Accepted Manuscript

Cloud-based automatic test data generation framework

Priyanka, Inderveer Chana, Ajay Rana

PII: S0022-0000(16)00002-7
DOI: http://dx.doi.org/10.1016/j.jcss.2015.12.001
Reference: YJCSS 2945

To appear in: *Journal of Computer and System Sciences*

Received date: 9 March 2015
Revised date: 27 August 2015
Accepted date: 5 December 2015

Please cite this article in press as: Priyanka et al., Cloud-based automatic test data generation framework, *J. Comput. Syst. Sci.* (2016), http://dx.doi.org/10.1016/j.jcss.2015.12.001

Highlights

- Proposed the framework for effective cloud-based testing.
- Designed and developed Hadoop MapReduce based automated test data generation strategy using GA and PSO.
- Devised and implemented the new approach for the gbest evaluation using pareto-optimality.
- Empirical evaluation of the proposed framework.
- Comparison with the other existing soft-computing based cloud testing models.

Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 00 (2016) 1–33

JCSS

ELSEVIER

Cloud-based Automatic Test Data Generation Framework

Priyanka

Computer Science & Engineering Department

Thapar University

Inderveer Chana

Associate Professor

Computer Science & Engineering Department

Thapar University

Ajay Rana

Director

Amity University, Noida

Abstract

Designing test cases is one of the most crucial activities in software testing process. Manual test case design might result in inadequate testing outputs due to lack of expertise and/or skill requirements. This article delivers automatic test data generation framework by effectively utilizing soft computing technique with Apache Hadoop MapReduce as the parallelization framework. We have evaluated and analyzed statistically our proposed framework using real world open source libraries. The experimental results conducted on hadoop cluster with ten nodes are effective and our framework significantly outperforms other existing cloud-based testing models.

Keywords: Software Testing, Cloud Computing, MapReduce, Soft Computing, Particle Swarm Optimization, Genetic Algorithm, Pareto-optimal, Cloud-based Testing

1. Introduction

Software testing is the most imperative and rational facet to substantiate that developed software meets approved quality principles. Across industry sectors software testing process costs approximately 30% of the total cost of software development. Appropriate and accurate testing determines the precision at which the developed software would be able to support the business need. Both the elements (cost and the precision) are controlled through robust and effective test case design methodology. Manual test case design is time consuming and requires detailed expertise and experience. Nevertheless any manual process is prone to the risk to human errors. There have been instances wherein testers were not been able to design the appropriate test cases which caused ineffective software testing and

Email addresses: priyankamatrix@gmail.com (Priyanka), inderveer@thapar.edu (Inderveer Chana), ajay_rana@amity.edu (Ajay Rana)

often lead to increase in the cost of software when bugs were found at a later stage. Generating a test input to reach a specific location within the source code is known as undecidable and can be perceived as the halting problem of a Turing machine[7]. Soft computing techniques such as genetic algorithms and particle swarm optimization algorithm are found to be more efficient in solving the NP-complete problems. The prodigy of genetic and particle swarm optimization algorithm constitutes of a set of chromosomes or particles that forms the population. The solution to the problem is found by directing the search towards potential areas of the search space by comparing and contrasting the fitness function values with the overall search goal. Software test data generation is equivalent to combinatorial problems that can be solved by utilizing soft computing techniques at an acceptable computational cost. This is carried out by converting the test criteria to the fitness function. The fitness function decides the suitability of the prospective solutions and the exploration is headed towards the best possible optimal solution space. This is also termed as search-based software test data generation. Famous researchers have worked in this direction and contributed to the academic research in many ways [1][2][3][4][5][6].

The quality of the generated test data can further be improved if it satisfies multiple objective functions. That can be achieved by utilizing the concept of pareto-optimality. It helps in identifying the solutions that satisfy multiple objectives. This process of fitness function evaluations and comparisons consumes considerable amount of computational power and time. In addition to this, the population of large number of particles is preferred as it supports better exploration of the search space and helps in obtaining optimal and accurate solutions [8]. However, the exploration demands enormous amount of computational power, time and storage space. This requirement can be fulfilled by using cloud computing. Cloud computing offers dynamically scalable virtual resources by making use of commodity hardware as a service over the Internet. Although significant efforts have already been carried out to make soft computing techniques (evolutionary algorithms and swarm intelligence) work in parallel by employing the distributed architectures (like OpenMPI and grid computing). However, these solutions are not sufficient enough to address all the problems like heterogeneity, scalability, load balancing, communication, frequent failures, synchronization between distributed components etc. On the other hand, the architecture of MapReduce demonstrates the superior characteristics of distributed computing to the users and facilitates easy development of parallel applications in distributed environment [13]. It handles all the tedious tasks such as fault tolerance, load balancing etc. and the coordination of the execution of parallel tasks in the distributed environment. It works by dividing the bigger problems into smaller tasks and executing them in parallel on several machines at the same time. Users specify the computation in terms of a map and reduce function. It makes the application cloud ready and cloud offers dynamically scalable virtual resources as a service over the internet on rental basis at economical rates. It supports the distribution of an application over cloud in a cost-effective and convenient manner.

Apache Hadoop MapReduce is open source software and capable of distributing the task of test cases generation to several cheaper nodes available on the cloud easily and very effectively. It also helped us in obtaining test data set selection in a faster and cost effective way. Based on these above key salient features, Apache Hadoop MapReduce has been selected for the development of our proposed framework.

In this paper, we present a framework based on Apache Hadoop MapReduce for automatic test data generation. We have implemented our framework on java platform that takes bytecode of SUT, type of testing and goal of testing as an input from the user. The next step is to convert the test description file and store on the Hadoop Distributed File System (HDFS) in the Hadoop cluster. The underlying strategy for test data generation of the proposed framework is called by Map-only job. This is designed by amalgamation of genetic and particle swarm optimization algorithm. Rather than applying the velocity and distance update rules to the particles for updating their values, we applied genetic operators for update and evolution. It was done because genetic operators are more explorative and it is comparatively much easier to apply these operations on the object oriented structure. The particle representation contains not only parameter values but also the constructor and method invocations. This also helped us to achieve the increased population diversity of genetic algorithm and fast convergence rate of particle swarm optimization by selecting the best individuals as the new candidate solution in the upcoming generation. The Gbest particle set is formed from the pareto fronts and PBest are the particles that possess better pareto ranks. Pareto optimal test suite thus obtained is further minimized and prioritized by the mapper and finally JUnit file is generated. The cost of generated test cases is evaluated by Cost Evaluator based on four parameters namely test goal, cost budget, size of the software under test and type of testing. The features of our proposed framework are as follows:

i. Able to generate test data for more than one object oriented class in SUT efficiently.

ii. Pareto optimal solutions that satisfy multiple objective functions (e.g. fault detection, coverage and test case execution time) are formed by using the concept of pareto front and pareto ranks.
iii. Explorative and exploitative heuristic using the properties of both genetic and particle swarm optimization algorithm.
iv. Easier parallel implementation through Apache Hadoop MapReduce.

1.1. Motivation of our work

The motivation of our work stimulated from the need of intelligent and fast automatic test data generation strategy that reveals faults present in the software at earlier stage in minimum time during unit testing. In spite of the fact that soft computing techniques can solve test data generation problem automatically but still software industries did not opt for this. This may be due to complex and iterative nature of the existing strategies which requires huge cost in terms of computational resources and time. Although significant efforts have been carried out to make soft computing techniques (evolutionary algorithms and swarm intelligence) work in parallel by employing the distributed architectures (like OpenMPI and grid computing), yet these solutions are not sufficient enough to address problems like heterogeneity, scalability, load balancing, communication, frequent failures, synchronization between distributed components. Apache Hadoop Map Reduce framework, in contrary, provides the programmers a cutting edge to handle all the tedious tasks like fault tolerance, load balancing and the coordination of the execution of parallel tasks in the distributed environment. It is lightweight programming technique that is executed faster in parallel on lot of machines which makes the application cloud ready and offers dynamically scalable and virtual resources as a service over the internet on a rental basis. Several cheaper nodes available in the cloud provide massive computational resources required for test data generation in cost-effective manner.// Another benefit of MapReduce framework that makes it very useful is that it works in tandem with Hadoop Distributed File System. This means that data storage and computation can coexist on the same physical nodes within the same cluster. It supports the idea of moving computations to data i.e. rather than moving data to computing location, the computing is executed at the location of the data itself. This is exactly opposite to how it is done in the case of classical distributed processing systems where-in the application and the data both need to be moved to the available computation resources. MapReduce takes advantage of data proximity and is capable of processing large amounts of data without being affected by traditional bottlenecks like network bandwidth.

1.2. Technical challenges

Robust and fault free software is the ultimate target of any software development company. To achieve this, software companies invest lot of money in software testing but despite of these efforts and investments, software failures are identified at later stage and sometimes even by the end customers even after deploying software at client site. This leads to a massive added expenditures and complications while correcting the faults that exists in the software. Hence it is very necessary for the software development companies to identify and rectify faults in the software at an early stage to reduce the overall software development cost. Test data is used to verify and validate the software and it is a very important aspect which often decides the success of software testing. Therefore, test data need to be chosen very carefully to ensure that it not only cover the paths specified by test goal given as an input by user (or coverage of all the public classes and methods by default) but also reveal faults within the code. Automated test design using soft computing technique is a very good solution to deal with this problem but software companies might face the challenges while adopting soft-computing technology. Few of the key challenges are listed as below:

- Computational Cost: Despite the efforts of various researchers in the field of test data generation based on soft computing still the application of this process within software industry is limited [43][44][45][46][47][48][49]. which may be due to the fact of associated added computational cost with these techniques.

- Test size: Population based test data generation strategy often results in massive amount of test data being generated due to large number of iterations and population size. This generated intermediate test data need to be saved onto a disks before selecting optimal test suites.

1.3. Our approach

The primary contribution of this paper lies in the development of cloud-based automatic test data generation framework and the underlying test case generation heuristic. In our previous work [41], we have designed and proved experimentally that the performance of our hybrid strategy based on genetic and particle swarm optimization algorithm for automatic test data generation is better than the other traditional strategies like GA, PSO, Hill Climbing, Simulated Annealing and Memetic algorithms. Therefore, we have designed and implemented a strategy based on hybrid particle swarm optimization and genetic algorithm in which 'gbest' and 'pbest' particles are computed using the pareto-optimality approach. Pareto optimal approach helps in finding the optimal solution that possesses best fitness values for multiple objectives. Mutation and Crossover is further applied to these selected particles. Only best solutions are retained and carried over to the next generation. The two component fitness function has been designed which computes the branch coverage as well as the number of faults lying in the traversed paths. The optimal test data that reveals more faults in minimum time is given the priority than the others.

Our framework is generic enough to use 'test oracle' from the available code contracts. Contracts comprise of preconditions and post conditions, are scientifically labeled for each method of the class [50]. Fault is said to be occurred when an exception is generated without any violations of the precondition of the contract. However, in this paper all the unchecked exceptions reported on artificial seeding of the bugs in the SUT have been termed as faults.

To deal with huge data sizes and computational costs, we can utilize the resources provided by the cloud computing. Thus, we have developed our framework utilizing the Hadoop MapReduce which helps in making our framework compatible with the cloud environment. Hadoop MapReduce framework distributes the task of test data generation to several mappers and enables concurrent execution on different nodes. Due to parallelization, we are also able to reduce the time required to generate the optimal test suites. Furthermore, the implementation of this framework over cloud can facilitate the software tester to run massive test jobs with no upfront cost. The framework can thus promote agile testing in multiple environments without the need of setting up of test infrastructure. It also leads to cost saving and augments customer satisfaction by detecting bugs earlier in the development phase.

In this paper, our main focus is to look deeper and devise an efficient soft computing based test case generation strategy that can be easily adopted by software industries. The main contributions of this research paper are mentioned below:

i. Proposed the framework for effective cloud-based testing.
ii. Designed and developed Hadoop MapReduce based automated test data generation strategy using GA and PSO.
iii. Devised and implemented the new approach for the gbest evaluation using pareto-optimality.
iv. Empirical evaluation of the proposed framework.
v. Comparison with the other existing soft-computing based cloud testing models.

1.4. Paper Organization

The paper is structured as follows: Section 2 introduces the basic concepts of Apache Hadoop MapReduce, Pareto-optimality, genetic and particle swarm optimization algorithms, section 3 gives the related work, describes the proposed framework in detail, section 4 describes the proposed framework in detail, section 5 throws light on performance evaluation and section 6 summarizes the paper and gives suggestions for future work.

2. Background

In this section, we have introduced the fundamentals of genetic algorithm, particle swarm optimization algorithm and Apache Hadoop MapReduce.

