising more than one class. The implementation of our approach would be much easier with the utilization of Apache Hadoop MapReduce, which can take care of tasks like fault tolerance, load balancing, and the coordination of the execution of parallel tasks in the distributed environment.[28] Furthermore, our testing framework has the advantage of providing user friendly interface for job submission. The transparent cost model that we have developed is also cost effective as it charges the customers on the basis of cost per job.

Lastovetsky[29] has made significant contribution in the field of parallel testing of computer software systems. He also demonstrated regression testing for a distributed programming system in a parallel environment with a speed up of 7.7 on two quad processor workstations. Lastovetsky[29] and GridUnit[30-32] have executed JUnit test cases in parallel. Both the tools were based on master slave architecture where master controls the various slave nodes to execute test cases in parallel that speeds up the testing process. Joshua used the Jini framework for parallel distribution of regression test suites over several nodes, and the GridUnit employed grid computing. In the work of Parveen et al,[33] the authors designed the HadoopUnit, which carries out unit testing using JUnit for distributed execution framework. The HadoopUnit utilizes MapReduce primitives for distributive execution of test cases over the cloud. The important files like testing tools and test cases had been used by mapper node and uploaded to distributed file system. Thereafter, the master instructs the mapper nodes for execution of test cases and the reducer collects the test case reports and stores them in the distributed file system. An elementary case study[33] done with HadoopUnit on a 150-node cluster has shown a speedup of 30x in execution time. The aforementioned work emphasize that multiple nodes should be used for the execution of already generated test data. However, it deficits the user support for automatic designing of intelligent test cases and other features of cloud-based testing framework proposed by us.

Misailovic et al[34] devised a new algorithm that accepts complex test inputs. This algorithm named as Korat algorithm requires input by the user as imperative predicates and finitization that limits the test input size. Imperative predicates and finitization are java methods, so the tester must have sound knowledge in java programming language, whereas, in our strategy, there is no such condition from the user. The user is only required to upload the bytecode of the software, and our strategy automatically generates test cases for all the testable methods and classes in an object oriented program. Moreover, we have developed user-friendly interface for job submission and a transparent cost model that charges the customers on per job basis. Oriol and Ullah[7] designed YETI, which is a random testing tool over cloud in which test programs are written in Java and .Net. The authors utilized the Apache Hadoop MapReduce for distributing the task of test case generation over Amazon's EC2. However, Yeti needs more research to be done in the field of automatic mapping of testing sessions over multiple nodes for more than one class and defining test adequacy criteria of test cases for better detection of faults in random testing. On the contrast, our proposed strategy generates optimal test cases very easily and efficiently in minimum time using branch coverage and fault detected in SUT as test adequacy criteria. This strategy can enable automatic distribution of all the classes present in software over several nodes in a Hadoop cluster for parallel generation of automatic test suites. Moreover, we have designed a complete framework that provides convenient user interface for job submission and transparent cost model that charges the customers as per their jobs.

Yu et al proposed the novel model token as a service for the provision of testing resources to end users. The authors utilized computing resources efficiently by devising their scheduling and dispatching algorithms. They have examined the tolerance of the framework by raising load of test task and have also analyzed the total computing time by dispersing into two components, ie, test task scheduling and test task processing time.[35] On the other hand, we devised automatic test data generation and we utilized Apache Hadoop MapReduce to inculcate features like load balancing, fault tolerance, and scheduling of MapReduce jobs over Hadoop cluster.

Geronimo et al[36] devised a strategy to generate test data using traditional GA with Hadoop Map Reduce. While we have adopted the hybrid strategy employing the concept of PSO, GA, and Pareto optimality principle to identify best optimal test suites in minimum possible time. The authors have only reported the strategy for automatic test data generation, whereas we have designed the complete framework with several features.

In the work of Ferrer et al,[37] the authors worked on evolutionary algorithm for multiobjective test data generation problem. The objectives chosen by the authors were branch coverage and oracle cost. They compared direct multiobjective approach of test data generation with combination of a mono-objective algorithm and multiobjective test case selection optimization. To accomplish this task, they utilized four state-of-the-art multiobjective algorithms and two mono-objective evolutionary algorithms along with multiobjective test case selection based on Pareto efficiency. Their findings indicated better values for oracle cost in case of direct multiobjective approach. Maximum branch coverage was achieved with the second variant of multiobjective test data generation problem. On the contrary, our proposed framework utilized multiobjective and Pareto optimality approaches for test data generation problem. In addition, we have also

parallelized our proposed hybrid approach consisting of genetic and PSO algorithms over the cloud that helped us in attaining cost-effective solution to the test data generation problem.

