A Framework for Testing As a Service over Cloud

A Thesis submitted

for the award of the degree of

DOCTOR OF PHILOSOPHY

by

Priyanka
Roll No: 951003010

Under the supervision of

Dr. Inderveer Chana
Professor, CSED
Thapar University, Patiala

Dr. Ajay Rana
Professor
Amity University, Noida

Computer Science and Engineering Department

Thapar University, Patiala-147004, INDIA

[April 2016]

To

My Guruji Sri Sri RaviShankarji

and

My Teachers, Parents and Family

Acknowledgement

First of all and the greatest importance, I would be thankful to GOD for all his blessings without which nothing of my work would have been carried out.

I am highly indebted to Prof. (Dr.) Prakash Gopalan, Director, Thapar University, Patiala for providing me the opportunity to pursue my Ph.D. and course work.

This thesis would not be possible without the continuous guidance from my supervisor Dr. Inderveer Chana, Professor, Computer Science & Engineering Department, Thapar University, Patiala. The extensive discussions with her have helped me to stay on the right track towards finishing this work. She was always patient and helpful whenever her guidance and assistance was needed. She also helped me by sparing her valuable time for reviewing my experimental setup, publications, reports and presentation of my work from time to time. She also helped me in shaping my initial work of thesis during synopsis submission and I would like to acknowledge her with deferential thanks.

I owe countless and immense gratitude to my supervisor Dr. Ajay Rana, Director, Amity University for his day and night advice, guidance and support throughout the entire work. As my supervisor, he provided me with the encouragement and freedom to pursue my own ideas. He taught me how to do the research. In other words, he taught me how to make gold out of stones. He always remained tranquil while outlaying his edifying time on me. I am also grateful to him for spending several hours reviewing the document for accuracy and providing valuable feedback during the several iterations that this document went through. Therefore, I want to show my deep appreciation to him.

I would also like to convey my thanks to Dr. Maninder Singh, Head, Computer Science & Engineering Department for providing me the continuous feedback throughout my Ph.D. work. I am thankful to my doctoral committee members Dr. Neeraj Kumar, Dr. V. P. Singh Kaushal and Dr. M.D Singh for their constructive comments and ensuring the progress of my research work in the right direction. Their constants opinions and ideas help me to reach at this stage of my course. The helped rendered by them is greatly accredited. My deep regards to Dr. O.P Pandey, Dean of Research and Sponsored Projects for his enormous help in completion of my course work.

I will go amiss if I forget to thank Dr. Deepak Garg, Dr. Rajesh Kumar and Dr. Parteek Bhatia for their valuable suggestions during various progress monitoring presentations. I wish to thank all the faculty and staff members of Computer Science and Engineering Department of Thapar University, Patiala for their co-operation and support.

Lastly, I would like to pay my gratitude to my daughter Samridhi, my husband and to all friends for their love and encouragement all through my life. They have always helped me at every twirl of my life when I am in a need. Words cannot truly express my deepest gratitude and appreciation to all of the above.

I am sorry, if I have forgotten someone and I cannot thank everyone enough for the involvement they have shown and the willingness they have expressed to take on the completion of tasks beyond their comfort zones.

Priyanka

Certificate

I hereby certify that the work which is being presented in this thesis titled, "**A Framework for Testing As a Service**" in fulfillment of the requirements for the award of degree of "**Doctor of *Philosophy***" *submitted in* ***Computer Science and Engineering Department*** *of* Thapar University, Patiala is an authentic record of my own work carried out under the supervision of Dr. Inderveer Chana and Dr. Ajay Rana. It refers other researchers works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Priyanka)

Regn. No. 951003010

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Inderveer Chana)
Professor, CSED
Thapar University, Patiala
INDIA

(Dr. Ajay Rana)
Director, ASE
Amity University, Noida
INDIA

Prof. (Dr.) Ajay Rana
Director
Amity School of Engineering
Amity University, Uttar Pradesh
Sector-125, Noida-201301

Abstract

Cloud Computing refers to a new IT paradigm that has the ability to provide shared pool of configurable computing resources from any computing devices such as laptop, tablet, mobile etc. with minimal management effort or support from service provider. Cloud computing paradigm helps software companies to save huge amount of financial costs by relieving them from the tasks of procurement and maintenance. This provides an opportunity to software companies to focus on innovative practices of problem specific solutions.

Software Testing is the most vital aspect of software development that should be performed judiciously to authenticate that the developed software meets approved quality principles. It has been observed and experienced by software industry that expenditure for testing the software is almost 30% of the total cost of the software development. Additionally, application is said to be robust if it has been tested on multiple operating systems and browsers of different versions. This ultimately raises the need of in-house testing labs that have the ability to cater required intricacies and diversities. But these labs often remain inoperative for a long period of time and have been proved as a massive drain on assets and computational resources. Therefore, software testing can be safely moved to the cloud as it offers high-quality testing at reduced cost and time and relieves the software companies from the burden of owning, reserving and maintaining private test labs. Thus, there is a need for offering a framework for testing as a service over cloud and is the motivation of this research work. To achieve the set objectives of addressing the challenges of creating cloud-based testing service, a comprehensive literature review has been done. Prevalent approaches of cloud-based testing models have been studied, analyzed and compared to identify inherent limitations of the existing work. A thorough study of automatic test data generation strategies has also been done as test case design is one of the most significant activities of software testing and an intelligent selection of test cases can radically reduce the cost of software testing. A comparative study of existing soft-computing based test data generation strategies has been presented. Based on the literature survey, it has been found that there is a dire need of an automatic test data generation strategy that detects faults present in the software in minimum time and cost. Soft computing techniques such as genetic algorithms and particle swarm optimization algorithm are found to be more efficient in solving test data generation problem. But, the biggest challenge confronting automated test data generation using soft computing technique is huge computational cost and massive test size. To address the above challenge, Apache Hadoop MapReduce has been used to distribute the task of test data generation to several mappers and enable concurrent execution on different nodes. Due to parallelization, time required to generate the optimal test suites has been reduced. The underlying test data generation strategy of proposed framework is hybrid of Genetic and Particle Swarm Optimization algorithm and uses the concept of pareto-optimality for the evaluation of 'gbest' and 'pbest' particles. Pareto optimal approach helps in finding the optimal solution that possesses best fitness values for multiple objectives.

Cloud computing services face significant obstructions in acceptance due to lack of trust among potential users. A security mechanism has been incorporated in the proposed framework that ensures protection of data and code of different users by implementing role-based access control. Further, cost of service should also be minimized and satisfaction level of customers should be maximized. To address this challenge, a mathematical pricing model has been designed to fulfill the expectations of customers and to maximize the net profit of service providers. Cloud service request model has also been proposed that postulates Service Level Agreements (SLA) between customers and service providers. The proposed framework utilizes open source profiling and prediction tool Starfish to help users in decision making of resource selection. It provides users with the most appropriate cluster configuration parameters like number and type of VMs and MapReduce configuration parameters.

Verification and validation of the proposed framework has been done by conducting experiments on local Apache Hadoop cluster and cluster of AWS EC2 machines. The accuracy of the underlying test data generation strategy has been assessed for various perspectives such as fault detection capability, branch coverage, resources utilized, time taken etc., that proves the effectiveness of proposed test data generation strategies. Further, the efficacy of proposed framework has been assessed for cost and time requirements by executing it in the cloud environment. Statistical and practical differences between the proposed strategy and the other existing strategies have been computed using two-tailed Mann-Whitney U-test. In order to assess the magnitude of the improvements in the proposed strategy, Vargha and $Delaney's\ \hat{A}_{12}$ statistics have been used as standardized effect size in conjunction with Null Hypothesis Significance Testing (NHST). The framework has been compared with existing cloud based testing frameworks such as YETI, Cloud9 and HadoopUnit to validate the outcomes. The results show that the proposed framework successfully and collectively addresses the issues of offering testing as a cloud service.

Contents

List of Figures

List of Tables

List of Publications

International Journals (indexed by SCI)

1. 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). [Impact factor = 0.434]

2. Priyanka Chawla, Inderveer Chana, Ajay Rana. "Cloud-based Automatic Test data generation Framework"[J]. JCSS 82(5): 712-738 (2016). http://dx.doi.org/10.1016/j.jcss.2015.12.001. [Impact Factor = 1.307]

3. Priyanka Chawla, Inderveer Chana, Ajay Rana. "The Framework for Cloud-based Intelligent Software Test data generation Service"[J]. SPE,2016. [Impact Factor = 0.897][Under Review]

International Journal

1. Priyanka, Inderveer Chana, Ajay Rana. 2012. Empirical evaluation of cloud-based testing techniques: A Systematic Review. SIGSOFT Software Eng. Notes. ACM:USA, 2012;37(3):1-9. [Citations=14] [It has been uploaded by East Carolina University, NC, USA on its website http://core.ecu.edu/STRG/materials.html to its Useful Material Section]

International Conferences

1. Priyanka, Inderveer Chana, Ajay Rana. "An Effective Approach To Build Optimal T-Way Interaction Test Suites Over Grid Using Particle Swarm Optimization", in International Conference on Facets of Business Excellence-Leveraging Information Technology for Strategic Advantage on November 4-7, 2011 at India Habitat Center, New Delhi. pp.183-196.