2.1. Fundamentals of GA

Genetic Algorithms (GAs) is an evolved metaheuristic optimization technique that aims to find set of promising solutions. This is done by encoding the candidate solution into some data structures called as chromosomes. This technique copies the principle of Darwian theory of biological evolution, as it applies recombination and mutation operators on these chromosomes yielding one or many candidate solutions. GA manages and maintains the set of candidate solution throughout the solution process. Initially, the set of chromosome is randomly generated and in next

successive steps, a selection operator is employed, choosing two best solutions from the current set. The selection is directed by the results given by the fitness function that is used as quantifier. It is then followed by the application of crossover operator on two selected solutions. The applied crossover operator introspect through exchanging section between the two selected solutions on pretexts of a defined crossover probability. Out of them, the fittest solution is chosen by mutation operator and value at each position in solution is altered as per standard given by mutation probability. Then the algorithm is terminated when a defined stopping criterion has met. A Basic algorithm for GA is described in Algorithm 1 [15]

ALGORITHM 1: Genetic Algorithm

input : Random Population and appropriate Fitness function
output: Optimized Solution

Initialize(population);
Evaluate(population);
while not*satisfied* **do**
    Selection (population);
    Crossover (population);
    Mutate (population);
end

2.2. Basic PSO

Particle Swarm Optimization utilizes swarm-based meta-heuristic search technique given by Eberhart and Kennedy in 1995 [16]. It mimics social norms of bird flocking and animals herding. Here like swarms, searching for food in a unified way, PSO is also a set of random potential problem solutions, called particles. PSO strategy is based on the movement and exploration of virtual input search space by a particle. Particle can be observed as an object which consists of position and velocity as two components. As particle moves, the position and velocity of a particle is continually updated with every repetition of an algorithm. Every particle keeps the record of particle's personal best position in the swarm and the global best position. In the initialization phase, each particle possesses random position and velocity within the domain of the specific function. In the process, applying the evaluating function, particle modifies its velocity, so as be place toward its best personal point and global best point with respect to the input space. The following equations are used in PSO to update the position X and velocity V of a particle. A Basic algorithm for PSO is described in Algorithm 2. In the algorithm rp and rg are the two random values in the range [0,1].

$$\begin{aligned} V_{j,d}(t) = wV_{j,d}(t-1) + cr_{j,d}(pBest_{j,d}(t-1) - X_{j,d}(t-1)) \\ +c'r'_{j,d}(lBest_{j,d}(t-1) - X_{j,d}(t-1)) \end{aligned} \tag{1}$$

$$X_{j,d}(t) = X_{j,d}(t-1) + V_{j,d}(t) \tag{2}$$

where $X(t)$ and $V(t)$ represents the position and the velocity of particle at time t respectively. The inertia factor is specified by w; r and r' are the constants. A Basic algorithm for PSO is shown in Algorithm 2 [17].

2.3. Pareto-optimality Ranking Approach

Pareto optimality is a theory in economics which has wide range of utilization in engineering, game theory and social sciences. The basic concept of pareto-optimality is defined as an occurrence of allotment of resources in which it is impossible to make any one individual better off without making at least one individual worse off. Pareto optimal solution is the solution that is not dominated by another solution in the search space. It cannot be enhanced with reference to any objective without deteriorating at least one another objective. The set of corresponding objective functions in the objective space are known as Pareto Front. A decision vector x is said to dominate a decision vector y (also written $x \succ y$) if and only if their objective vectors $g_i(x)$ and $g_i(y)$ satisfies:

$$\begin{aligned} g_i(x) \geq g_i(y)\, \forall i \in \{1, 2, \ldots, N\}; \textit{and} \\ \exists i \in \{1, 2, \ldots, N\} \,|\, g_i(x) > g_i(y) \end{aligned}$$

ALGORITHM 2: Particle Swarm Optimization

```
input  : Random Population and appropriate Fitness function
output: Optimized Solution
foreach Particle i do
    initPosition(i);
    initBestLocal(i);
    if i=1 then
        initBestGlobal();
    end
    if improvedGlobal(ı)==1 then
        updateBestGlobal(i);
        initVelocity(i);
    end
end
while notendingCondition do
    foreach Particle i do
        createRnd(rp,rg);
        updateVelocity(i,rp,rg);
        updatePosition(i);
        if improvedLocal(ı)==1 then
            updateBestLocal(i);
        end
        if improvedGlobal(ı)==1 then
            updateBestGlobal(i);
        end
    end
end
```

Mathematically, Multi-objective Optimization Problem can be defined as follows:

Given: A vector of decision variables, x, and a set of objective functions, $g_i(x)$ where $i = 1, 2, \ldots, N$
Definition: Maximize $\{g_1(x), g_2(x), \ldots, g_N(x)\}$ by finding the Pareto optimal set over the feasible set of solutions.

Pareto frontier identification process in engineering helps in making knowledgeable decisions when there is trade-off between multiple objectives. Selection of pareto efficient subset of the test suite is the main task of multi-objective test case selection problem and has been formulated by Shin Yoo et al.[37]. It is defined as follows:

Multi Objective Test Case Selection
Given: A test suite, T, a vector of N objective functions, g_i , $i = 1, 2, \ldots, N$.
Problem: To find a subset of T, T', such that T is a Pareto optimal set with respect to the objective functions, g_i, $i = 1, 2, \ldots, N$.

The objective functions are the mathematical descriptions of test criteria concerned. A subset t_1 is said to dominate t_2 when the decision vector for t_1 ($\{g_1(t_1), ..., g_N(t_1)\}$) dominates that of t_2.

2.4. Introduction to MapReduce

MapReduce is the child product given by Google researchers Dean and Ghemawat [13]. It was developed by Google for the economical processing of data-intensive problems by exploiting huge number of commodity hardware lying in the data center in an easy way [14]. It is an extremely scalable and parallel processing framework that facilitates the creation of parallel programs easily for distributed cluster of processors or stand-alone computers without worrying about intra-cluster communications, failure handling and task monitoring. MapReduce is combination of

two distinct functions Map and Reduce which implements divide and conquer strategy. The Map function handles parallelization by splitting large data set into autonomous chunks and systematizes them into key and value pairs whereas Reduce collates the results. This process augments the speed of the cluster by providing rapid solutions in a reliable manner. The InputFormat arranges the divided input into ranges and map function creates a map task for each range. The distribution of map tasks among worker nodes (slave nodes) is performed by the JobTracker (or Master node). The Reducer processes and collates the set of intermediate key value pairs with the same key values generated by Mappers and subsequently writes its output back into HDFS.

MapReduce Framework has been implemented in several unique ways. Among them AppEngine MapReduce [9] and Hadoop MapReduce [12] are the most popular implementations. Apache Software Foundation has launched the open source implementation of MapReduce which aids programmer simultaneous execution of enormous chunk of data in parallel over large cluster of machines.

Apache Hadoop MapReduce is better than Google AppEngine as it supports high fault tolerance and high data availability. It also comes with lower hardware costs and minimum data transfer latency.

3. Related Work

The dissemination of software testing on numerous computing machines in parallel has several advantages and has been illustrated by Starkloff [56]. Unfortunately, very few researchers have paid attention on this area. Lastovetsky and Alexey [57] demonstrated the speed up of 7.7 on two 4-processor workstations of regression test suite over the serial execution of test suite that took 90 hours for each bug detected[59][60][61]. Joshua employed Jini framework to carry out the distribution of the execution of regression JUnit test suite over multiple nodes, whereas GridUnit used the computational Grid for distributing the task of JUnit test suite execution over multiple machines on Grid. Yao et al.[62] simulated grid-based unit testing for NUnit and dbUnit with a support of C# and database-driven projects. However, the above approaches does not address automated test data generation and requires manual creation of test suites beforehand for their execution on either LAN or a Grid. The shortcoming of using LAN is that it has limited number of nodes and Grid has computing machines of varying capacities. This requires a rigorous effort by the administrators to efficiently balance the load across all machines to get the best output. In this case the programmers have taken explicit care of heterogeneity, scalability, load balancing, communication, frequent failures and synchronization between distributed components.

Since, manual test data generation is prone to errors, we have proposed cloud-based automated test data generation framework for efficient test data design and effective utilization of parallel machines. The usage of Apache Hadoop Map Reduce has equipped our strategy with the benefit to overcome the problems associated with LAN or Grid as explained above.

This section is segregated into two subsections. The first subsection describe the work done by other researchers on parallel genetic and particle swarm optimization algorithms using Map Reduce. The second subsection illustrates several studies proposing different models for the cloud-based testing.

3.1. Parallel Genetic and PSO Algorithms based on MapReduce

In this section, we will describe the work done in the field of parallelization of GA and PSO algorithms using MapReduce model. However, the perspective of the aforementioned research is not software testing and is different from our proposed framework in a number of ways. In [55] the author proposed MRPGA in which he emphasized the iterative nature of GA process and added additional reduce phase for selecting best individuals at the end of iteration of parallel GA. It is composed of single master and several mappers with reducers. Execution of tasks in parallel on multiple mappers is scheduled by Master. Mapper evaluates the fitness function values of chromosome and the reducer chooses the optimum chromosomes. It has been implemented in .NET platform using C sharp language and problems like DLTZ5 and DLTZ4 has been solved [51]. Verma et al.[54] modelled genetic algorithms into the Hadoop MapReduce model by using multiple reducers. Their experiments demonstrated the convergence and scalability up to 105 variables for the ONEMAX problem. Ibrahim Aljarah et al. [52] proposed a framework that produces parallel version of particle swarm optimization using MapReduce. The authors devised clustering algorithm analogous to the k-means clustering method and demonstrated efficient usage of commodity hardware. Scalability and linear speed up was proved experimentally. McNabb et al. [53] has also worked for the parallelization of Particle Swarm Optimization

using MapReduce model. The author evaluated his model with benchmark function and shown a scaling of up to 256 processors and node failure tolerance. The map phase performs update operation of particle's velocity, position, particle value and then evaluates particle's best value. The global best value is computed by the reducer phase.

Our approach is different from the above mentioned proposal as our main objective is to speed up the test data generation task by utilizing MapReduce primitives. We configured our job as Map only job where number of reduce task is set to zero. Mapper performs entire task with its InputSplit and the number of output files generated is equal to the number of mappers and is written to the Hadoop Distributed File System on the same machine. This has helped us to reduce the overhead associated with the data movements to the Reducer. All the necessary operations are performed in parallel by the Mapper.

3.2. Existing Cloud-based Testing Frameworks

Software testing is a periodic activity and its conventional model requires new environments to be set up for each project. Test labs established in companies typically sit idle for longer periods, consuming capital, power and space. The anecdotal and published reports states that approximately 50% to 70% of the technology infrastructure earmarked for testing is underutilized [63].

The model of cloud computing provides several benefits such as cost reduction, on-demand flexibility, improved alliance, increased efficiency, freedom from acquiring assets and reduction in time to-market for key business applications. The advent of cloud computing and the transformation of IT paradigm from Cap-Ex model to Op-Ex model demands the migration of software testing activities to cloud. Software testing is one of the most important activities of software development and can be safely moved to the cloud as it does not contain sensitive corporate data.

This section portrays certain related works that proposes different frameworks for the cloud-based testing. Depending on the types of research models, Cloud-based testing can be divided into seven categories [31]. In our previous work [36], we have analyzed and described the existing Cloud-based testing models and various approaches of software testing in cloud environment. We have compared our proposed framework with some of the popular existing cloud-based testing models in the Comparison Table 6.

D-Cloud is a cloud based software testing environment for testing of distributed systems [18]. It uses virtual machines using Eucalyptus private cloud environment. It facilitates fault tolerance testing by instigating device faults by using virtual machine. On comparing with our testing model, it only provisions the execution of test jobs designed by tester and does not generates test data automatically without intervention of tester.

VATS makes use of HP LoadRunner for generation of load and automatically evaluates the performance SAP R/3 system operating in Xen-based environments [19]. However, it only supplements with the Service Lifecycle Management System and only supports the testing of applications compatible with it. Ganon et all. proposed performance testing framework for Network Management System (NMS) which allows testing distributed system containing VoIP private branch exchange (PBX) networked through SIP. Authors make use of emulation agents to write application level test cases [26]. Authors demonstrated automated execution of tests in considerable less amount of time whereas our proposed strategy focuses on automatic test data generation along with execution using soft-computing technique with a remarkable time reduction capability.

Liviu Ciortea et al. [21] and Stefan Bucur et al. [20] both made contributions towards the Cloud9 development. Their efforts were based on parallel symbolic testing that calibrates to large clusters of machines. Stefan Bucur et al. [20] was inspired by the research of Liviu Ciortea and he proposed a new symbolic environment model that braces all essential aspects of the POSIX interface, like synchronization, networking, processes, threads, IPC, and file I/O. Authors mainly concentrated on devising the parallel version of symbolic execution engine and also implemented load balancing for automated testing of UNIX utilities. on contrary, we developed the automated test data generation strategy using Apache Hadoop MapReduce for object oriented software. The implementation of our approach is much easier and simpler due to the fact it takes care of tasks like fault tolerance, load balancing and the coordination of the execution of parallel tasks in the distributed environment [13] using Apache Hadoop MapReduce technology, additionally our strategy also supports the test case generation of applications containing more than one class.