Maenhaut et al[38] demonstrated the benefits of migrating the legacy software to the cloud. The authors investigated and confirmed the long term benefits of migrating applications (such as nurse call systems, schedule planner, etc) for managing medical appointments that are widely used within hospitals. The conclusion made by the authors is similar to our research work in which we proved effectiveness of test data generation framework over cloud. Although our proposed strategy works with different set of applications, our vision for the significance of working on cloud environment is same as that of the work of Maenhaut et al.[38]

In the work of O'Shea et al,[39] the authors developed methodology to generate automated test cases through the transformation of design models, whereas our work is complete framework that not only assists in the generation of test data, but also provides a mechanism for effective utilization of resources, protection of data and code of submitted jobs, maximization of profit of service providers, as well as customers. Drave et al[40] presented virtualized test automation framework implemented by DellEMC to maximize the use of underlying infrastructure for the existing test automation tools such as Selenium, TestComplete, etc. In contrast to this, our work caters to cost-effective test data generation service using YARN framework that manages and schedules resources rationally and leads to a reduction in the cost. The CSTS facilitates users by providing them with the most appropriate cluster configuration parameters like number and type of VMs and MapReduce configuration parameters like number of mappers and reducers per VM. It also provides solution for trust establishment in cloud computing services by implementing security mechanism.

8 | CONCLUSIONS

In this work, we have presented a CSTS that facilitates unit testing of the SUT with a stronger focus on QoS parameters such as fault detection capability, resource utilization, security, and cost. Our proposed framework provides users with the most valid and appropriate Hadoop and cluster configuration for better performance in terms of time and cost. Apart from this, we have also designed a transparent pricing model that fulfills the expectations of customers and maximizes the net profit of service providers. Experimental results conducted in cloud environment clearly demonstrate the effectiveness of our framework. Thus, CSTS is comprehensive, cost effective, and efficient automatic software test data generation service framework.

ORCID


Priyanka Chawla https://orcid.org/0000-0002-4135-0686

REFERENCES

1. Mell P, Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Last Accessed October 2017. 2011.
2. Chawla P, Chana I, Rana A. Cloud-based automatic test data generation framework. *J Comput Syst Sci*. 2016;82(5):712-738.
3. Sandhu R, Ferraioli D, Kuhn R. The NIST model for role-based access control: towards a unified standard. In: Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC'00); 2000; Berlin, Germany.
4. Patel H, Patel D. A review of approaches to achieve data storage correctness in cloud computing using trusted third party auditor. In: Proceedings of the International Symposium on Cloud and Services Computing; 2012; Mangalore, India.
5. Rizvi S, Cover K, Gates C. A trusted third-party (TTP) based encryption scheme for ensuring data confidentiality in cloud environment. *Procedia Comput Sci*. 2014;36:381-386.
6. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication. https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf. Last Accessed October 2018.
7. Oriol M, Ullah F. Yeti on the cloud. In: Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops (ICSTW); 2010; Paris, France.
8. Palanisamy B, Singh A, Liu L. Cost-effective resource provisioning for MapReduce in a cloud. *IEEE Trans Parallel Distrib Syst*. 2015;26(5):1265-1279.
9. Herodotou H, Babu S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *Proc VLDB Endow*. 2011;4:1111-1122.
10. Verma A, Cherkasova L, Campbell RH. Profiling and evaluating hardware choices for MapReduce environments: an application-aware approach. *Performance Evaluation*. 2014;79:328-344.
11. Rana OF, Warnier M, Quillinan TB, Brazier F, Cojocarasu D. Managing violations in service level agreements. In: proceedings of the 5th International Workshop on Grid Economics and Business Models; 2008; Gran Canaris, Spain.