2. Priyanka, Inderveer Chana, Ajay Rana "An Effective Approach to build optimal t-way Interaction Test Suites over Cloud using Particle Swarm Optimization", Third International Conference, CNC 2012, Chennai, India, February 24-25, 2012, published by Springer in Lecture Notes of the Institute for Computer Sciences,Social Informatics and Telecommunications Engineering Volume 108, 2012, pp 193-198. [Scopus Indexed]

Paper Communicated

1. Priyanka, Inderveer Chana and Ajay Rana. "Big data processing frameworks : Concept, Approaches, opportunities and Challenges". Communicated to the SCI Indexed Journal Science of Computer Programming.

1

Introduction

Cloud computing is an emerging area of distributed computing that offers many potential benefits to organizations by making Information Technology (IT) services available as a commodity. The inclination towards cloud computing had started in the late 1980s with the model of grid computing. Cloud computing can be defined as a new style of computing in which dynamically scalable and often virtualized resources are provided as a service over the internet.

Software Testing is the most vital aspect of software engineering that should be performed judiciously to authenticate that the developed software meets approved quality principles. Software testing can be safely moved to cloud to test software on multiple operating systems and browsers of different versions. The quality attributes of software like availability, reliability, security, performance, scalability and elasticity can be attained effectively by cloud testing. The pay-per-use model shortens the development cycle and provides continuous testing services. Thereby, it helps to reduce costs incurred in software development.

This chapter provides a high level view of the thesis. It discusses the fundamental concept of cloud computing and provides the evolution of cloud computing from two perspectives: technology paradigm shift and evolution as a data centre technology. In addition, the concepts of software testing and issues of traditional software testing have been discussed. It further provides motivation to propose software testing in cloud environment. This chapter culminates with the cognizance of the organization of the rest of the thesis along with its contributions.

1.1 Cloud Computing: An Overview

Cloud Computing is the paradigm of computing wherein huge groups of machines are connected in private or public networks to provide data, applications and storage services. It is also inferred as a delivery system of computing services over the internet in the same way as that of the delivery system of a power grid that delivers electricity. The computing services are provided to the users without any burden of complexity and in-depth knowledge of technology. The emergence of this technology has appreciably reduced the computing cost by transforming Capital Expense (CapEx) into Operational Expense (OpEx). Cloud Computing offers an efficient and agile IT services at lower costs that provides an opportunity to software industries to focus on innovation rather than on IT functions of acquirement and management. It not only helps to save money but also helps in saving time and efforts.

The US National Institute of Standards and Technology (NIST) has developed a definition that encompasses various views on cloud computing. NIST defines cloud computing as; "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, application and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]".

With cloud computing technology, users can use a variety of devices, including PCs, Laptops, smartphones and PDAs to access programs, storage and application development platforms over the internet via services offered by cloud computing providers as shown in Figure 1.1.

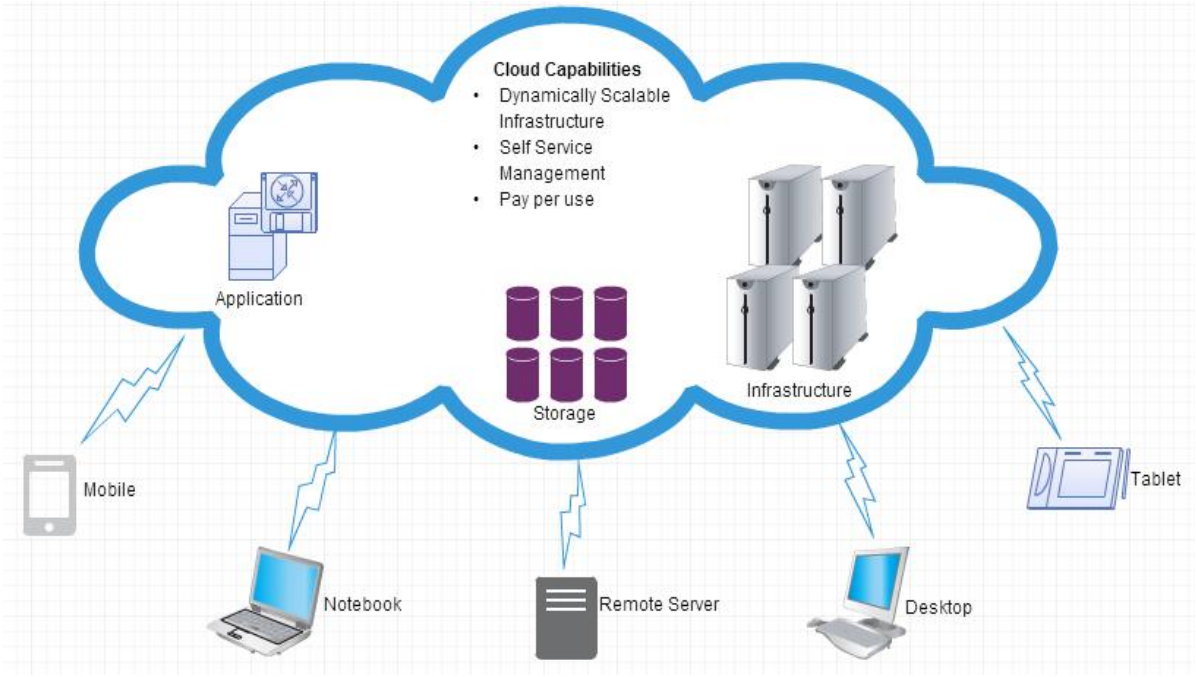

Figure 1.1: Cloud Computing

1.2 Evolution of Cloud Computing

The inclination towards cloud computing started in late 1980s with the concept of grid computing and for the first time, a large number of systems were applied to a single scientific problem requiring exceptionally high levels of parallel computing.

In 1990s, the concept of virtualization was expanded beyond virtual servers to higher levels of abstraction-first the virtual platform, including storage and network resources, and subsequently the virtual application that has no specific underlying infrastructure. During late 90's, utility computing offered clusters as virtual platforms for computing with a metered business model. In the past years, Software As a Service (SaaS) has raised the level of virtualization by introducing a business model in which users would be charged as per the utility of the application to subscribers and not by the resources consumed. Subsequently, the concepts of grid, utility computing and SaaS gave birth to the concept of cloud computing. These emerging concepts facilitate the users to gain access to their applications from anywhere at any time through their connected devices.

The time-line depicting the key events in the history of cloud computing is shown in the Figure1.2.

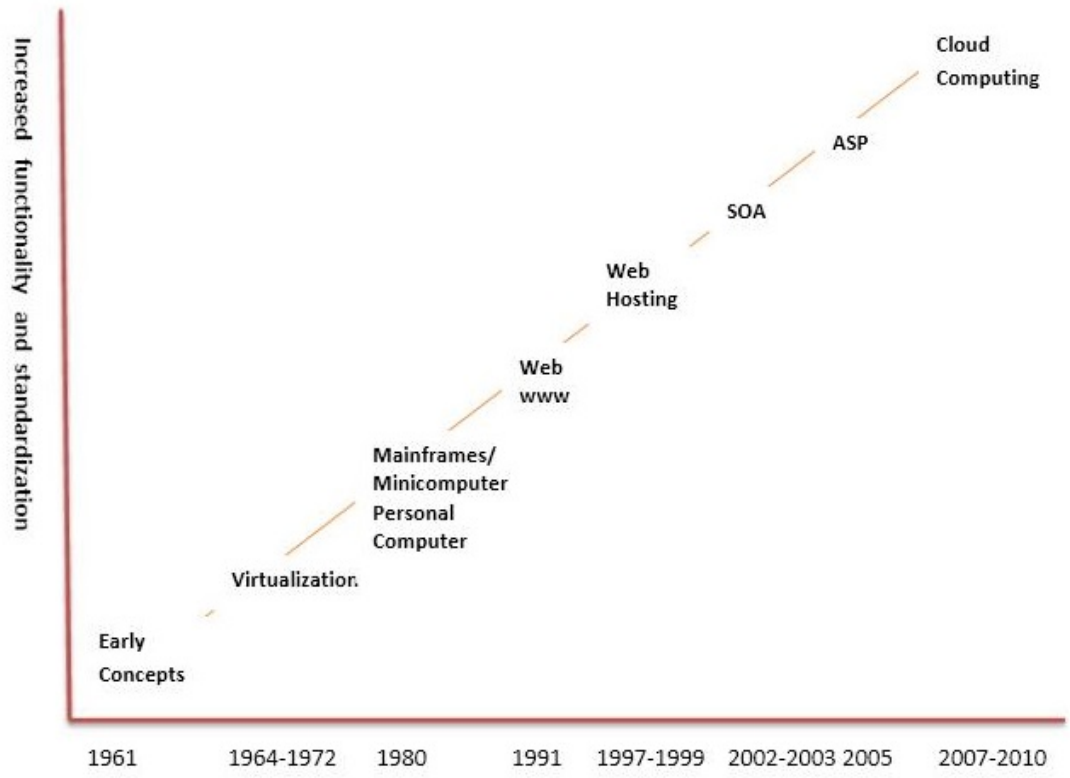

Figure 1.2: Evolution Time-line

1.2.1 Technology Paradigm Shift

Cloud computing refers to leverage multiple computing resources to deliver a "service" to the end user. Gratification of cloud computing technology includes cost savings, high availability and easy scalability. Voas and Zhang showed six phases of computing paradigms, from dummy terminals/mainframes to PCs, network computing to grid and cloud computing [18] as shown in Figure 1.3.