Alexey Lastovetsky[22] made contributions in the field of parallel testing of computer software systems. In his work author demonstrated regression testing for a distributed programming system in a parallel environment and observed a speed up of 7.7 on two quad processor workstations. Joshua[22] and GridUnit [27][28][29] executes JUnit test cases in parallel. Both the tools are based master slave architecture where master node instructs the numerous slave nodes to execute test cases in parallel which fastens the testing process. Joshua harnessed Jini for parallel

distribution of regression test suite over several nodes, whereas GridUnit employed grid computing. In [30] author designed HadoopUnit which performs unit testing using JUnit for distributed execution framework. It made the use of MapReduce primitives to distribute test cases execution over the cloud. The necessary files such as testing tool and test cases are uploaded to the Hadoop Distributed File System prior to its utilization by a Mapper node. Mapper nodes are instructed by a Master for execution of test cases and the Reducer collects the report and stores it in the distributed file system. An elementary case study [30] done with HadoopUnit on a 150-node cluster showed astonishing speedup of 30x in execution time. These research works basically focused on execution of manually generated test data over multiple nodes and substantially lack the user support in term of automatic designing of intelligent test cases.

Sasa Misailovit'c et al. [23] worked on novel approach based on the Korat algorithm for complex test inputs. It requires input by the user in the form of imperative predicates and finitization that limits the size of test inputs. Imperative predicate and finitization are java methods and needed the involvement of tester having expertise in java programming language. On the other hand, in our framework there is no such requirement from the user. The user only needs to upload the bytecode of the software and our framework automatically generates test cases for all the testable methods and classes in an object oriented program.

Manuel Oriol [24] designed YETI which is random testing tool over cloud and test programs written in Java and .Net. The authors have used the Apache Hadoop MapReduce for distributing the tasks of generating test cases over Amazons Elastic Compute Cloud (EC2). Yeti however, face challenges in terms of automatic mapping of testing sessions for more than one class, defining test adequacy criteria of test cases for better detection of faults in random testing sessions distributed over nodes. Our proposed framework is capable of generating optimal test cases very easily and efficiently with branch coverage and fault detection as test adequacy criteria. It is also capable of automatic distribution of all the classes present in software over several nodes in a hadoop cluster for parallel generation of automatic test cases.

Yu et al. proposed novel model TaaS for the provision of testing resources access to end users. Authors utilized computing resources efficiently by devising their scheduling and dispatching algorithms. They also examined the tolerance of the framework by raising load of test task and have also analyzed the total computing time by dispersing it into two components i.e. test task scheduling and test task processing time [25]. On the other hand, we devised automated test data generation and employed Hadoop Map Reduce to enhance features like load-balancing, fault-tolerance and scheduling of MapReduce jobs over hadoop cluster. Linda Di et al. [35] employed traditional genetic algorithm using Hadoop Map Reduce to generate test data. On the other hand, we adopted the hybrid strategy employing the concept of particle swarm optimization, genetic algorithm and pareto optimality principle to identify best optimal test suites in minimum possible time.

Although the research carried out at Fujitsu [39] suggested that testing is the most likely workload that can utilize cloud computing resources, yet very few works from academia has been reported on cloud based software testing strategies. Most of the research work are around the parallel test execution. Only few authors [35][24] have worked on automatic test case generation strategy. QoS factor considered by them is either branch coverage or bugs detected. Our research work is centered on optimal and intelligent test case generation with complete quality check of an application submitted for unit testing. This has been achieved by giving emphasis on QoS parameters such as %branch coverage and average percentage of faults detected per unit time. A novel software testing framework for automatic test data generation has been designed and evaluation is performed by comparing it empirically with the other existing soft computing based test data generation strategies in cloud environment.

4. Architecture Overview of Proposed Framework

In this section, we have described the architectural overview of our proposed framework and is illustrated in Figure 1. As depicted, our framework is composed of the following components:

i. **Starter**: It is the front-end of our framework which takes an input SUT, TestGoal from the user and transforms it in the form of Test Description File(TDF) and stores in a file in HDFS. It invokes the MR-PSOG in Optimizer.
ii. **Optimizer**: It is the main module which performs necessary computations for automatic test data generation. It gives the JUnit file and also stores information related to number and type of resources utilized in the generation of test suites.

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

Table 1: Comparison Chart of Existing Cloud-based Testing Models

iii. **Cost Evaluator**: It generates the bill based upon the factors namely test goal, cost budget, size of the software under test and testing type.

The Overall flow of the Framework is described through the following steps and is depicted in Figure 2:

i. The user submits the test request to the Starter. The test request consists of bytecode of the SUT, type of testing and the test goal in the form of Test Description File (TDF). TDF is saved on the Hadoop Distributed File System (HDFS) in the form of different files.

ii. The Starter signals the Optimizer about the user request and request it to read the TDF stored on the HDFS and other related files related the code of the SUT. The Optimizer then instruments the bytecode of the SUT and starts MapReduce job. This node containing Optimizer Module acts as the master node of the Hadoop Cluster.

iii. The MR_PSOG algorithm (main component of Optimizer) forks many copies of itself on the Hadoop cluster. Subsequently, the master node allocates map tasks to the slave nodes of the Hadoop cluster. The Map tasks consist of all the operations related to the execution of intelligent test data generation. It also performs the minimization and selects the most optimal test suites generated over mapper-slaves and writes the Final TestSuite to the HDFS. The number of nodes used to perform map tasks are also written in the HDFS.

iv. The workers notifies about its completion to the master node. After that, the Optimizer module sends the signal to the Cost Evaluator to generate the bill. It reads the data stored in the files related to number of resources used and other input factors such as type of testing, test goal etc to compute the cost incurred.

v. The Cost Evaluator sends the bill to the Starter module. The Starter sends the bill to the user and asks for the payment using Payment Gateway service. The TestReport is sent to the user upon the reception of the payment.

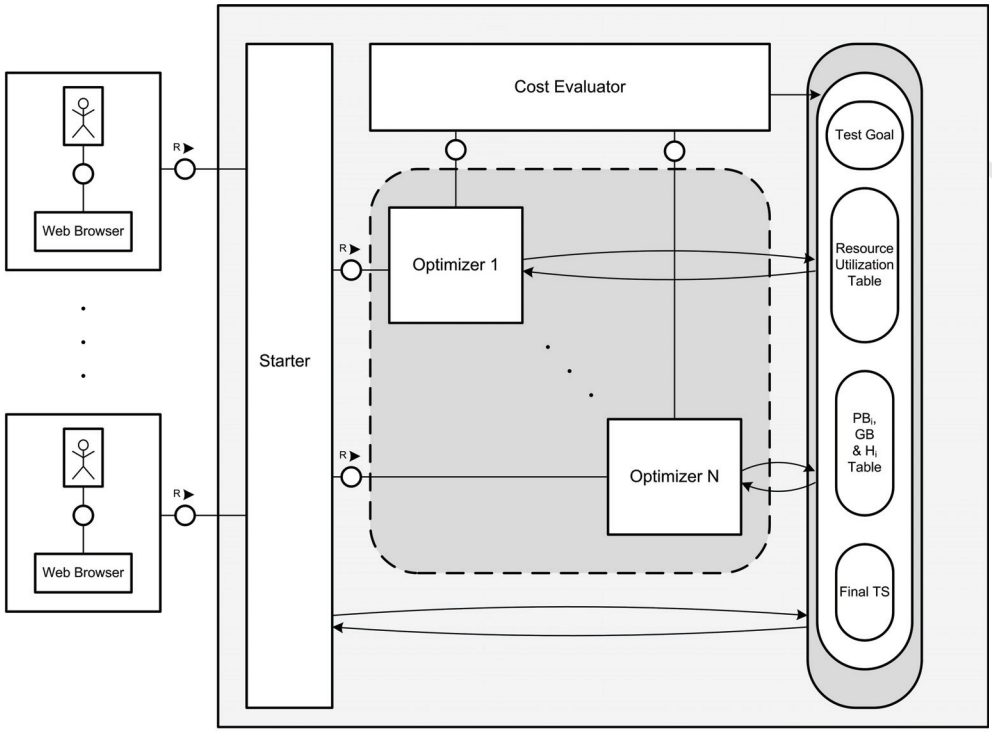

Figure 1: Framework of Cloud-based Testing

4.1. Starter

This is the front end of the framework where user inputs the purpose of testing (test goal), type of testing and bytecode of the Software Under Test (SUT). This is converted to Test Description File (TDF) and saved to the HDFS. It also consists of Report Submitter module which delivers the Test Report and Test Suites to the user after obtaining the 'payment received' signal from Cost Evaluator. Subsequently, it invokes the Optimizer for optimal test case generation.

4.2. System Architecture of Optimizer

This module is responsible for generating optimal test cases. The process of test data generation begins with the formation of class control flow graph followed by the derivation of the optimal test inputs that results into the execution of the desired paths. The suitability of the test inputs is decided by the fitness function and the desired path can be specified by the user in the form of test goal. Mathematically, test data generation can be defined as follows:

Given a SUT S **and a path z, generate input** $i \in A$**, so that** i **traverses path** z.

$A \rightarrow$ **Set of all possible Inputs**

SUT S can be represented mathematically as a function mentioned below:

S **:** $A \rightarrow R$**, where** A **is set of all possible inputs and** R **is set of all possible output.**

The control flow graph of object oriented program is shown below in the Fig. 3. The Class control flow diagram shown in the above figure shows that upon the call of a method the control flow graph of a method is created which is same as that of control flow graph of a method of procedural program.

The system architecture is shown in the Fig. 4. It consists of following two main sub-modules:

i. **MR_PSOG Algorithm**: This module manages the overall execution of MR_PSOG. It performs all the task related to initializations and once it terminates it returns test suites for Software Under Test.

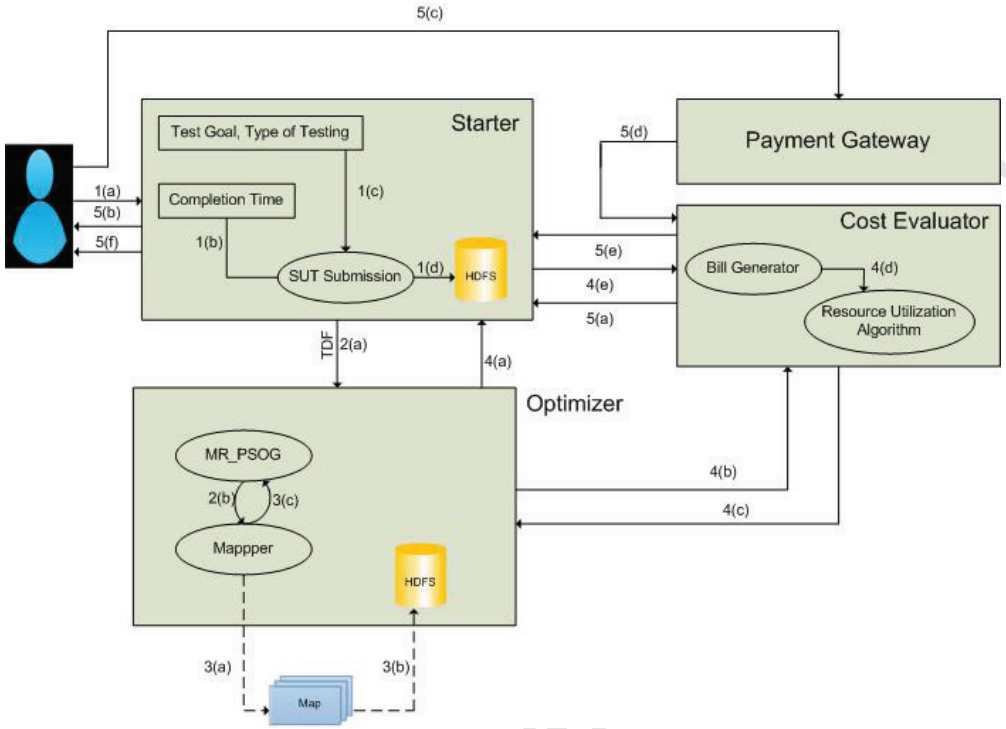

Figure 2: Execution Flow

ii. **Mapper**: It is invoked by MR_PSOG to evaluate fitness of each particle and their associated pareto ranks. Also, it keeps track of the best particle and writes it to a global file in HDFS. It is followed by the minimization of the test suites by removing redundant test cases and the prioritization of the resulting Test Suite. It also updates TestReport, FinalTS and Resource Utilization Table on HDFS.

The operative principle of the proposed approach as described above is also depicted through flowchart shown in Figure 5.