12. Yeo CS, Buyya R. Service level agreement based allocation of cluster resources: handling penalty to enhance utility. In: Proceedings of the 7th IEEE International Conference on Cluster Computing; 2005; Boston, MA.
13. Irwin DE, Grit LE, Chase JS. Balancing risk and reward in a market-based task service. In: Proceedings of the 13th International Symposium on High Performance Distributed Computing; 2004; Honolulu, HI.
14. Linlin W, Garg SK, Buyya R. SLA-based admission control for a software-as-a-service provider in cloud computing environments. *J Comput Syst Sci*. 2012;78(5):1280-1299. https://doi.org/10.1016/j.jcss.2011.12.014
15. Amazon Web Service. http://www.aws.org/instance. Last Accessed October 2018.
16. Amazon Web Service. http://www.aws.org/instance/price. Last Accessed October 2018.
17. Gartner Report. http://www.gartner.com/newsroom/id/2613015. Last Accessed October 2018.
18. Merill Lynch Report. http://readwrite.com/2009/11/25/merrill-lynch-cloud-computing. Last Accessed October 2018.
19. Fujitsu. Confidence in cloud grows, paving way for new levels of business efficiency. Fujitsu Press Release, November 2010. http://www.fujitsu.com/uk/news/. Last Accessed October 2018.
20. Chawla P, Chana I, Rana A. Empirical evaluation of cloud-based testing techniques: a systematic review. *SIGSOFT Software Eng Notes*. 2012;37(3):1-9.
21. Chana I, Chawla P. Testing perspectives of cloud based applications. In: Mahmmood Z, Saeed S, eds. *Software Engineering Frameworks for Cloud Computing Paradigm*. London, UK: Springer; 2013. http://www.springer.com/computer/communication+networks/book/978-1-4471-5030-5
22. Chawla P, Chana I, Rana A. A novel strategy for automatic test data generation using soft computing technique. *Front Comput Sci*. 2015;9(3):346-363.
23. Banzai T, Koizumi H, Imada R, Hanawa T, Kanbayashi T, Sato T. D-cloud: design of a software testing environment for reliable distributed systems using cloud computing technology. In: Proceedings of the 2010 10th IEEE ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society; 2010; Melbourne, Australia.
24. Gaisbauer S, Kirschnick J, Edwards N, Rolia J. VATS: virtualized-aware automated test service. In: Proceedings of 5th International Conference on Quantitative Evaluation of Systems; 2008; St Malo, France.
25. Ganon Z, Zilbershtein IE. Cloud-based performance testing of network management systems. In: Proceedings of the IEEE 14th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks CAMAD'09; 2009; Pisa, Italy.
26. Ciortea L, Zamfir C, Bucur S, Chipounov V, Candea G. Cloud9: a software testing service. *SIGOPS Oper Syst Rev*. 2010;43(4):5-10.
27. Bucur S, Ureche V, Zamfir C, Candea G. Parallel symbolic execution for automated real-world software testing. In: Proceedings of the Sixth Conference on Computer Systems; 2011; Salzburg, Austria.
28. Apache Hadoop MapReduce. http://www.hadoop.apache.org/mapreduce. Last Accessed October 2017.
29. Lastovetsky A. Parallel testing of distributed software. *Inf Softw Technol*. 2005;47(10):657-662.
30. Kapfhammer GM. Automatically and transparently distributing the execution of regression test suites. In: Proceedings of the 18th International Conference on Testing Computer Software; 2000; Berlin, Germany.
31. Duarte A, Cirne W, Brasileiro F, Machado P. GridUnit: software testing on the grid. In: Proceedings of the 28th International Conference on Software Engineering; 2006; Shanghai, China.
32. Duarte A, Wagner G, Brasileiro F, Cirne W. Multienvironment software testing on the grid. In: Proceedings of the 2006 Workshop on Parallel and Distributed Systems: Testing and Debugging; 2006; Portland, ME.
33. Parveen T, Tilley S, Daley N, Morales P. Towards a distributed execution framework for JUnit test cases. In: Proceedings of International Conference on Software Maintenance; 2009; Edmonton, Canada.
34. Misailovic S, Milicevic A, Petrovic N, Khurshid S, Marinov D. Parallel test generation and execution with Korat. In: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering ESEC-FSE'07; 2007; Dubrovnik, Croatia.
35. Yu L, Tsai W, Chen X, et al. Testing as a service over cloud. In: Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering; 2010; Nanjing, China.
36. Geronimo LD, Ferrucci F, Murolo A, Sarro F. A parallel genetic algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites. In: Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation; 2012; Montreal, Canada.
37. Ferrer J, Chicano F, Alba E. Evolutionary algorithms for the multi-objective test data generation problem. *Softw: Pract Exper*. 2012;42(11):1331-1362.
38. Maenhaut PJ, Moens H, Ongenae V, De Turck F. Migrating legacy software to the cloud: approach and verification by means of two medical software use cases. *Softw: Pract Exper*. 2016;46:31-54. https://doi.org/10.1002/spe.2320
39. O'Shea D, Ortin F, Geary K. Virtualized test automation framework: a DellEMC case study of test automation practice. *Softw: Pract Exper*. 2018;49:329-337. https://doi.org/10.1002/spe.2658
40. Drave I, Hillemacher S, Greifenberg T, et al. SMArDT modeling for automotive software testing. *Softw: Pract Exper*. 2018;49:301-328. https://doi.org/10.1002/spe.2650

How to cite this article: Chawla P, Chana I, Rana A. Framework for cloud-based software test data generation service. *Softw: Pract Exper*. 2019;49:1307–1328. https://doi.org/10.1002/spe.2708