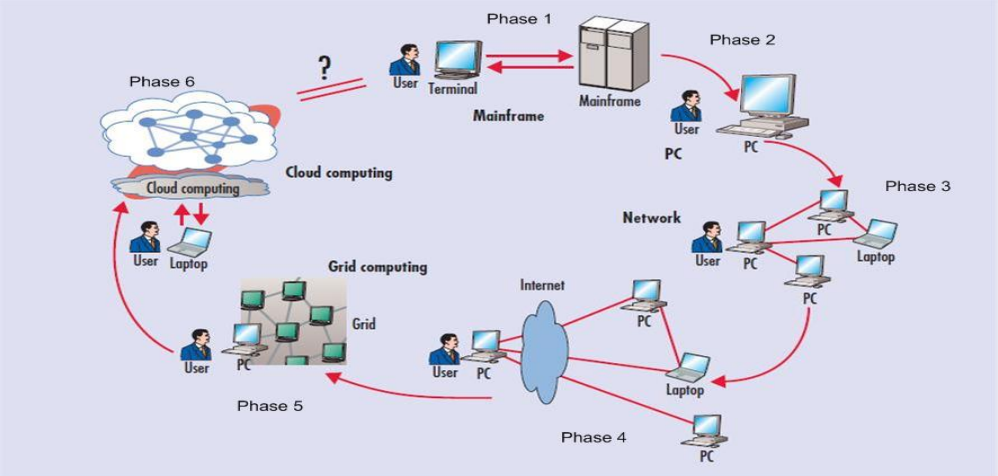

Figure 1.3: Paradigm Shift [18]

In Phase 1, multiple users accessed a central computer named as mainframe through dumb terminals. Mainframe computers were the powerful machines and dumb terminals were connected to those machines. The function of dumb terminals was to provide access to the mainframe. Mainframe computers were very expensive and provided large storage capacity and processing power. Terminals were basically little more than keyboards and monitors and used to provide shared access to a single resource in economical manner.

In Phase 2, mainframe computing became less effective because of its high cost. Lower cost computing was evolved in which there were multiple cheaper stand-alone personal computers (PCs). The PCs had enough computing power to store data and run applications. In this style of computing solution, each cheaper computer ran independently and the process was easier to manage than the mainframe computing wherein one bug within the mainframe could shut down every computer relying on it. However, this lower cost independent computing had many drawbacks. The independence meant computers did not coordinate with each other; data sharing was difficult and any saved resources were negated because each computer had to be changed/fixed/updated individually.

Phase 3 ushered in computer networks that allowed multiple computers to connect to each other. One could work on a PC and connect to other computers through local networks to share resources. Phase 4 saw the advent of local networks that could connect to other local

networks to establish a global network. Users could now connect to the Internet to utilize remote applications and resources. Phase 5 introduced the concept of an electronic grid to facilitate sharing of computing power and storage resources (distributed computing). People used PCs to access a grid of computers in a transparent manner.

In Phase 6, cloud computing helped in utilizing all available resources on the Internet in a scalable and simpler way. As shown in Figure 1.3, there is a conceptual layer in cloud on the Internet that hides all available resources (either hardware or software) and services, but it displays a standard interface. As long as users could connect to the Internet, they had the entire Web as their power PC. Cloud computing thus refered to the techniques that enabled and facilitated this scenario. While comparing to the infinitely powerful internet cloud, PCs seem like lightweight terminals allowing users to utilize the cloud. From this perspective, cloud computing paradigm seems like a re-invention of the original mainframe paradigm. But, the cloud computing paradigm is not as simple as that of mainframe paradigm. Mainframe is a physical machine that offers finite computing power whereas cloud represents all possible resources on the Internet, suggesting infinite power and capacity. Meanwhile, unlike a simple terminal acting as a user interface to a mainframe, a PC in the cloud computing paradigm possesses significant power to provide a certain degree of local computing and caching support. In short, cloud computing is a significant technological trend and can reshape the IT sector and IT marketplace.

After an overview of technology paradigm shift, the next section entails evolution and the emergence of cloud computing from the viewpoint of data center technology.

1.2.2 Evolution as a Cloud Data Center Technology

Cloud computing is an inherent augmentation of the data center technology. Figure 1.4 shows the roadmap of the evolution of cloud computing. Virtualization is the starting point and evolution occurs in five different phases. Each phase is impelled by different requirements, generates different benefits and requires different efforts. These phases require contemplation throughout the process and can be carried out successfully with the aid of following entities:

i. Management Tools: These are required to handle resources, pools and data centers.

ii. Operational Processes: To operate speed/agility.

iii. IT/customer relationship: It helps in the coordination of service providers, service brokers and customers for service requirements and leveraging speed.

iv. Standardization: It facilitates interoperability with external cloud-based offerings and upsurges automation potential.

v. Funding: It helps in gradual transfer from Capital Expenditure model to Operational Model.

The five different phases are as follows:

i. Phase 1: Server Virtualization:- The origin of virtualization is driven by the fact that it helps in reducing capital expenses such as server hardware and energy costs. The concept of virtualization is associated with the consolidation and aggregation of drive density and reduction in the server counts. It encapsulates and abstracts applications from the infrastructure which are very useful to extend the life of old applications. This phase establishes the data center with the aim of cost containment and increased utilization.

ii. Phase 2: Distributed Virtualization:- The virtualization concept evolved the concept of Virtual Machine (VM) that enables the foundation of basic automation tools, rapid provisioning and rapid restart. Data center architecture is based upon virtual machines and also virtualizes different types of networks (like LAN, SAN and IPC) into one single unified fabric. This phase also helps in decreasing capital and operational expenditures. The other benefits like agility, flexibility, speed and efficient management of downtime are also attained. Server utilizations are increased with the help of automation tools and live migrations of VMs. It is also named as Unified Computing as it focuses on virtualization of the data center through pre-integrated architecture that binds up network, server and compute virtualization together.

iii. Phase 3: Private Cloud:- This phase is responsible for an actual realization of cloud computing concepts. IT services are delivered as utility with the help of foundation setup established by previous phases. Public and private clouds can be setup very easily with support of the architecture of Unified fabric and Unified Computing. Private cloud environment is established by various organizations to accomodate short life span workload or peaky workloads, standard web servers and raw computing capabilities.

iv. Phase 4: Hybrid Cloud:- Hybrid cloud helps in the realization of self-service interface and abstraction. The most crucial factor to successfully achieve the benefits of hybrid cloud computing is interoperability on the technology, management, service and funding level. In this phase, external and internal clouds can co-exist in which resources can be shared dynamically.

v. Phase 5: Public Cloud:- Virtualization facilitates a stepping stone path from internal virtualization to full migration of a service to the public cloud. Management of cloud sourcing requires new expertise in managing relationship with multiple cloud providers.

Figure 1.4 depicts the road-map of cloud data-centre that shows the transition from virtualization technology to the public cloud computing [121].

This section discusses about the road-map of cloud computing w.r.t data centres. The next section presents the building blocks of cloud computing.