4.2.1. MR_PSOG Algorithm

It acts as the driver and performs all the tasks required to initiate the MapReduce. The implementation has been carried out in java programming language. The steps of MR_PSOG has been showed in Algorithm 3 and are explained below:

i. At the first step, new configuration is generated and the bytecode of the SUT is loaded on the HDFS. It extracts all the names of the classes present in the API in a file on HDFS.
ii. The MapReduce job is created and MRJob and Mapper class are set.
iii. The number of Reduce tasks is set to zero in the configuration. By making number of reduce tasks to zero the job becomes Map only job and Mapper does all its task with its InputSplit and no job is specified for the Reducer. In this case, the number of output files is equal to number of mappers and is written the HDFS. This helped us in reducing the overhead associated with the data movements to the Reducer. All the necessary operations has been performed in parallel with the Mapper.
iv. Output key Class and Output Value class are initialized.

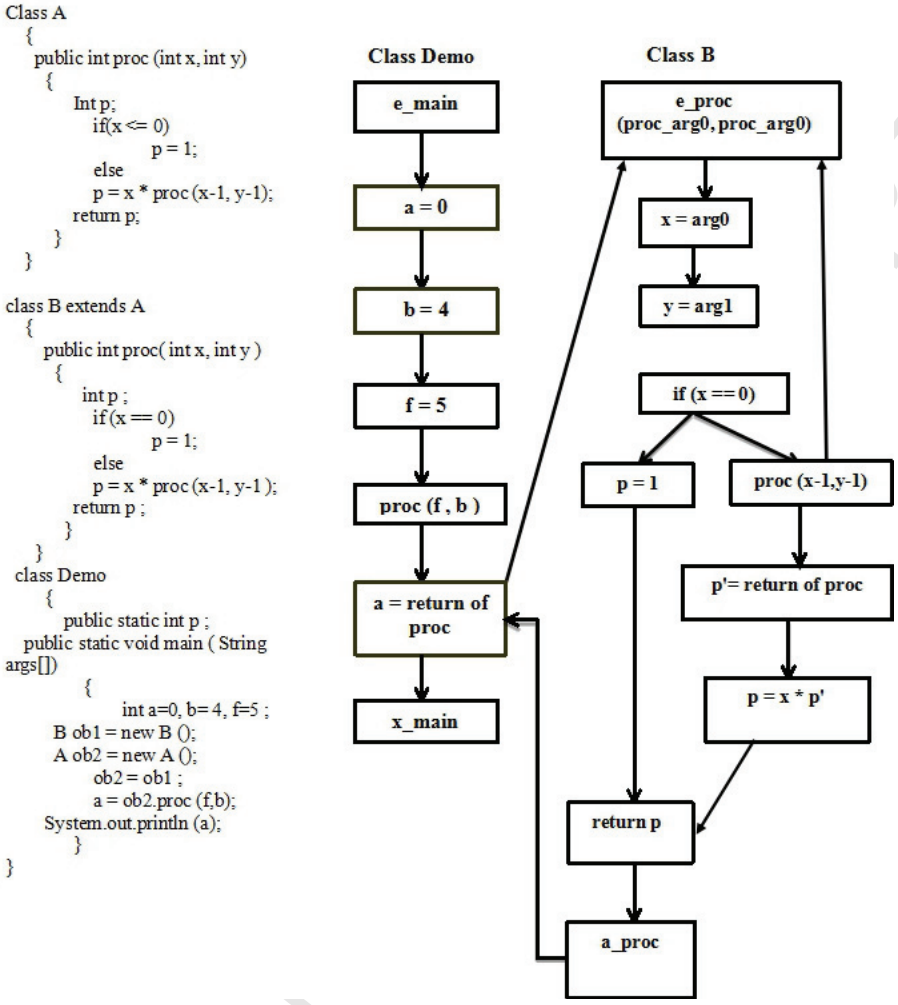

Figure 3: The Class Control Flow Diagram [41]

4.2.2. Mapper

This module is called by MR_PSOG from Master node (JobTracker) for all the classes as a Map Task over different worker machines (TaskTrackers). It makes call to the following submodules:

i. genBestPopulation: This module perform all the necessary operations of automatic test data generation for all the public classes of the SUT.
ii. minTestCases: It minimizes the test data by eliminating the redundant test cases.
iii. priortizeTestCases: It priortizes the test cases for the maximal fault detection.
iv. printTestCases: This module is responsible for printing of the test cases in the form of JUnit file
v. The JUnit files are written to the HDFS by calling context.write() method

4.2.3. The Test data Generation

Soft Computing can be defined as sequence of procedures and methodologies to find out the solution of hard problems such as NP-complete problems in the same way as done by human beings by using intelligence, common sense, consideration of appropriate approaches and analogies. Software test data generation is an undecidable problem and

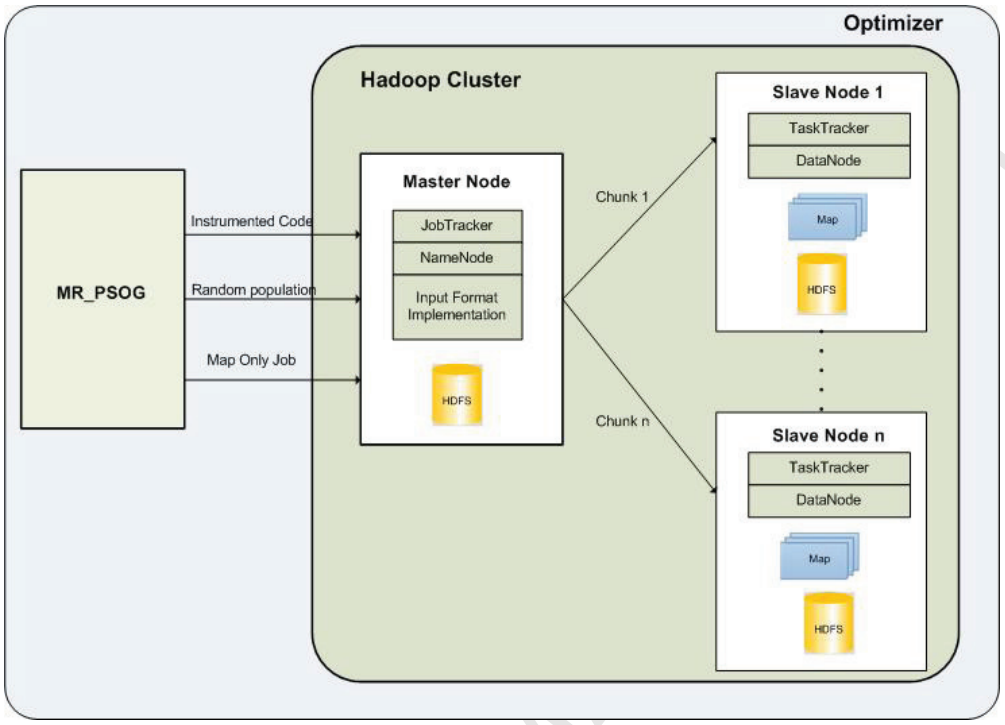

Figure 4: System Architecture of Optimizer

ALGORITHM 3: MR_PSOG

input : Byte code of the SUT
input : Path of input directory
input : Path of output directory

Create New Configuration;
Load byte code of SUT on HDFS;
Create new MRPSOG Job;
Set MRJob Class;
Set Mapper Class;
Set Number of Reduce Tasks = 0;
Set Output Key Class;
Set Output Value Class;

ALGORITHM 4: Mapper(*Key*, *Value*, *Context*)

input : Class Under Test of the SUT
output: JUnit File

classUnderTest = *value.getBytes*();
getBestPopulation(classUnderTest);
minTestCases();
prioritizeTestCases();
printTestCases();
junitFile = *getJunitFileData*(classUnderTest);
context.write(junitFile);

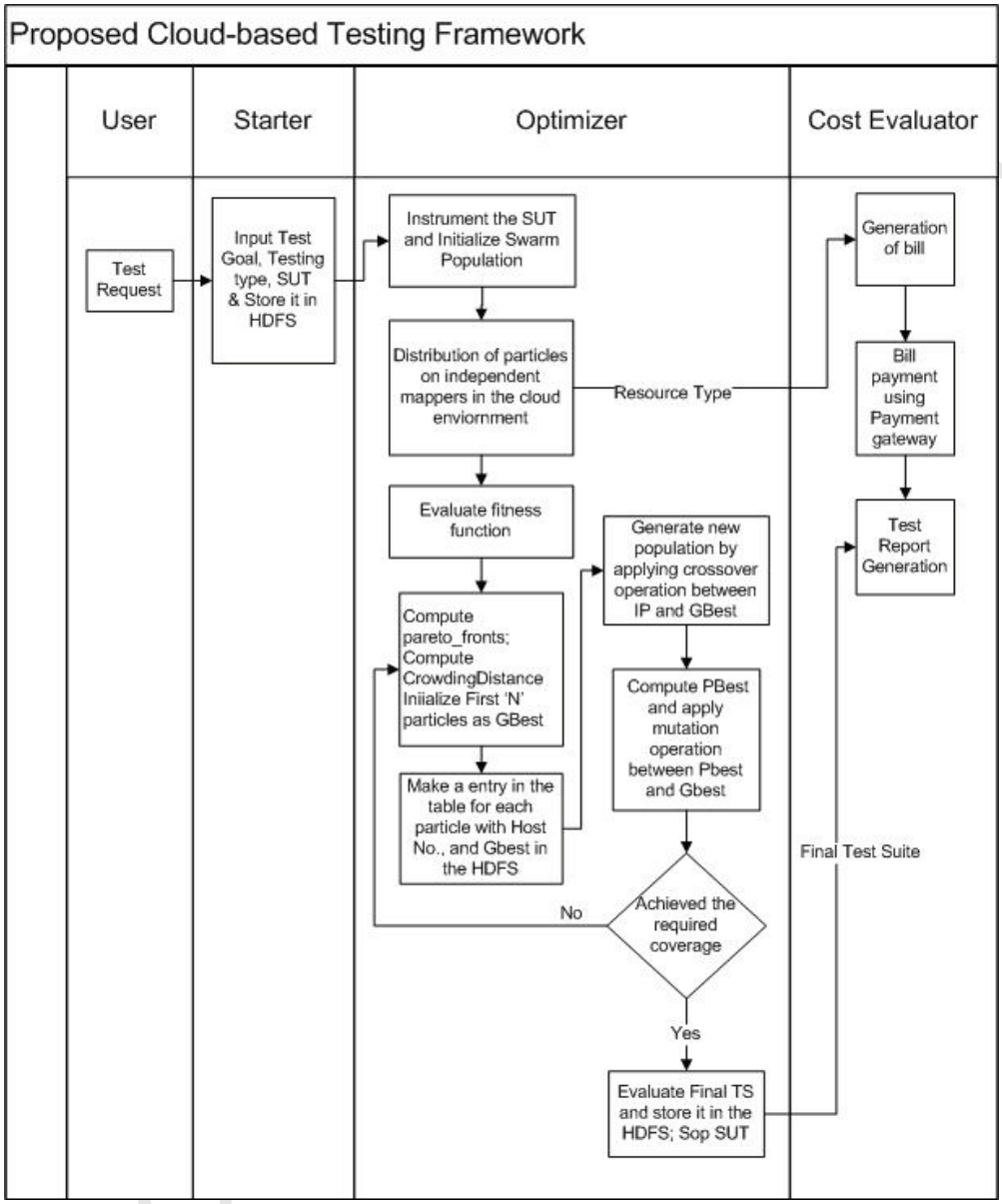

Figure 5: Flowchart of the Proposed Framework

often it is termed as NP Complete problem which can be efficiently solved by utilizing soft computing techniques such as genetic algorithms and particle swarm optimization algorithm. There are two main prerequisite of soft computing technique that should be met before it can be used to solve particular. Firstly, the method of encoding candidate solution to the problem. The second requirement is the mechanism for fitness function evaluation. Both the perquisite adopted by us in the proposed framework are explained below:

Representation

The first step to find the solution of a problem by applying soft computing technique is to encode the candidate solu-

tion. The encoding can be done with much precision with the help of illustration of the grammar. The grammar is the set of productions which defines the syntax of the candidate solution. In our research work, a solution is a test suite which encompasses set of test cases. Test case is not just a combination of parameter values rather it is a sequence of constructor and method calls with the associated parameter values. This representation has been used by many authors [32] [33] which is necessary for object oriented software. The set of rules are described below with the help of the grammar.

$$P \rightarrow BV$$
$$B \rightarrow AB$$
$$A \rightarrow c(P) \mid c \in \mid c.f(P)$$
$$P \rightarrow L, P$$
$$L \rightarrow built - in - type \mid user - defined$$
$$V \rightarrow N, V$$
$$N \rightarrow int \mid real \mid Boolean \mid String$$

Here P represents particle, B represents sequence of constructor and function instantiation represents actual parameters passed to the function, c represents class name and f represents function name.
Particle representation and its corresponding test case format is a set of JUnit test cases for a given Class Under Test (CUT) as shown in the Fig. 6

Particle Representation
ob =B() a=C(5)
ob2=T (a) ob.insert(ob2):
d=D (5) ob.search(d)

Corresponding Test Case Format

```
public void Test()
{
B ob = new B ();
C a = new C (5);
T ob2 = new T (a);
ob.insert (ob2);
D d = new D (5);
assertTrue (ob.search(d));
}
```

Figure 6: The template of Particle and Test Case Format [41]

Fitness Function

Our novel heuristic attempts to uncover test suites that have maximum branch coverage and reveals faults present in those traversed paths in minimum time. The fitness function determines the quality of a candidate solution (i.e. test suites) and is evaluated in terms of branch coverage and faults detected. The mathematical description of the fitness function adopted by us is as follows:

$$fitness = (f1, f2) \tag{3}$$

where $f1$ computes the branch coverage and assess the closeness of Test Suite to cover all branches of SUT.
$f2$ computes the number of faults detected lying in the traversed paths.

$$f_1 = |S| - |S_T| + \sum_{b_i \in B} \frac{Dmin(b_i, T)}{1 + Dmin(b_i, T)} where 0 < t \leq CT \tag{4}$$

First part → It deduces approach level of the test suite where S and S_T denotes method sequence and traversed methods respectively.
Second Part → It evaluates the normalized branch distance where B denotes set of branches of SUT and Dmin denotes minimal branch distance for each branch $b_i \in B$.
The constraint $0 < t \leq CT$ keeps execution time in check where t refers to actual execution time and CT denotes maximum completion time.

There is need to maximize branch coverage and faults detected whereas time should be minimized. The quality of branch coverage of a particle is measured in terms of approach level and branch distance which is required to be minimized and is only used to guide the search to better optimal solution. For a given path that does not cover the target branch, the approach level can be defined as number of branching nodes that are in the way between nodes covered by the individual and the target node. The approach level directs the search towards the target branch whereas branch distance guides the exploration of the search algorithm towards promising solution space. Branch distance is calculated on the instrumented code for the branch predicates by applying the recursive rules as defined in the Table 2 [2]. This has been implemented by making modifications to the source code of Jacaco.

Table 2: Branch Distance Evaluation Rules [2]

Element	Value
Boolean	If TRUE then 0 else K
a = b	If $abs(a-b) = 0$ then 0 else $abs(a-b) + K$
$a \neq b$	If $abs(a-b) \neq 0$ then 0 else K
$a < b$	If $a - b < 0$ then 0 else $(a-b) + K$
$a \leq b$	If $a - b \leq 0$ then 0 else $(a-b) + K$
$a > b$	If $b - a < 0$ then 0 else $(b-a) + K$
$a \geq b$	If $b - a \leq 0$ then 0 else $(b-a) + K$
$a \vee b$	min (cost (a), cost (b))
$a \wedge b$	Cost (a) + cost (b)
$\neg a$	Negation is moved inwards and propagated over a

$$f2 = \sum_{i=1}^{|z(x)|} q_j \tag{5}$$

Here x is input of the program, $Z(x)$ is traversed path and q_j are number of faults detected in jth node of $Z(x)$

In our work, the unchecked exceptions reported on artificial seeding of the bugs in the SUT has been termed as fault. Unchecked Exceptions can be termed as a collection of Errors and RuntimeExceptions. RunTimeException is descended from java.lang.RuntimeException and defined as the internal exceptional conditions of an application from which it cannot recover at its own. Runtime exceptions are specified by the RunTimeException class and the classes derived from it. It usually points toward the presence of programming bugs such as logical errors or inappropriate use of an API. For an instance, NullPointerException are ArrayIndexOutOfBoundException are Runtime Exceptions. On the other hand, Error is a subclass of Throwable and points towards the external exceptional state of an application from which it cannot recover at its own. Error is the exception specified by the class Error and its subclasses. Examples of Errors are OutOfMemoryError and java.io.IOError.

The test data of our proposed framework is generated by calling the method getBestPopulation(). It uses the concept of GA and PSO and performs all the necessary operations for automatic test data generation. The steps of the getBestPopulation() are explained below and also depicted in Algorithm 5:

i. At the first step, initial population is generated randomly using random test data generator for the list of targets present in each class of jar file. The target list is formed in such a way that it covers all the methods and constructors of the class files. After initialization, SUT is instrumented. The role of instrumentation is to modify the original executable image by inserting some pattern of instruction before every block to indicate that the block was executed. Though the modified code still performs the same operations as by the original program but also convey the value taken by the reached condition. The instrumentation of the SUT is essential

for the evaluation of fitness function. To evidence the branch distances, extra instruction is added before every predicate. These instructions are responsible for the computation of branch distance and notify the testing environment of it.

ii. We have used Jacaco as the instrumentor, it performs code coverage analysis using standard technology in Java VM based environment [40]. It can be integrated easily with several development tools because of being lightweight, flexible and well documented library. Jacaco can perform efficient on-the-fly instrumentation and analysis of applications even in the absence of source code.

iii. The algorithm is iterated until optimal solution has been found or maxIterations has been reached. maxIterations denotes maximum allowable test data generation time and can be set by the tester.

iv. MR_PSOG invokes several mappers for all the classes in the SUT. It computes particle fitness function values, pareto fronts and perform genetic operations in parallel for all the classes.

v. Initially, it evaluates the objective function of the randomly generated population. The fitness function values are evaluated as per the defined fitness function explained above.

vi. Pareto fronts are computed using non-dominated sorting algorithm. Non-dominated sorting algorithm helps to resolve the conflicts among different objective functions. We have used three objective functions (branch coverage, faults detected and execution time) for evaluation of the quality of particles. The algorithm of genparetoFront is shown in Algorithm 6.

vii. The computed pareto fronts are used to determine the ranks of the particle. The rank of each particle is assigned with the pareto front number. If p_i belongs to PF_i then rank of the particle is set to i.

viii. gencrowdDistance algorithm is used to introduce the diversity amongst non-dominated solutions. Crowding distance function facilitates the selection of particles that are placed in less dense areas and hence helps to avoid local stuck problems and adds more explorative and exploitative solutions.

ix. The particles are sorted in such a way that the particles possessing lower rank are given priority over the others. If the ranks of particles are same then the particle which has greater crowding distance is given priority over others. It is shown mathematically in equation No. 3 and the algorithm is described in Algorithm 7 [34].

$$p_i \geq p_j = \begin{cases} if\ rank(p_i) < rank(p_j)\ or \\ (rank(p_i) = rank(p_j))\ and \\ (distance(p_i) > distance(p_j)) \end{cases} \tag{6}$$

x. The sorted pareto fronts thus obtained are stored in the GBest

xi. The new population is evolved by applying crossover operations between GBest elements.

xii. It then figure out PBest particles by comparing the pareto ranks of the current iteration with the rank of particle in the previous iteration. Lower the pareto rank better is the individual for the selection. The particle possessing better rank values are retained whereas others are rejected. PBest particle is the state of the individual who has acquired the best fitness value when compared it with its previous value in the earlier generation.

xiii. The mutation operator is applied over the new evolved particles (represented by O1 and O2 in the Algorithm 5) and PBest particles. The particles are modified only when the fitness value of mutated particle is higher otherwise earlier particles are retained.

xiv. The above step is augmented with crossover operation between newly formed PBest particles and GBest particles. In this way, the best solutions are evolved and saved in the variable IP for the subsequent iteration

xv. The instrumented code is executed to find for unchecked targets.

xvi. The intermediate results like PBest, Covered Targets are saved to a file in HDFS for further reference.

xvii. The above operations are repeated until we get optimal solution or exhausted the threshold of iterations.

4.2.4. Minimization and Prioritization

The module minTestCases() performs minimization of the test cases and eliminate the redundant test cases. It also updates the TestReport, FinalTS, Resource Utilization Table on HDFS. The priortizeTestCases() module prioritize the test cases such that fault detection rate increases. To achieve this, we have utilized the concept of Test Prioritization defined by Rothermel [11] for the prioritization of the test suite. The prioritized test suite covers all the branches and

ALGORITHM 5: getBestPopulation(*classUnderTest*)

input : Class Under Test
output: Fitness value of each particle in the population along with their pareto ranks

IP← *Generate random population of N particles*;

Instrument the SUT;
while $t \leq maxIterations$ **and** *optimal solution* **not** *found* **do**
  for $p_i \in IP$ **do**
    Evaluate fitness of each particle p_i *as per equation (4)*;
  end
  genParetoFront(IP);
  if $p_i \in PF_i$ **then**
    $r_1(p_i, t) = i$
  end
  genCrowdDistance(PF_i);
  $Q = Q \cup PF_i$;
  Sort the elements of Q as per relation (7);

  $GBest = Q[0 : N]$

  $Z \leftarrow$ elite of $GBest$;
  while $|Z| \neq |GBest|$ **do**
    if CP **then**
      $O_1, O_2 \leftarrow Crossover\ GB_1, GB_2$
    else
      $O_1, O_2 \leftarrow GB_1, GB_2$
    end
    if $(r_1(p_i,t) < r_1(p_i,t-1))$ **then**
      $PBest = (p_i, t)$
    else
      $PBest = (p_i, t-1)$
    end
    Apply mutation over PBest;
    If its fitness value greater than earlier PBest set it as new PBest;
    If all the individuals have been processed then write best individuals in PBest and GBest files on HDFS;
  end
  Apply Crossover operation on each particle stored in PBest with GBest;
  $IP \leftarrow$ elite of Z ;
  Execute the instrumented code for each particle to check if there any other untested branches being tested;
  Update CovTable, TesGoal, TestSuite on the HDFS;
end

ALGORITHM 6: genParetoFront(IP)

```
input  : Population IP
output: Non-dominated Fronts
for p_i ∈ IP do
    for q_i ∈ IP do
        if p_i ≻ q_i then
            T_p=T_p∪ {q_i}
        end
        if q_i ≻ p_i then
            n_p=n_p+ 1;
        end
        if n_p = 0 then
            PF_1=PF_1∪ {p_i};
        end
    end
end
c=1;
while PF_i ≠ ∅ do
    L=∅;
    for p_i ∈ PF_i do
        for q_i ∈ T_p do
            n_q=n_q− 1;
            if n_q = 0 then
                L=L∪ {q_i};
            end


            c = c + 1

            PF_i=L;
        end
    end
end
```

ALGORITHM 7: genCrowdDistance(PF_i)

```
input  : Pareto Front
output: Crowding distance
l= |PF_i|;
For each i, Set I[i]_dis=0 ;
For each objective m;
I=sort(I,m);
I[1]_dis=I[l]_dis = ∞ ;
for i = 2to (l − 1) do

    [i]_dis = I[i]_dis + (I[i + 1].m + I[i − 1].m)

end
```

detect maximum possible faults in the software under test. It is described as below:

The Test Case Prioritization Problem:
Given: T, a test suite; PT, the set of permutations of T;f is a function from PT to the real numbers.
Problem: Find $R \in PT$ such that $\{\forall(R \in PT)(R \neq T)[f(T) \geq f(R)]\}$

Here, PT represents the set of all possible prioritization of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

4.2.5. GBest Evaluation

We have utilized Pareto ranking methodology to assign the priority to the solution for selection among other solutions in the solution space. Pareto fronts are computed by making a call to method genPareto Front as shown in Algorithm 6. This concept has been used by many researchers to solve multi-objective optimization problem[34][37]. On the contrary, we have used it to form a set of promising solution (Gbest particles set) which not only satisfy coverage criteria but also detects faults present in the code in minimum possible time. The GBest variable represents the particles (i.e. test suite) which possess best fitness function value in whole population at a given iteration. The method genParetoFront() evaluates dominance relation between two particles. Dominance relation is defined as follows: A decision vector x is said to dominate a decision vector y (also written $x \succ y$) if and only if their objective vectors $g_i(x)$ and $g_i(y)$ satisfies:

$$g_i(x) \geq g_i(y)\ \forall i \in \{1, 2, \ldots, N\};\ and$$
$$\exists i \in \{1, 2, \ldots, N\} \mid g_i(x) > g_i(y)$$

The objective functions chosen by us comprises code coverage and fault detection and it should be maximized for a given time. Time should be minimized for a given traversed path. When we instantiate the optimization problem to satisfy above three objectives, it should retain the following properties:
Suppose, we achieve the population of a subset of test suite T with coverage C, fault detection D and execution time t' on the Pareto frontier, then,
T1: Any other subset of T cannot achieve more coverage than C without spending more time than t'.
T2: Any other subset of T cannot achieve more fault detection than D while achieving coverage equal to C.
T3: Any other subset of T cannot finish in less time than t' while achieving coverage equal to or greater than C.
Thus the concept of Pareto optimality helps in accomplishing the balance between the three objectives. This facilitates in finding set of optimal solutions rather than just a single solution obtained through single objective that merely approximates the global optimum solution. Each member of the set is a candidate solution upon which improvement is not possible.
Pareto fronts thus obtained are sorted as per their ranks and crowding distance as shown mathematically in Equation No. 6 [34].

$$p_i \geq p_j = \begin{cases} if\ rank(p_i) < rank(p_j)\ or \\ (rank(p_i) = rank(p_j))\ and \\ (distance(p_i) > distance(p_j)) \end{cases} \quad (7)$$

Crowding distance function shown in Algorithm 7 facilitates the selection of particles that are placed in less dense areas and hence helps to avoid local stuck problems and adds more explorative solutions. The sorted pareto fronts thus obtained are stored in the GBest.