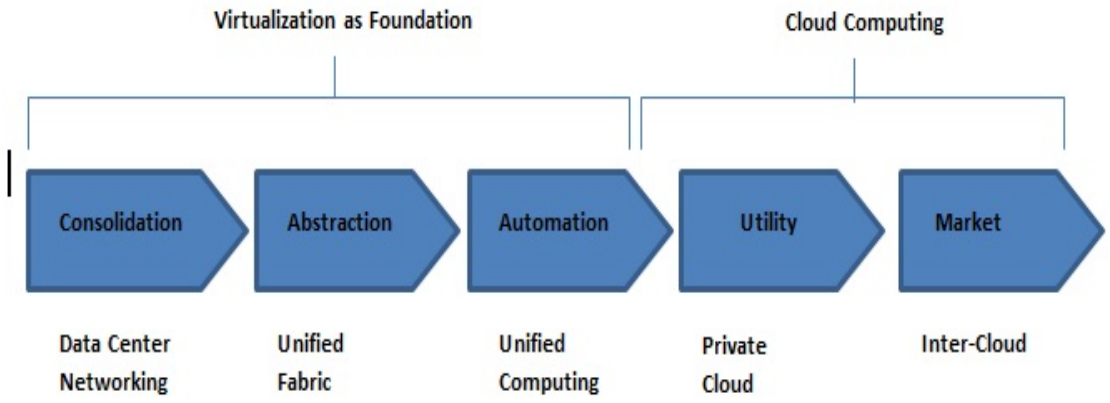

Figure 1.4: Road-map of Data centre

1.3 Cloud Building Blocks

Cloud computing is an applied methodology that facilitates direct cost benefits and it has the potential to transform a data center from a capital-intensive set up to a variable priced environment. The notion of cloud computing is based on elementary principle of reusability of IT capabilities. The cloud computing broadens horizons across organizational boundaries. The building blocks of cloud computing are enabling technologies [2] [3], essential characteristics, deployment models [1] and service delivery models as shown in Figure 1.5.

1.3.1 Enabling Technologies

The enabling technologies are the set of technological advances that made the appearance of cloud computing possible. The enabling technologies can be enumerated as below:

- *Utility Computing* - Utility Computing concept is one of the enablers to sell cloud computing services, in the same way as other utilities are available [4]
- *Virtualization* - It gives the capability to run almost any software on any platform. Several instances can co-exist simultaneously on the same machine. It also helps in improving the usage of the resources while maintaining the security by isolating them.
- *Grid computing* - Grid computing aggregates distributed computing resources which are fundamental for the achievement of necessary scalability of cloud services. However, grid computing is more application-oriented and cloud computing is more service-oriented [67].

- *Web Services* - The service-oriented characteristic of cloud computing requires a standardized architecture like Service Oriented Architecture (SOA) and interoperability and collaboration of Web 2.0 with the associated Web Services and Mashups software.
- *Autonomic Computing* - Autonomic Computing technologies are self-managing solutions to optimize resource usage and they adapt in real time to the customers' needs and operating conditions [68].

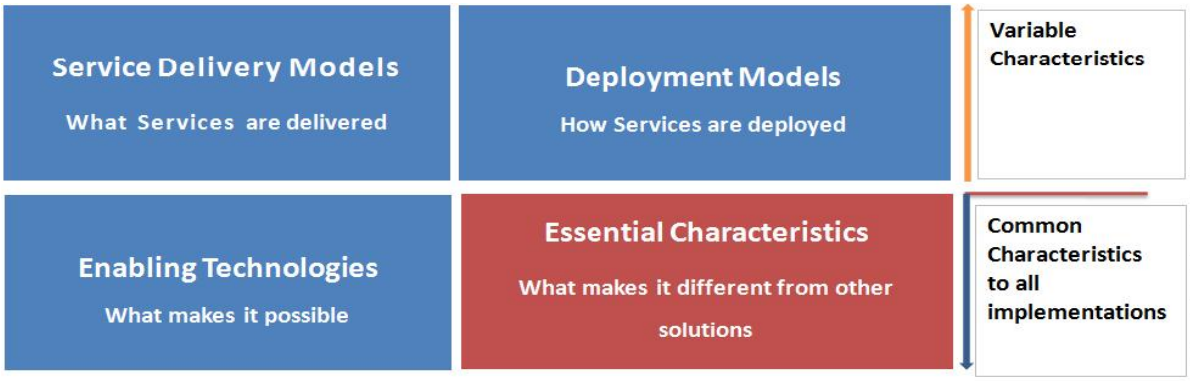

Figure 1.5: Building Blocks of Cloud [1]

The following section portrays the essential characteristics of cloud computing.

1.3.2 Essential Characteristics

Cloud computing provides a shared pool of resources, including data storage space, networks, computer processing power, specialized corporate and user applications. The cloud model is composed of five essential characteristics, three service models, and four deployment models. Cloud computing differentiates itself from other Information and Communications Technology (ICT) implementations by its specific essential characteristics [1] [7]. The essential characteristics make cloud computing unique and differentiate it from other solutions. These characteristics are enumerated as follows:

- Resource pooling
- Scalability
- Ubiquitous network access
- Elasticity
- On-demand self-service
- Measurable

- Pay-per-use

Resources are pooled together that enables scalability and also facilitates ubiquitous network access [13]. But, like any other utility customer's requirements may change over time. Hence, the service must have elasticity to dynamically adapt to the customer needs in real time when required through an on-demand self-service. Moreover, cloud services are measured and paid on the basis of quantity of used services (pay-per-use).

After discussing the essential characteristics, the subsequent section explicates the deployment models of cloud computing.

1.3.3 Deployment Models

Deployment models represent a specific type of cloud environment, primarily distinguished by ownership, size, and access. They state the inherent features and objectives of establishment. Deployment models can be described from two perspectives: Technical perspective and Organizational perspective. These perspectives are described below:

- *Technical Perspective-* According to technical perspective, deployment models can be open cloud, green cloud, virtual private cloud and spatial cloud. Open Cloud guarantees an independence from vendors and facilitates the interoperability between clouds and uses open standards supported by Open Cloud Manifesto or Open Cloud Consortium guarantees [9] [10]. While deploying on low consumption devices for energy saving, it is called Green Cloud [11]. The most common access to public cloud is based on the usage of a Virtual Private Network (VPN) to improve security of the connection and it receives the name of Virtual Private Cloud [12] [13]. Spatial Cloud term is used for spatial data which is geographically dispersed [14].

- *Organizational Perspective-* According to organizational perspective, cloud can be categorized based upon the management of cloud implementations. Cloud can be classified as public cloud, private cloud and hybrid cloud. Public cloud, or external cloud is a deployment where several customers can access cloud services managed by a provider. Clouds deployed for private usage are called Internal cloud or private cloud. In this type of implementation, the management of resources can be done either internally or can be contracted to third parties. Hybrid Cloud is used for implementations where customer accesses private and public clouds.

The next section describes the various service delivery models of cloud computing.

1.3.4 Service Delivery Models

The service delivery model specifies how the IT resources are delivered as a service while the deployment models identify how those services are deployed. Cloud computing provides three types of service models to deliver the computing services over the Internet. Cloud-computing service models are often self-service and cloud services allow individuals and businesses to use software and hardware that are often managed by third parties at remote locations. The available data in the cloud can be accessed very easily and ubiquitously at much lower cost. Service Delivery Models can be categorized into three basic service models and are represented with three layers (IaaS - Infrastructure-as-service, PaaS - Platform-as-a-service, and SaaS - Software-as-a-service). Infrastructure as a Service (IaaS) refers to making available computing resources (storage, processing and networking) as a service. The bills are generated based on the actual time of usage. The consumer is responsible for maintaining operating system and software on the rented machines. Some of the well known IaaS providers are AmazonEC2, Eucalyptus, RackSpace, and GoGrid. The PaaS service provides a software platform for systems that includes operating system, middleware and software development frameworks for developing applications on an abstract platform. Example of PaaS providers and solutions are Microsoft Windows Azure and Google App Engine. In SaaS model, the entire software stack is provided by the cloud environment (from hardware to operating systems and to end user applications). This is an alternative to locally executed software applications and the only client software needed is a web browser. An example of SaaS is on-line versions of typical office applications such as word processors and spreadsheet applications. A well known SaaS provider is Google with its Google Apps including Google Docs and Google Mail. Another major SaaS provider is Salesforce.com which provides on-demand customer relationship management(CRM) software.

Figure 1.6 depicts the nature of cloud services by comparing the levels of technology stacks provided by different service types. The service type definitions represent the separation between the responsibilities of consumer and provider of the services. In case of SaaS, the complete software stack is provided by vendor, but in other two cases it is the responsibility of consumer to provide remainder of the stack.

After discussing the evolution, essential characteristics, deployment models and delivery models of cloud computing, the following section describes about cloud computing applications.