4.2.6. Working Example

In this section, we have explained the working of the mapper algorithm by taking an example of a population of six particles. The first step evaluates the fitness values of each particle as described in the Section 3.2.6. The fitness values of each particle in the format of (%branch coverage, %fault detection, time) is as under:
P1=(60,70,3)

P2=(58,69,5)
P3=(54,65,4)
P4=(75,82,4)
P5=(70,80,3)
P6=(74,83,4)

In the second step, the algorithm genParetoFront is called and two entities (n_p an T_p) have been computed for each particle and it is shown in the Table 3

n_p : Number of particles which dominates p_i.

T_p : Set of particles which particle p_i dominates.

For each $n_p = 0$, the corresponding particles are included in the first Pareto Front (PF_1).

Therefore, $PF_1 = \{P4, P6\}$

For each element of PF_1, n_p of each element of corresponding dominated particles T_p is decreased by one and all those particles are selected whose n_p becomes 0. This constitutes the current working set and next pareto front. Hence,

$PF_2 = \{P5\}$

$PF_3 = \{P1\}$

$PF_4 = \{P2\}$

The next step is to sort the elements of pareto fronts. The sorting is carried out by applying the operator as mentioned in Equation No. 3. The crowding distance is evaluated for PF_1 by implementing the algorithm gencrowdDistance. The Figure 7 clearly shows that distance (T4) < distance (T6).

Hence, Gbest = $\{P6, P4, P5, P1, P2\}$

Particle	n_p	T_p
P1	3	P2, P3
P2	4	P3
P3	5	-
P4	0	P1,P2,P3,P5
P5	2	P1,P2,P3
P6	0	P1,P2,P3,P5

Table 3: Non-dominated Sorts

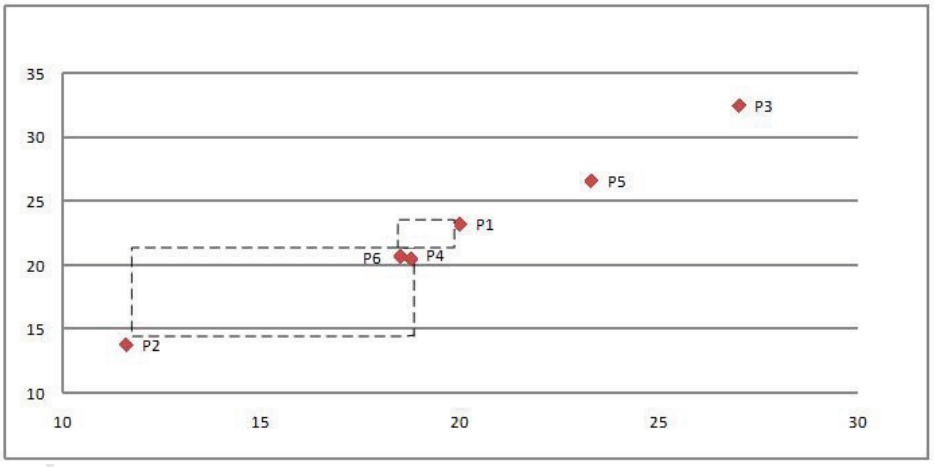

Figure 7: Crowding Distance

The next step is the application of crossover operation on the elements of GBest. Subsequently, set PBest is formed by comparing the ranks of initial value of GBest and the modified value achieved after crossover. The solution possessing lower rank is set to PBest and other solutions are rejected. PBest is further mutated to save best particle in

the set PBest. This step is followed by crossover operation between the PBest and GBest particles, resulting into the evolution of new population. The above steps are repeated until optimal solution has been found or maxIterations has been reached. maxIterations denotes maximum allowable test data generation iterations and can be set by the tester. The type of crossover and mutation operators used by us are explained below and is similar to our previous work [41].

i. One-point crossover: Crossover operation transforms particles to new particles by randomly choosing the cut point. The particles are sliced at the cut point and tails of the sliced particles are swapped with each other to form two new particles. It is shown with help of an example in Figure 8.

ii. Mutation operator: Mutation operator can be applied in three ways and are described below:

- Mutation of parameter values: Parameter values of methods is changed with the another value of same type. The Figure 8 shows the example in which parameter value passed to class B is changed from 1 to 2.
- Mutation of Constructor: Constructors can be mutated by changing it with another constructor with different parameters. Constructor mutation can be seen in the example shown in Figure 8.
- Mutation of methods: It is carried out by insertion of new method invocations or through removal of some methods. It is also shown in the example of Figure 8.

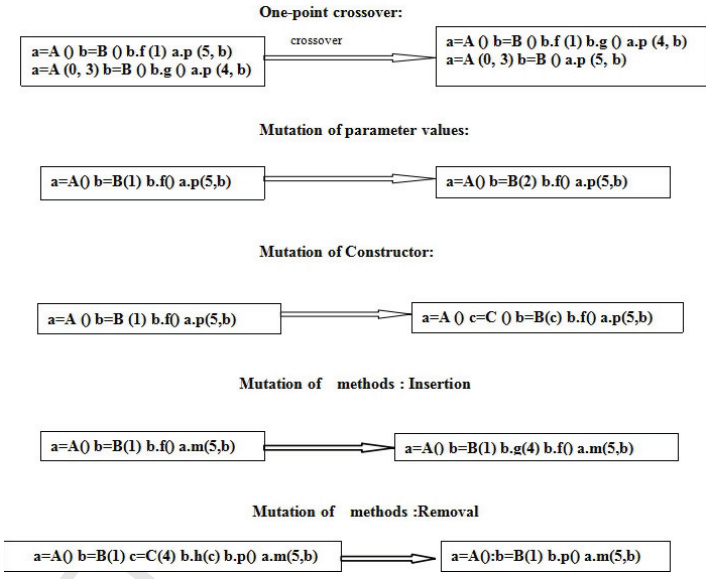

Figure 8: Example demonstrating Crossover and Mutation [41]

The above steps are augmented with the minimization of the redundant test cases and prioritization of the minimal test suite. The test cases are finally printed in the form of JUnit file.

4.3. Cost Evaluator

Our test data generation framework is meant to operate on 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 software under test and testing type. The test goal defines the stopping criteria of test data generation strategy. The cost budget indicates the upper limit of the chargeable amount the client could afford. The testing type presents the alternative choices of testing strategies. Presently, our work is focussed on unit testing as the type of testing offered to the client. In future, other testing strategies may also be offered to the clients as a choice.

The number and type of resources depend upon the size of the software under test, test goal and cost budget. For example, the client provides the following inputs for the software under test

i. Test goal of 90% of branch coverage and generation of test cases for crashes.
ii. Cost budget is $500.
iii. Size of the software under test is 10MB.
iv. Type of testing is unit testing.

The number and type of resources will be calculated based on 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 charger for each crash inducing defects. Both $x and $y will be proportional to the size of the software under test.

$$Bill = n_1 * x + n_2 * y \tag{8}$$

where n_1 denotes the percentage coverage of software under test, n_2 denotes the number of faults detected and *Bill* <= Cost Budget

After the completion of test data generation task, the Cost Evaluator generates the bill. It reads the data stored in the files related to number and type of resources used. Then, it sends the bill to the starter module. The starter sends the bill to the client and asks for the payment using payment gateway service. The test report is provided to the client upon receiving the payment.

5. Performance Evaluation

To perform the empirical evaluation of the proposed strategy carefully, the following research questions have been designed:
RQ1: Does the proposed strategy improve the fault detection and coverage capabilities?
RQ2: What will be the performance of the map reduce based strategy when compared with the sequential strategy?
RQ3: What will be the performance of the proposed strategy when compared with other existing cloud based strategies in terms of time taken and resources utilized to produce the desired results?

5.1. Case Study

We took five open source libraries provided by Apache as our case study subjects as shown in the Table 5. The case study subjects chosen for testing covers varied range of Lines of Code(LOC) i.e. from 6337 to 19041 LOC. We evaluated our strategy over the Apache Multi-node Hadoop cluster setup in the LAN. We took ten nodes in the cluster each having the hardware configuration as Intel Core i7-4980HQ CPU as the processor with quadra core 2.8 GHz processor, 16 GB of RAM. The software installed on the machines are Hadoop 1.04 and Ubuntu 12.04. In the master-slave mode, one of the nodes was made master and other nine acted as slaves. The job of test data generation of a jar file is distributed among 30 Mappers in parallel. The generated test data is written to the HDFS in the form of JUnit File. The test adequacy criterion chosen by us are branch coverage and faults detection. The performance adequacy criterion are the percentage increase in APFD score, percentage increase in coverage per unit time and the percentage decrease in time while using the same allotted computational resources. Table 4 shows the parameter settings used for experiments.

Table 4: Parameters Settings

Population Size	200
Crossover Probability	0.9
Mutation Probability	0.2
Number of Iterations	1000

Table 5: Features of Case Study

Case Study	LOC	No. of Branches	No. of Classes	
			Public	All
JC Java Collections	6,337	3,531	30	118
JT Joda Time	18,007	7,834	131	199
CP Common Primitives	7,008	2,874	210	213
CC Common Collections	19,041	8,491	244	418
GC Google Collections	8,586	3,549	91	331

5.2. Experimental Design, Results & Analysis

In order to find out the exact cause-and-effect relationships, the guidelines stated in the work [9][10] have been followed. We conducted numerous experiments to obtain the suitable answers to the above designed research questions. The design of experiments have been planned as stated below:

i. Null Hypothesis Significance Testing(NHST): It is accomplished by defining null hypothesis and alternative hypothesis to compare the strategies.
ii. Two-tailed Mann-Whitney U-test: It is used to perform statistical analysis in conjunction with NHST.
iii. Evaluation of Vargha and *Delaney's* $\hat{A}_{12}$ statistics: It is standardized effect size which act as an objective and standardized measure of the magnitude of observed effect.

We have empirically compared our proposed framework with Yeti (based on random strategy)[24] and strategy based on traditional genetic algorithm (MR_G) in cloud computing environment. We have chosen these strategies because of their similarity with our approach.

RQ1: Does the proposed strategy improve the fault detection and coverage capabilities?

To answer the RQ1, we executed all the three strategies(which includes our proposed strategy, yeti and genetic algorithm based strategy) w.r.t APFD for each case study subjects. APFD measures the faults detected by a test suite, higher the score of APFD the higher is the percentage of fault revealed by a particular test suite. It is evaluated using the following formula:

$$APFD(T, S) = 1 - \sum_{i=1}^{l} detect(i, T) \div nl + 1 \div 2n \tag{9}$$

In the above equation, T represents Test Suite, S represents Software Under Test, 'n' represents total number of test cases and 'l' represents the number of faults contained in the software under test.// The faults have been injected with the aid of fault injecting tool JACA [42]. It helps in assessing the quality of test data generated by our proposed framework. The seeded faults simulates classic programming errors such as use of wrong relational operator, wrong parameter value and condition missing etc. The percentage increase in APFD score is shown in the box plots in the Figure 9 and found that on the average percentage increase in APFD score comes out to be 4.86 and 19.56 when compared with MR_G and YETI (based on random strategy) respectively.

We have chosen percentage Coverage achieved per unit execution time (C) as a criterion for performance comparison. As coverage should always be maximized whereas execution time should be minimized, so percentage coverage achieved per unit execution time should be maximized for a better strategy. We executed the strategies 30 times to gather sufficient information on the probability distribution of C, performance of the compared algorithms is shown in Table 6 and is also shown graphically Figure 10.