1.4 Cloud Computing Applications

Cloud environment provides successful execution of diverse applications that are generally executed in traditional distributed computing environment. Cloud Computing is an ideal solution

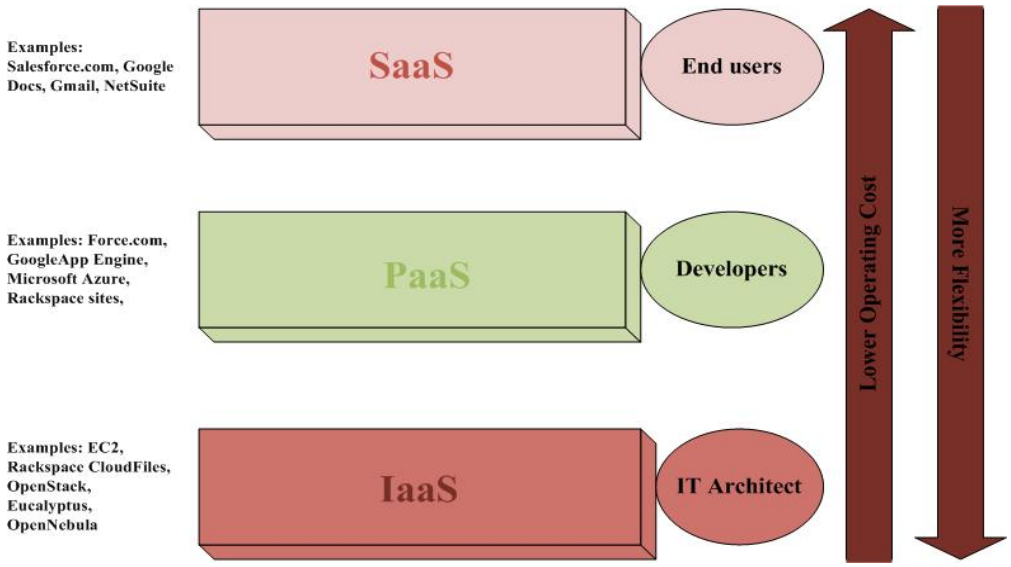

Figure 1.6: Service Models of Cloud

to rapidly design, deploy and execute business applications. It facilitates an efficient use of enterprise resources with minimal management effort by changing the traditional application life cycle significantly. Cloud applications share the design objectives of distributed applications. They act as a hybrid between the desktop applications and traditional web applications. They also provide a rich user experience with minimum response time like the desktop applications. However, cloud applications need not to be installed on local machines as associated data is stored in cloud and managed by service providers and can be updated any time by uploading its newer version.
Classes of applications that can be benefited by cloud computing and contribute further to its momentum are:

i. Mobile interactive applications: These applications reside on mobile devices and help in connecting several organizations to all consumers and employees. Mobile applications generally rely on large data sets hosted in large data centres. Such applications respond in real time to the queries made either by their users or by nonhuman sensors [19].

ii. Parallel batch processing: Batch processing refers to execution of programs in some specified sequence on a computer without manual intervention. Parallel processing uses more than one CPU or processor cores to execute a program at the same time. Cloud computing is very useful for batch processing and analytical jobs for analyzing terabytes of data as these jobs can take several hours to finish. Parallel processing of data reduces the execution time considerably by using hundreds of computers. For example, Peter Harkins, a Senior Engineer at The Washington Post, used 200 EC2 instances (1,407 server hours) to convert 17,481 pages of Hillary Clinton's travel documents into a form that is more friendly to use on WWW within nine hours [20]. Programming abstractions such

as Google's MapReduce [21] and its open-source counter-part Hadoop [22] allow programmers to execute similiar tasks while hiding the operational complexity of parallel execution across hundreds of cloud computing servers.

iii. Business analytics: It is a special case of compute-intensive batch processing that expends a large share of computing resources to understand customer's buying habits and other relevant data. With the continuous growth of online transaction volumes and rapid growth of decision support, the resource balance in database processing is shifting from transactions to business analytics.

iv. Extension of compute-intensive desktop applications: Cloud computing is being used to extend the basic versions of the mathematics software packages such as Matlab and Mathematica to perform expensive evaluations. For example, symbolic mathematics involves large amount of computing per unit of data. An interesting alternative model might be to keep data in the cloud and rely on having sufficient bandwidth to enable suitable visualization and a responsive GUI back to the human user.

v. Web Applications: Web applications are the applications that can be accessed anywhere via the web browser. Web application development through cloud computing provides cost-effective solution to provide specialized services to customers without having to build, maintain or host the applications. Businesses can depend on cloud service providers to collect, maintain and store their data. For example, Multi-tiered web applications like RUBiS [23] and Media Wiki [24] can also be ported to cloud platform [25].

vi. Scientific Workflow Applications: Scientific workflow applications can be executed efficiently over utility computing platforms such as Amazon Elastic Compute Cloud, Google App Engine and academic clouds like Nimbus Science. Few examples of scientific workflow applications are listed below:

- In astronomy, scientists use workflows to generate science-grade mosaics of the sky [26], to examine the structure of galaxies to understand the structure of the universe [27].
- In bioinformatics, workflows are used to understand the underpinnings of complex diseases [28] [3].
- In earthquake science, workflows are used to predict the magnitude of earthquakes within a geographic area over a period of time [4].
- In physics, workflows are used to measure gravitational waves [5] and model the structure of atoms [6].

In this section, application areas of cloud computing have been discussed. The following section describes the differences between traditional applications and cloud-based applications.

1.4.1 Traditional Applications versus Cloud-based Applications

The cloud computing environment is distinct from a traditional environment in terms of application deployment, configuration, execution and management. Traditional applications and Cloud-based applications differ considerably and have been compared on the basis of parameters such as type of users, multi-tenancy, security etc. as shown in Table 1.1.

Cloud applications can also be categorized on the basis of degree of multi-tenancy required for an application; Multi-tenancy is enabled by the concept of virtualization, which supports sharing of computational power, storage, and network resources among multiple clients. In a cloud, a client (tenant) could be a user, a user group, or an organization. Cloud applications are of two types, namely Cloud Hosted applications and Cloud Optimized applications:

i. Cloud Hosted applications: Cloud Hosted applications are the one that can be executed on cloud. In Cloud Hosted applications, multi-tenancy is enabled at the Infrastructure layer i.e. only infrastructure would be shared by providers to support multiple client applications such as Amazon EC2 and Rackspace.

ii. Cloud Optimized applications: Cloud Optimized applications are the one that can leverage cloud to its fullest potential. These applications meet stringent requirements and delivers the maximum return on cloud investment. In Cloud Optimized applications, multi-tenancy is supported at all different layers like infrastructure, application, and database by leveraging a PaaS platform. For example, product Salesforce.com named as Force.com.

Table 1.1: Traditional Application versus Cloud-based Applications

Parameters	Traditional Applications	Cloud-based Applications
User base	Known at Design time	May not be known and could be dynamic
Multi-tenancy	Not required	Assumed
Security	Enforced by application architecture	Service contracts like WS-Security, SAML provided by cloud providers.
Deployment	Only traditional tools	Requires knowledge and utilization of vendor specific Cloud API and tools
Down Time	Upgrades and enhancements are associated with down time	No down time
Infrastructure	Structured and controlled	Un-structured and is un-structured and managed by Cloud fabric
Components	Components co-located in same environment	Components are mostly scattered around one or many Clouds
Testing	In controlled environment	Application (integration) is tested on the Cloud to ensure seamless orchestration between services on one or many Clouds.

This section highlights the disparity existing between the traditional applications and cloud-based applications in context of various parameters. The next section introduces the concept of software testing.

1.5 Software Testing

Software testing ensures correctness, robustness, reliability and quality of software and is thus fundamental to software development. Testers often execute software under a stipulated en-

vironment as well as out of bounds with the intent of finding errors in it [43]. According to IEEE, software testing is the process of analyzing a software item to detect differences between existing and required conditions and to evaluate the features of the software item [44]. Software testing is considered to be a critical element of software quality assurance due to the following reasons [45]:

- To test a developed system for its performance, reliability and quality.
- To ensure long-lasting working of the software without any failure.
- To detect bugs and deviations from software specifications before delivering it to customer.

Software testing comprises of verification and validation tasks. Verification is the process of evaluating a system or component to determine whether the product of a given development phase satisfy the conditions imposed on that phase. Validation is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies specified requirements [IEEE/ANSI]. Hence, software testing is not limited to the execution of software to find defects only but also to test documents and other non-executable forms of a software product.

Software testing often becomes bottleneck in software development and must be performed at every step of the system development [46]. A good testing life cycle begins with the requirement elicitation phase of software development and concludes when the product is ready to be installed or shipped followed by a successful system test. Software testing process must be planned, specified, designed, implemented and quantified. A formal testing process covers activities such as review of the program plans, development of the formal test plans, creation of test documentation (test design, test cases, test software, and test procedures), acquisition of automated tools, test execution, updating of the test documentation and tailoring the model for projects of all sizes [45].

Similar to Software Development Life Cycle (SDLC), Software Testing life Cycle (STLC) also has phases such as Test Planning, Test Design, Test Execution and Post Execution/Test Review as shown in Figure 1.7
The upcoming section discusses the various software testing techniques.

1.5.1 Software Testing Techniques

Software Testing is the procedure of analyzing a program to detect errors at every phase of code development starting from requirement phases to design and code phase. Its main purpose is to ascertain accuracy, comprehensiveness, security and eminence of software products. Software testing techniques have been broadly divided into black box testing and white-box testing.

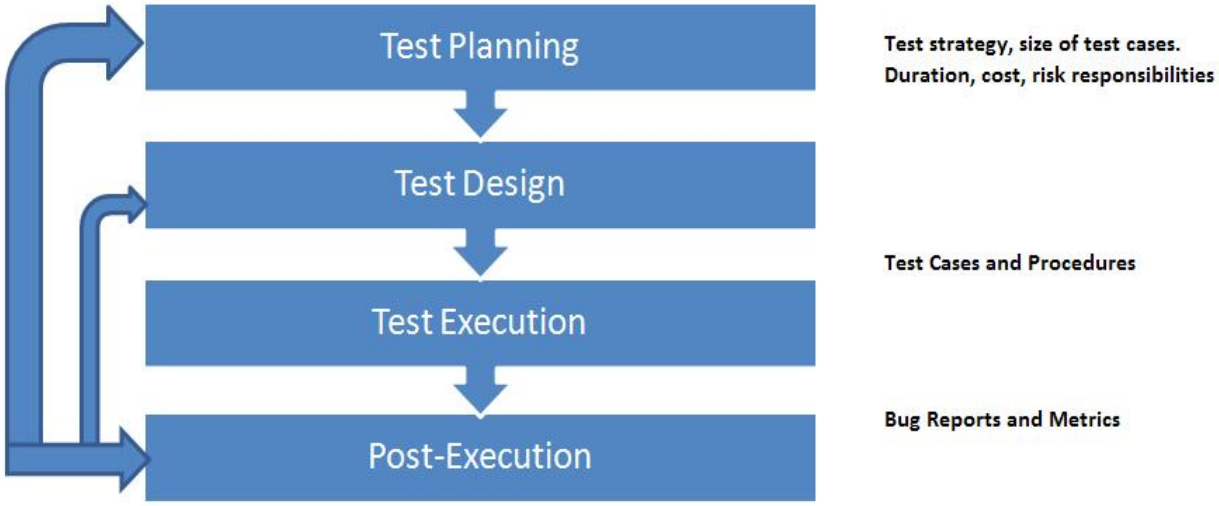

Figure 1.7: Phases of Testing [1]

Black box testing considers the software under test as a black-box which means that the internal structure and behavior of the program under test is not known. Its main objective is to test the functionality of the system. Black box testing entails software requirements, uses cases, executable program and its data. Black box testing is also termed as functional or specification based testing.

White box testing is one of the most important and prevalent software testing technique. It is typically very effective in validating design, decision, assumptions and identification of implementation errors in the software. It is based on analysis of the internal structure, behavior of the program under test and knowledge of system implementation. It is very effective and efficient in validating design decision and assumptions. The main advantage of white box testing is that it reveals error in hidden code and also helps in removing extra lines of code.

After describing the various software testing techniques, the next section discusses the types of software testing.

1.5.2 Types of Software Testing

There are mainly two approaches to software testing: Static and Dynamic Testing.

Static testing is performed by reviewing code, specification documents or design models manually to reveal syntax and logical errors. It is inclined to check the completeness and correctness in order to decrease the defects of requirement document, program code and algorithm. Static testing is also termed as non-execution technique, because in this technique, software under test is not executed. The bugs and errors discovered in this phase of software development are less costly to fix than in any other phase of development.

Dynamic testing refers to an actual execution of program code with a given set of test cases. Dynamic testing can be performed before the program is 100% complete in order to test particular sections of code (modules or discrete functions). Either stubs/drivers or execution from a debugger environment is used in such cases. Dynamic Testing can be further classified based upon a set of five classifiers. These classifiers map a set of features to the set of testing techniques and are explained below [46]:

(i) Source of test generation :- Types of testing can be classified based upon the source input utilized to generate test data. For example, in black-box testing, test cases are generated by taking informal or formal specified requirements as an input whereas, in white-box testing, source code is used in the generation of test cases. In Interface testing, tests are generated using component's interface. Interface mutation and pairwise techniques are used to generate tests from a component's interface specification.

(ii) Life cycle phase :- Testing is often categorized based on the phase in which it occurs. For example, unit testing is performed during early coding phase. Integration testing is done when units are integrated and a large component is formed. System testing is performed during system integration. Regression testing is performed while maintenance and beta testing is performed during post system and pre-release.

(iii) Goal-oriented testing :- Goal-oriented testing looks for specific types of failures. For example, the goal of vulnerability testing is to detect if there is any way by which the system under test can be penetrated by unauthorized users. Similarly, penetration testing is to evaluate security policies and security measures of a system.

(iv) Design testing :- The design of the code should be tested before it is implemented. This can be done during the design phase and this form of testing is known as design testing.

(v) Test Process Models :- Software testing can be integrated into software development life cycle in a variety of ways. It leads to various models for the test process like testing in waterfall model, testing in V-model, Spiral testing and agile testing.

This section summarizes that the black-box testing and white-box testing are the two most fundamental testing techniques that form the foundation of software testing. All the remaining testing techniques fall into either black-box or white-box categories.

The next section portrays the need of migration of software testing to cloud.

1.6 The need of Migration of Software Testing to Cloud

Software testing is a challenging activity for many software engineering projects, especially for large-scale systems. The amount of test cases can range from a few hundred to several thou-

sands, requiring significant computing resources and lengthy execution times. Cloud computing offers the potential to address both of these issues: It offers resources such as virtualized hardware, effectively unlimited storage and software services that can aid in reducing the execution time of large test suites in a cost-effective manner. However, migrating to the cloud is not without cost, nor it is necessarily the best solution to all testing problems. The two perspectives that should be considered before migration of software testing to the cloud are the characteristics of an application under test and the types of testing performed on the application [175].

However, there are several factors that account for migration of testing to the cloud [175]:

(i) One of the main requirements of software testing industry is to manage the variability of internal changes to the test labs. Although traditional testing model does accommodate it but it is a very complicated process as the changes in the application requires administration from the scratch. On the contrary, cloud model facilitates the issues around variability in a very easy manner as servers, connectors and applications can be hosted very conveniently.

(ii) In traditional testing models, it is very hard to build and maintain in-house testing facilities that impersonate real-time environments as it is very time-consuming and delay prone process.

(iii) The concept of cloud computing has unlocked new panorama of software testing by offering low cost and pay-per-use model of infrastructure as a service. It has eliminated the need of upfront capital expenditures. It has also provided other benefits like on-demand flexibility, freedom from holding assets, enhanced collaboration and most significantly reduced time-to-market for key business applications.

(iv) Testing is a periodic activity and requires new environment to be set up for each project. Test labs in companies typically sit idle for longer periods, consuming capital, power and space. It has been observed that approximately 50% to 70% of the technology infrastructure earmarked for testing is often under-utilized.

(v) Conventional software testing incurs high capital cost such as expenditure on hardware, software and its maintenance to simulate user activity from different geographic locations. In case of applications, where rate of increase in number of users is unpredictable or there is variation in deployment environment based on client requirements, cloud testing is more effective. Therefore, cloud testing has become a hot research topic in cloud computing and software engineering community.

(vi) Physical machines restrict the number of replicable test labs as it contributes to high capital costs whenever a large number of testing labs are needed. Mobile applications are associated with such functionalities, tools, and applications which makes anticipation

difficult for testing teams with existing tools. Organizations are caught off guard by the growth of multiple platforms, particularly mobile platforms. Tests are conducted on-premise in a closed environment with limited access. Off-shore locations are required to conduct their own tests locally, even if systems are identical.

(vii) One of the major bottlenecks in traditional testing model is to setup a test infrastructure to run tests in parallel. Cloud platforms offer a state-of-the art solution to this problem. Apart from unit tests, many tests such as integration and load tests depend upon the execution environment. Such environments can be easily replicated by cloning a VM image. Then, these tests can be distributed across many virtual machines to run in parallel.

(viii) In the conventional testing model, software is deployed and installed on the client machines where software companies usually cannot control the execution environment of developed and delivered applications. This lack of information makes the debugging process very cumbersome. On the other hand, in a cloud environment, every application runs inside a virtual machine and it is very easy to take a snapshot of an application and its entire runtime environment during the execution of an application. This has facilitated the debugging process.

(ix) In the traditional model of software testing, the repairing of bugs after delivery of the software is very expensive. Patches are made by the company and it is required to be downloaded and installed by the clients. This issue is very beautifully handled by the cloud-based testing model as software is installed on VM and the changes in the software can be easily updated using the patched version even without the knowledge of the clients.

(x) Testing is considered to be an important but non-business-critical activity. Moving testing to the cloud is seen as a safe bet because it doesn't include sensitive corporate data and has minimal impact on an organization's business-as-usual activities.