Null Hypothesis for our statistical test states that there is no difference between our proposed framework and the existing strategy. Alternative hypothesis states our proposed framework is better than the existing strategy. We performed pair wise comparisons between the proposed framework, yeti and MR_G. The level of significance is set to 0.05. We have also used Vargha and *Delaney's* $\hat{A}_{12}$ statistics, as standardized effect size [9][10] in conjunction with Null Hypothesis Significance Testing (NHST). An effect is an objective and standardized measure of the magnitude of observed effect. Vargha and *Delaney's* $\hat{A}_{12}$ statistics measures the probability that running algorithm A yields higher C values than running another algorithm B. It can be computed easily as per the following formula [9][10].

$$\hat{A}_{12} = (R1/m - (m + 1)/2)/n \tag{10}$$

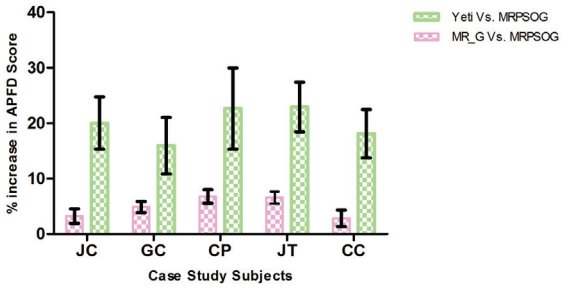

Figure 9: Boxplots of percentage increase in APFD for the case study subjects.

where R1 is the rank sum of the first data group we are comparing. In the above formula, m is the number of observations in the first data sample, whereas n is the number of observations in the second data sample.
Figure 10 shows the box plots showing the comparison of all the strategies w.r.t SUT which clearly presents the C score of novel strategy is better than others. Results in Table 6 exhibits positive results with high statistical confidence which certainly answers RQ1. The average percentage increase in coverage per unit time comes out to be 48.45. Hence our proposed framework is better than the other strategy.

Table 6: Comparison with Existing Traditional Genetic Algorithm based Testing Models in Cloud Computing Environment. Each algorithm has been stopped after evaluating up to 200,000 solutions. The reported values are calculated on 30 runs of the test.

Case Study	Strategy	Minimum	Median	Maximum	Mean	Std. Deviation	Std. Error	Vargha & Delaney's Statistics
JC	MR_PSOG	3.161	3.3	3.414	3.3	0.089	0.016	1
	MR_G	1.957	2.044	2.089	2.034	0.041	0.007	
	Yeti	1.67	1.76	1.857	1.76	0.04	0.009	
JT	MR_PSOG	2.13	2.178	2.227	2.17	0.028	0.005	1
	MR_G	1.517	1.559	1.603	1.56	0.026	0.004	
	Yeti	1.22	1.29	1.35	1.28	0.03	0.006	
CP	MR_PSOG	1.843	1.86	1.898	1.869	0.02	0.003	1
	MR_G	1.265	1.413	1.419	1.353	0.069	0.012	
	Yeti	1	1.08	1.22	1.09	0.05	0.01	
CC	MR_PSOG	1.8	1.84	1.898	1.841	0.025	0.004	1
	MR_G	1.217	1.25	1.284	1.25	0.017	0.003	
	Yeti	0.93	0.98	1.06	0.99	0.03	0.005	
GC	MR_PSOG	2.293	2.365	2.436	2.363	0.051	0.009	1
	MR_G	1.483	1.53	1.569	1.529	0.022	0.004	
	Yeti	1.2	1.27	1.38	1.28	0.057	0.01	

RQ2: How well does the map reduce based strategy perform compared to the sequential strategy?
The Apache Map Reduce based strategy achieves the same APFD and Coverage as that by the sequential version in much less time. The speedup of 6x to 30x with ten node Hadoop cluster has been observed and is shown in the Fig 11. In order to provide a more concrete quantitative analysis and statistical significance for the answers to RQ2, the results obtained has been compared using Two-tailed Mann-Whitney U-test.

RQ3: How well does the proposed strategy perform compared to other existing cloud based strategies in terms of time taken and resources utilized to produce the desired results?
To answer the above research question, we compared our hybrid strategy with MR_G [35] [38] and Yeti [24]. To measure the performance of our proposed framework, we observed the resources and time required for generating test cases. The resource requirement has been measured by measuring the number of mappers. It has been found that our novel strategy is able to produce the required results in much less time and resources than the other existing strategy. This is due to the fact that our strategy is able to find optimal solutions in much lesser iterations than genetic based strategy and random strategy based approach. This has been achieved because of the following reasons:

i. New particle are evolved by applying genetic operators over PBest and GBest particles. Only the particle

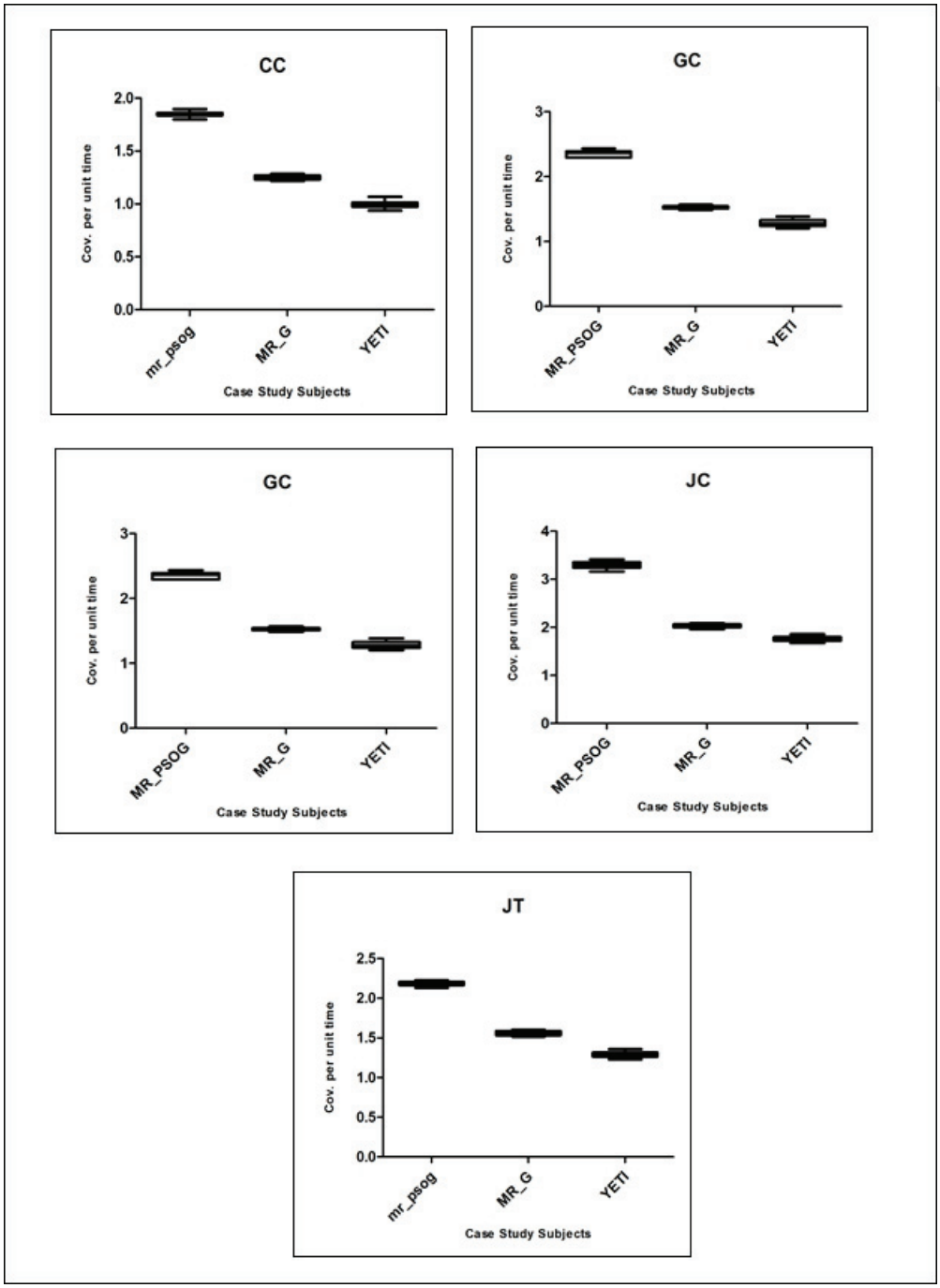

Figure 10: Boxplots of percentage coverage per unit time (C) for all case study subjects.

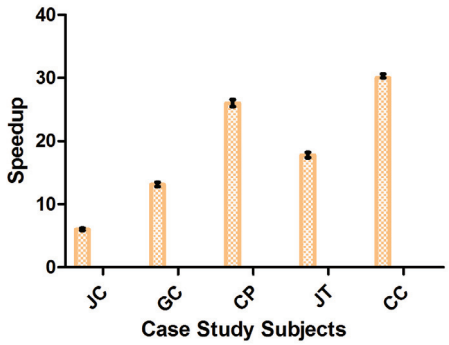

Figure 11: Boxplots of speedup for all case study subjects.

possessing best fitness values are retained. This feature has helped to obtain the faster convergence rate in lesser number of resources.

ii. The operations related to fitness evaluations and genetic operations are carried out on the Mappers. This has drastically reduced the unnecessary writes on the HDFS. This also helped us in reducing the overhead associated with the data movements to the Reducer. All the necessary operations has been performed in parallel with the Mapper.

iii. The test cases were prioritized and minimized effectively. The redundant test cases were deleted. Thus generated test suites were compact and took less time while execution than the random strategy based frameworks.

Our strategy is entirely different from the work proposed in [35][38] in a way because we restricted all the operations related to fitness evaluations and genetic operations to the Mappers and there is no reducer. On the other hand, the authors in [35][38] performed fitness evaluation through Mappers whereas other genetic operations were performed over Reducer. This results delay in the completion time of the MapReduce job of test data generation as the intermediate data from the mappers needs to be saved on the HDFS before start of the Reducers. We have performed experiments and found that our proposed framework is on the average 31.08% faster than traditional genetic algorithm based test data generation. These results have been shown in Figure 12.

6. CONCLUSIONS & FUTURE WORK

In this article, we have presented an automated test data generation framework to generate optimal test suites that not only covers maximum branches but also detects faults in minimum possible time. It applies pareto dominance concept to resolve conflicts among multiple objectives and finds the best optimal solution. Apart from that, we utilized Apache Hadoop MapReduce framework for making the framework cloud ready. Apache Hadoop MapReduce framework facilitates convenient and cost-effective distribution of the tasks for test case generation of Java API to several mappers. The comparison of existing test data generation models in cloud environment with our proposed framework has also been carried out. Experimental results conducted on hadoop cluster with ten nodes clearly reveal the effectiveness of our framework. As our future work, we intend to extend our framework available as a service over actual cloud provided by Amazon Web Service where users can upload the services in addition to the desktop application to get the optimal test suites as an output. The benefits of cloud-based architecture by considering parameterized testing objectives would also be evaluated through controlled experiments.

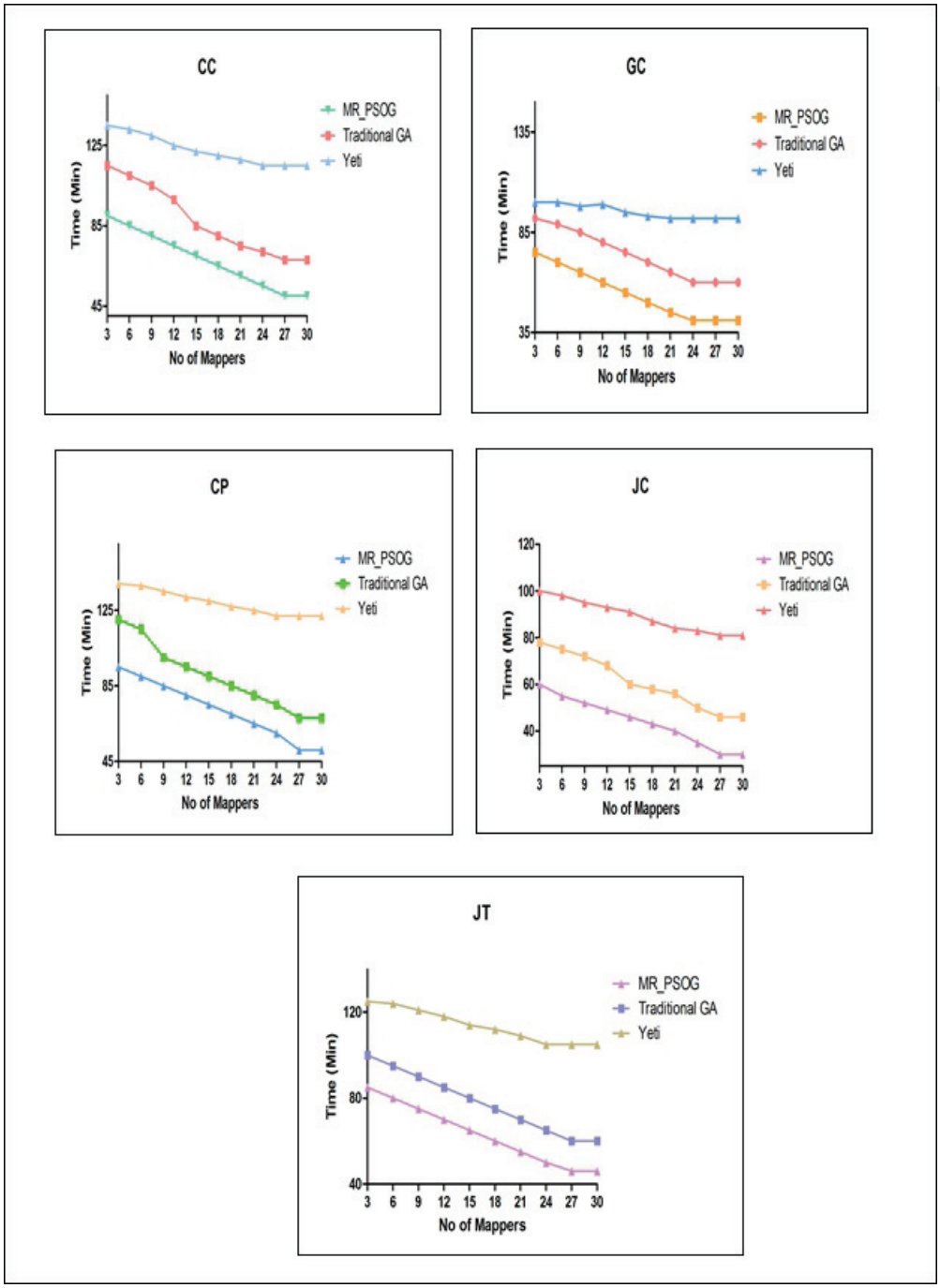

Figure 12: Comparison of models from resource requirement aspect

[1] Mark Harman. 2007. The Current State and Future of Search Based Software Engineering. In *2007 Future of Software Engineering(FOSE' 07)* IEEE Computer Society, Washington, DC, USA, 342-357.

[2] Phil McMinn. 2004. Search-based software test data generation: a survey: Research Articles. *Softw. Test. Verif. Reliab.* 14,2 (June 2004),105-156.

[3] Aiguo Li and Yanli Zhang. 2009. Automatic Generating All-Path Test Data of a Program Based on PSO. In *Proceedings of the 2009 WRI World Congress on Software Engineering - Volume 04* (WCSE'09),Vol. 4. IEEE Computer Society, Washington, DC, USA,189-193.

[4] Sheng Zhang; Ying Zhang; Hong Zhou; Qingquan He, "Automatic path test data generation based on GA-PSO",*Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on* , vol.1, no.,pp.142,146, 29-31 Oct. 2010

[5] Andreas Windisch, Stefan Wappler, and Joachim Wegener. 2007. Applying particle swarm optimization to software testing. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*(GECCO'07). ACM, New York, NY, USA, 1121-1128.

[6] Xiang Chen; Qing Gu; Jingxian Qi; Daoxu Chen, "Applying Particle Swarm Optimization to Pairwise Testing,"*Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, vol.,no., pp.107,116, 19-23 July 2010.

[7] Booher, Jeremy. "Computability: Turing Machines and the Halting Problem." (2008).

[8] Harman, M., Lakhotia, K., and McMinn, P. (2007a). A Multi-Objective Approach to Search-based Test Data Generation. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1098–1105, London, England. ACM.

[9] Arcuri, A.; Briand, L., "A practical guide for using statistical tests to assess randomized algorithms in software engineering," *Software Engineering (ICSE), 2011 33rd International Conference on* , vol., no., pp.1,10, 21-28 May 2011.

[10] Shaukat Ali, Lionel C. Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. 2010. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. *IEEE Trans. Softw. Eng.* 36, 6 (November 2010), 742-762.

[11] G. Rothermel, R. Untch, C. Chu, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Trans. Software Eng.*, vol. 27, no. 10, pp. 929-948, Oct. 2001.

[12] Apache Hadoop Map Reduce, "`http://www.hadoop.apache.org/mapreduce`" Accessed May 2013

[13] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters.*Commun. ACM* 51, 1 (January 2008), 107-113.

[14] D. A. Patterson, Technical perspective: the data center is the computer,*Communications of the ACM* 51-1, 105,January 2008.

[15] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[16] James Kennedy and Russell C. Eberhart. Particle swarm optimization.In *International Conference on Neural Networks IV* , pages 1942–1948,Piscataway, NJ, 1995. IEEE Service Center. Learning(1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[17] P. Rabanal, I. Rodríguez, F. Rubio, A functional approach to parallelize particle swarm optimization, In: *Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'12*, 2012.

[18] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "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, CCGRID'10* Washington, DC, USA, pp. 631-636, IEEE Computer Society, 2010.

[19] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia, "VATS: Virtualized-Aware Automated Test Service," in *Quantitative Evaluation of Systems, 2008. QEST '08. Fifth International Conference on*, pp. 93-102, September 2008.

[20] S. Bucur, V. Ureche, C. Zamfir, and G. Candea, "Parallel Symbolic Execution for Automated Real-World Software Testing," in *Proceedings of the sixth conference on Computer systems, EuroSys '11,* (New York, NY, USA), pp. 183-198, ACM, 2011

[21] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. 2010. Cloud9: a software testing service. *SIGOPS Oper. Syst. Rev.* 43, 4 (January 2010), 5-10.

[22] Alexey Lastovetsky. 2005. Parallel testing of distributed software.*Inf. Softw. Technol.*47, 10 (July 2005), 657-662.

[23] S. Misailovic, A. Milicevic, N. Petrovic, S. Khurshid, and D. Marinov, "Parallel test generation and execution with korat," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* ESEC-FSE' 07, pp. 135-144, 2007

[24] M. Oriol and F. Ullah, "Yeti on the Cloud," in *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on* pp. 434-437, April 2010.

[25] L. Yu, W. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao, "Testing as a Service over Cloud," in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering* 2010, pp. 181–188

[26] Z. Ganon and IE Zilbershtein, "Cloud-based Performance Testing of Network Management Systems" In *Computer Aided Modeling and Design of Communication Links and Networks* 2009. CAMAD'09, IEEE 14th International Workshop on, pages 1-6, IEEE, 2009.

[27] G. M. Kapfhammer, "Automatically and transparently distributing the execution of regression test suites," In *Proceedings of the 18th International Conference on Testing Computer Software*, 2000.

[28] A. Duarte, W. Cirne, F. Brasileiro, P. Machado, "Gridunit: software testing on the grid," in *Proceedings of the 28th international conference on Software engineering* 2006, pp. 779-782.

[29] A. Duarte, G. Wagner, F. Brasileiro, W. Cirne, "Multienvironment software testing on the grid," in *Proceedings of the 2006 workshop on Parallel and distributed systems: testing and debugging* 2006, pp. 61-68.

[30] Parveen, T., Tilley, S., Daley, N., Morales, P., "Towards a distributed execution framework for JUnit test cases," *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, vol., no., pp.425,428, 20-26 Sept. 2009.

[31] Priyanka, Inderveer Chana, and Ajay Rana. 2012. Empirical evaluation of cloud-based testing techniques: A Systematic Review. *SIGSOFT Software Eng. Notes* 37, 3 (May 2012), 1-9.

[32] Fraser, G.; Arcuri, A., "Evolutionary Generation of Whole Test Suites," *Quality Software (QSIC), 2011 11th International Conference on* , vol., no., pp.31,40, 13-14 July 2011.

[33] P. Tonella, Evolutionary testing of classes, in: *Proceedings of the International Symposium on Software Testing and Analysis* (ISSTA), 2004, pp. 119-128.

[34] Deb,K.; Pratap, A.; Agarwal, S.; Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II," *Evolutionary Computation,*

IEEE Transactions on , vol.6, no.2, pp.182,197, Apr 2002.
[35] Linda Di Geronimo, Filomena Ferrucci, Alfonso Murolo, and Federica Sarro. 2012. 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 (ICST '12). IEEE Computer Society,* Washington, DC, USA, 785-793.
[36] Inderveer Chana, Priyanka Chawla. Testing Perspectives of Cloud Based Applications. Springer, Software Engineering Frameworks for Cloud Computing Paradigm,Mahmmood, Z and Saeed, S (eds.), http://www.springer.com/computer/communication+networks/book/978-1-4471-5030-5
[37] Shin Yoo and Mark Harman. 2010. Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation. J. Syst. Softw. 83, 4 (April 2010), 689-701.
[38] S. Di Martino, F. Ferrucci, V. Maggio, and F. Sarro, "Towards migrating genetic algorithms for test data generation to the cloud", in. IGI Global, 2012, ch. 6,pp. 113–135.
[39] Fujitsu, "Confidence In Cloud Grows, Paving Way For New Levels Of Business Efficiency". Fujitsu Press Release, November 2010. Available online at http://www.fujitsu.com/uk/news/ Last Accesed: September 2014.
[40] Jacaco Web Page *http://www.eclemma.org/jacoco/.*
[41] Priyanka,Inderveer Chana, Ajay Rana. A Novel Strategy for Automatic Test Data Generation using Soft Computing Technique[J]. Frontiers of Computer Science 9(3): 346-363 (2015).
[42] Jaca Web Page *http://www.ic.unicamp.br/ eliane/JACA.html.*
[43] Pargas R P, Harrold M J, Peck R R. Test-data generation using genetic algorithms. Software Testing Verification Reliability, 1999, 9(4): 263-282.
[44] Windisch A, Wappler S, Wegener J. Applying particle swarm optimization to software testing. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO '07. 2007, 1121-1128.
[45] Wegener J, Baresel A, Sthamer H. Evolutionary test environment for automatic structural testing. Information and Software Technology, 2001, 43(14): 841-854.
[46] Jones B F, Sthamer H, Eyres D E. Automatic test data generation using genetic algorithms.Software Engineering Journal, 1996, 11(5): 299-306.
[47] Xanthakis S E, Skourlas C C, LeGall A K. Application of genetic algorithms to software testing. In: Proceedings of the 5th International Conference on Software Engineering and its Applications. 1992, 625-636.
[48] Harman M, Jones B. Search-based software engineering. Information and Software Technology, 2001 43(14): 833-839.
[49] Clark J, Dolado J J, Harman M, Hierons R, Jones B,Lumkin M, Mitchell B, Mancoridis S, Rees K, Roper M, Shepperd M. Reformulating software engineering as a search problem. IEE Proceedings-Software, 2003, 150(3): 161-175.
[50] C4J Web Page *http://c4j-team.github.io/C4J/index.html*
[51] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multiobjective optimization test problems". In *Proc. of Congress on Evolutionary Computation*, 2002
[52] Aljarah, I.; Ludwig, S.A., "Parallel particle swarm optimization clustering algorithm based on MapReduce methodology," *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, vol., no., pp.104-111, 5-9 Nov. 2012
[53] McNabb, A.W.; Monson, C.K.; Seppi, K.D., "Parallel PSO using MapReduce," Evolutionary Computation, 2007. *CEC 2007. IEEE Congress on*, vol., no., pp.7-14, 25-28 Sept. 2007
[54] A. Verma, X. Llorà, D.E. Goldberg, R.H. Campbell, "Scaling Genetic Algorithms Using MapReduce". In *Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications (ISDA' 09)* IEEE Computer Society, Washington, DC,USA, 2009, pp.13-18.
[55] C. Jin, C. Vecchiola, R. Buyya: MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms.*eScience* 2008: 214-221
[56] E. Starkloff, "Designing a parallel, distributed test system", Aerospace and Electronic Systems Magazine, IEEE, vol. 16, no. 6, pp. 3 –6, jun 2001.
[57] A. Lastovetsky, "Parallel testing of distributed software," Inf. Softw. Technol., vol. 47, no. 10, pp. 657–662, 2005.
[58] G. M. Kapfhammer, "Automatically and transparently distributing the execution of regression test suites," in In Proceedings of the 18th International Conference on Testing Computer Software, 2000.
[59] A. Duarte, W. Cirne, F. Brasileiro, and P. Machado, "Gridunit: software testing on the grid," in ICSE'06: Proceedings of the 28th international conference on Software engineering. New York, NY, USA: ACM, 2006, pp. 779–782.
[60] A. Duarte, G. Wagner, F. Brasileiro, and W. Cirne, "Multienvironment software testing on the grid," in PADTAD '06: Proceedings of the 2006 workshop on Parallel and distributed systems: testing and debugging. New York, NY, USA: ACM, 2006, pp. 61–68.
[61] R. N. Duarte, W. Cirne, F. Brasileiro, P. Duarte, and D. L. Machado, "Using the computational grid to speed up software testing," in Proceedings of 19th Brazilian Symposium on Software Engineering, 2005.
[62] Y. Li, T. Dong, X. Zhang, Y. duan Song, and X. Yuan, "Largescale software unit testing on the grid", may 2006, pp. 596 –599.
[63] Cognizant, "Taking Testing to the Cloud". Cognizant Whitepaper, September 2011. Available online at http://www.cognizant.com/Taking-Testing-to-the-Cloud.pdf Last Accesed: September 2014.