(xi) Applications are increasingly becoming dynamic, complex, distributed and component-based, creating a multiplicity of new challenges for testing teams. For instance, mobile and Web applications must be tested for multiple operating systems and updates, multiple browser platforms and versions, different types of hardware and a large number of concurrent users to understand their performance in real-time. The conventional approach of manually creating in-house testing environment that fully mirrors these complexities and multiplicities, consumes huge capital and resources.

(xii) Cloud-based testing service providers offer a standardized infrastructure and pre-configured software images that are capable of reducing errors considerably.

The on-demand provisions of cloud address the above mentioned issues of software testing with one click. Moreover, the effort and resources saved in the development and testing area can be

redeployed for core business pursuits. As per the research carried out by Fujitsu [111], it has been found that testing & application development is the second most workload that can be put to cloud after websites (61%). Research analysis has been shown in Figure 1.8.
This necessitates to design and develop a framework for testing as a service using cloud computing. The next section discusses the cloud-based testing framework, which has been chosen as the area of research for this thesis.

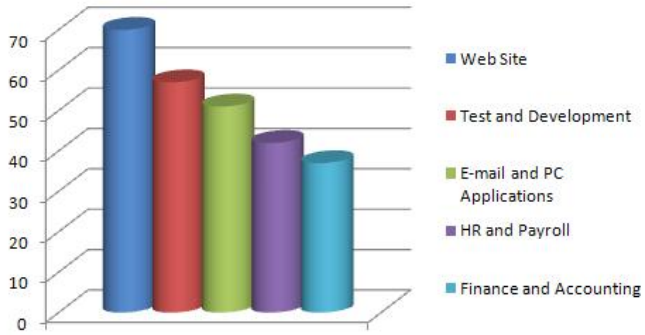

Figure 1.8: Top Application of Cloud [85]

1.7 Cloud-based Testing Framework: Research Motivation

The motivation of this work twigs from the fact that software testing is the most vital aspect that should be performed judiciously to authenticate that the developed software meets approved quality principles. It has been observed and experienced by software industry that expenditure for testing the software is almost 30% of the total cost of software development. Additionally, application is said to be robust if it has been tested on multiple operating systems and browsers of different versions. This ultimately raises the need of construction of in-house testing labs that have the ability to cater required intricacies and diversities. But, these labs often remain inoperative for a long period of time and have been proved as a massive drain on assets and computational resources. Software testing can be safely moved to the cloud as it does not comprise of susceptible business data and has insignificant impact on the organization's accustomed proceedings. Online test labs available on the cloud help in attaining quality attributes like availability, reliability, security, performance, scalability and elasticity. Further, pay-per-use model of cloud computing supports agile software development process by shortening the development cycles and by providing continuous testing services. Therefore, it helps in reduction of costs incurred in software development.

Cloud-based software testing has the capability to offer high-quality testing at reduced cost and time with no upfront cost. It relieves the software companies from the burden of owning, reserving and maintaining private test labs. The resource requirements can be elastically

provisioned and released depending upon the complexity and size of testing jobs. Various software industries like SOASTA, Microsoft, Rackspace, Sogeti, IBM, CloudTesting, Wipro and HP have worked in this direction and provided various cloud-based testing models. Still very limited academic work is available on this subject. There is also very limited utilization of cloud computing infrastructure for automatic software test data generation by users. In addition, other issues associated with the cloud-based testing problem also come to the surface. These issues are summarized below:

- A Cloud-based Testing Framework must take into account the requirement of an intelligent and fast automatic test data generation strategy that reveals faults present in the software at an earlier stage in minimum time during unit testing.
- Cloud-based Testing Framework must be able to address the problems like heterogeneity, scalability, load balancing, communication, frequent failures and synchronization between distributed components.
- It must alleviates the user to handle the tedious tasks like choosing the right cloud cluster configuration for his job.
- The framework must be able to make the application cloud ready so as the service provider may offer huge computational resources required for test data generation in a cost-effective manner.
- It must consider the priority of users for executing a transparent SLA between user and service provider.
- There must be transparency in the pricing schemes of the provided service and should consider mutual benefits of clients and service providers.
- Cloud-based Testing Framework must preserve the interest of users as well as service providers. It must be able to assist service providers to choose optimal combination of resources so that their profit is maximum while maintaining the quality of offered services.
- In addition, it must caters to the need of a secure mechanism for maintaining the safety and security of user data.

Due to the factors listed above, development of an effective Cloud-based Testing Framework has been the motivation behind this work. The following section details the organization of the thesis.

1.8 Organization of Thesis

The thesis is structured as follows:

i. *Chapter 1* Introduction: This chapter provides an introduction of Cloud Computing and Software Testing. It also discusses the evolution of cloud computing from two perspectives: technology paradigm shift and evolution as a data centre technology. In addition, the concepts of software testing and issues of traditional software testing have been discussed. Further, it summarizes the driving factors of migration of software testing over cloud. Last but not the least, motivation factor for this research work, thesis organization and thesis contributions have also been covered. The introduction to the thesis presented in Chapter 1 is partially derived from:

   - Priyanka, Inderveer Chana, Ajay Rana. 2012. Empirical evaluation of cloud-based testing techniques: A Systematic Review. SIGSOFT Software Eng. Notes. ACM:USA, 2012;37(3):1-9. [Citations=14]

ii. *Chapter 2* Literature Review: This chapter presents literature survey on cloud-based testing techniques in academia and corporate. The classification of cloud-based testing models has been done based upon the type of testing technique adopted. The state-of-the-art techniques and related approaches in the context of cloud-based testing and test data generation strategies have been presented. This chapter has helped us in understanding the basic gaps in existing testing models and the associated challenges. Based on the findings of the available literature, the chapter concludes with the problem formulation and corresponding objectives/research goals. This chapter has been derived from:

   - Priyanka, Inderveer Chana, Ajay Rana. 2012. Empirical evaluation of cloud-based testing techniques: A Systematic Review. SIGSOFT Software Eng. Notes. ACM:USA, 2012;37(3):1-9. [Citations=14]
   - Priyanka, Inderveer Chana, Ajay Rana. "An Effective Approach To Build Optimal T-Way Interaction Test Suites Over Grid Using Particle Swarm Optimization", in International Conference on Facets of Business Excellence-Leveraging Information Technology for Strategic Advantage on November 4-7, 2011 at India Habitat Center, New Delhi. pp.183-196.
   - Priyanka, Inderveer Chana, Ajay Rana "An Effective Approach to build optimal t-way Interaction Test Suites over Cloud using Particle Swarm Optimization", Third International Conference, CNC 2012, Chennai, India, February 24-25, 2012, published by Springer in Lecture Notes of the Institute for Computer Sciences,Social Informatics and Telecommunications Engineering Volume 108, 2012, pp 193-198. [Scopus Indexed]

iii. *Chapter 3* Proposed Cloud-based Test Data Generation Technique: This chapter proposes a test data generation strategy in single node environment. A cloud-based automatic test data generation algorithm has been designed by using the hybrid strategy of genetic and particle swarm optimization algorithms. Apache Hadoop MapReduce has been used for parallel implementation to make it ready for the cloud. The concepts of pareto-front and pareto-ranks have also been utilized to generate pareto optimal solutions that satisfy multiple objective functions. It also describes the mathematical model and detailed elaboration of the architecture followed by description of components of the cloud-based testing technique. Furthermore, the chapter details the design of underlying test data generation strategy and discusses the complexity analysis. It also presents the differences between single node and cloud-based automatic test data generation strategies.
This chapter derives from:

- 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). [Impact factor = 0.434]
- Priyanka Chawla, Inderveer Chana, Ajay Rana. "Cloud-based Automatic Test data generation Framework"[J]. JCSS 82(5): 712-738 (2016).
- Priyanka, Inderveer Chana and Ajay Rana. "An Effective Approach To Build Optimal T-Way Interaction Test Suites Over Grid Using Particle Swarm Optimization", in International Conference on Facets of Business Excellence-Leveraging Information Technology for Strategic Advantage on November 4-7,2011 at India Habitat Center, New Delhi. pp. 183-196.
- Priyanka, Inderveer Chana, Ajay Rana "An Effective Approach to build optimal t-way Interaction Test Suites over Cloud using Particle Swarm Optimization", Third International Conference, CNC 2012, Chennai, India, February 24-25, 2012, published by Springer in Lecture Notes of the Institute for Computer Sciences,Social Informatics and Telecommunications Engineering Volume 108, 2012, pp 193-198.

iv. *Chapter 4* Proposed Framework of Testing As a Service: This chapter presents testing as a service framework based on Apache Hadoop MapReduce that generates test data automatically and efficiently and considers the QoS requirements of both the resource providers and resource consumers. Job profiling and prediction techniques have been utilized for forecasting the resources needed for different applications. A security model has also been designed for the safety assurance of data uploaded by the clients. In addition, transparent pricing model has also been formulated for the mutual benefits of clients and service providers. Furthermore, it presents the mathematical model, system design and architecture of the framework. Finally, the chapter concludes with the job execution flow of the proposed framework. This chapter is derived from:

- 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). [Impact factor = 0.434]
- Priyanka Chawla, Inderveer Chana, Ajay Rana. "Cloud-based Automatic Test data generation Framework"[J]. JCSS 82(5): 712-738 (2016). [Impact Factor = 1.307]
- Priyanka Chawla, Inderveer Chana, Ajay Rana. "The Framework for Cloud-based Intelligent Software Test data generation Service"[J]. SPE,2016. [Impact Factor = 0.897][Under Review]

v. *Chapter 5* Verification and Validation of the Proposed Framework: This chapter focuses on the verification and validation of the proposed framework. The framework has been verified by conducting experiments on local Apache Hadoop cluster and cluster of AWS EC2 machines. The experimental results have been analyzed from various perspectives such as fault detection capability, branch coverage, resources utilized, time taken, etc. Statistical and practical differences between proposed algorithm and existing algorithms have been assessed with a two-tailed Mann-Whitney U-test. The magnitude of improvement in the proposed algorithm h as been quantified with the Vargha and *Delaney*$'s$ standardized effect size $\hat{A}_{12}$. Vargha and *Delaney*$'s$ $\hat{A}_{12}$ statistics is a standardized effect size that acts as an objective and standardized measure of the magnitude of observed effect. The accuracy of the proposed framework has been assessed with respect to various perspectives used in the existing frameworks such as fault detection capability, branch coverage, resources utilized, time taken etc. The results obtained have been compared with other related cloud-based testing models such as YETI, Cloud9, HadoopUnit etc. Chapter 5 derives from:

   - 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). [Impact factor = 0.434]
   - Priyanka Chawla, Inderveer Chana, Ajay Rana. "Cloud-based Automatic Test data generation Framework"[J]. JCSS 82(5): 712-738 (2016). [Impact Factor = 1.307]
   - Priyanka Chawla, Inderveer Chana, Ajay Rana. "The Framework for Cloud-based Intelligent Software Test data generation Service"[J]. SPE,2016. [Impact Factor = 0.897][Under Review]
   - *Chapter 6* Conclusion and Future Directions: This chapter concludes the thesis and discusses the future scope of the work.

1.9 Thesis Contribution

This thesis contributes in the following ways:

i. An exhaustive study of cloud-based testing models with special focus on cloud application development and testing platforms is carried out along with their qualitative comparison w.r.t parameters like adopted strategy, implementation model etc.

ii. Test data generation strategies have been extensively explored and compared on the basis of parameters like test adequacy criteria, performance criterion etc. A comprehensive study of soft computing strategies such as genetic algorithm and particle swarm optimization is carried out.

iii. A novel test data generation strategy in single node environment for object oriented programs has been proposed and its empirical analysis and comparison with existing hybrid and traditional strategies has been done.

iv. A Hadoop MapReduce based automated test data generation strategy using genetic and particle swarm optimization has been proposed that uses the concept of pareto-optimality to devise and implement a new approach for gbest evaluation. A parallelization of proposed strategy using hadoop mapreduce has helped in attaining cost-effective test data generation solutions.

v. A framework of testing as a service has been proposed that can generate test data efficiently and henceforth achieves better increase in coverage and fault detection capability.

vi. A mathematical pricing model has been proposed that can fulfill the expectations of the customers and maximizes the net profit of service providers. Cloud service request model has also been designed that postulates Service Level Agreements (SLA) between customers and service providers.

vii. The test data generation effectiveness has been measured by the metrics like percentage of coverage achieved and percentage of faults detected. Further, the effectiveness of proposed framework has been assessed by cost and time requirements by executing it on the cloud environment. The proposed framework has been compared with other existing cloud-based testing frameworks. The results obtained have showed that the proposed framework outperforms the other existing frameworks with respect to various selected parameters.

viii. Statistical and practical differences between the proposed strategy and other existing strategies have been computed using two-tailed Mann-Whitney U-test. In order to assess the magnitude of improvements in the proposed strategy, Vargha and *Delaney's* $\hat{A}_{12}$

statistics has been used as standardized effect size in conjunction with Null Hypothesis Significance Testing (NHST).

2

Literature Review

Software Testing is a challenging activity for large software engineering projects. The amount of tests cases can range from a few hundred to several thousands, requiring significant computing resources and lengthy execution times. Cloud computing has the potential to provide unlimited storage and computing resources that can help in executing large test suites in a lesser amount of time at reduced cost.

Software testing can be safely moved to the cloud as it does not comprise of susceptible business data and has insignificant impact on the organization's accustomed proceedings. Cloud-based testing provides the opportunity of testing applications in different configurations in much lesser execution time without any need of installation and maintenance of test environment.

This chapter describes the concept of cloud testing and its various perspectives. It reports systematic review of cloud-based testing techniques published in major software engineering journals and conferences. The existing work on cloud-based testing has been categorized based upon the issues addressed and testing strategies adopted by them. In addition, comparative study of the state-of-art approaches of cloud-based testing models has been done. Empirical analysis of the existing cloud-based testing frameworks and their limitations have been laid out. Further, description of existing automatic test data generation strategies along with the merits of soft computing based automated test data generation strategy have been chalked down. Finally, the concluding section outlines the objectives of the thesis.

2.1 Cloud Testing

Cloud testing is a form of software testing in which cloud computing infrastructure is used to simulate real-world user traffic. It also refers to testing of resources such as hardware, software, etc. that are available on demand. The prominent features of cloud computing such as flexibility, scalability and reduced costs motivate the organizations to adopt it. It helps the organizations to acquire the required tools, software licenses and infrastructures at a very low cost without having to set it up themselves and bothering about its maximum utilization. Cloud testing alienates the approach of cloud and SaaS but has the capability to mime the real-world user traffic to carry out load and stress testing. It has completely transformed the way software testing is carried out and caters end-to-end solutions for software testing. As test servers can be provisioned very easily on demand by cloud service providers, it has effectively curtailed the cost and time requirements of software testing.

Cloud Testing can be viewed from three different angles:

i. Testing at different levels e.g. performance testing, security testing etc. of online SaaS or non-SaaS software. This has been referred as cloud-based testing in this work.

ii. Testing infrastructure and platforms across different deployment models of the cloud i.e. public, community, private or hybrid clouds.

iii. Testing of the cloud itself.

Cloud environment should be tested and measured for their performance, availability, security and scalability in order to support an efficient delivery of services [177] [181]. A majority of research papers have focused on cloud-based testing. The literature survey on cloud-based testing has been viewed from two angles as shown below:

- Cloud-based testing and issues.
- Offerings by Potential Providers.

Table 2.1 shows the distribution of work w.r.t the above stated angles. Based on Table 2.1, 21 papers discussed benefits, needs, issues and several factors to adopt cloud-based testing. Only 2 papers illustrated steps, techniques and methodologies to migrate testing to cloud and there are 15 papers on the approaches adopted by different vendors of cloud-based testing.

After discussion of cloud testing and its various perspectives, next section presents the differences between traditional software testing and cloud testing.

2.1.1 Traditional Software Testing Versus Cloud Testing

Traditional software testing cannot be applied to test applications in a cloud environment as traditional software testing is designed for on-premise single-tenant applications and cannot support multi-tenant applications. It does not support new business requirements and risks that come with cloud environment. Test engineers that are trained to perform traditional software testing need special training to perform testing in cloud.

New business needs and associated challenges with traditional software testing should be properly understood before migrating to cloud environment in order to meet cloud-testing requirements. Organizations need to be equipped with additional infrastructure such as different testing skills required by test engineers in order to perform the job of testing in a cloud [47] [182]. To identify the type of testing to be performed, it requires an understanding of cloud characteristics and the risks/challenges involved. An appropriate testing strategy should be selected by keeping an eye on the following challenges of cloud computing:

i. Quality risks of cloud computing such as reliability, flexibility, multi-tenancy, self-healing, pricing band on SLA's and location independence.

ii. Inherited risks associated with cloud computing like data governance, data security, virtualization, security, reliability, Monitoring and Manageability.

iii. Applicable cloud models to be tested like Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS) etc.

Cloud Testing exemplifies testing on demand and is perceived as future of testing services. The following types of testing are performed in general for cloud testing:

i. System Integration Testing / User Acceptance Testing: The cloud platform must be integrated with all the platforms and infrastructure services so that a user can build up his data online.

ii. Interoperability Testing /Compatibility Testing: Interoperability refers to moving cloud applications from one infrastructure to another cloud infrastructure. Any application on a cloud must have the ability to be operated on multiple platforms and environments. It should be able to get executed across any cloud platform.

iii. Performance Testing / Load Testing: Elasticity refers to using minimum resources and producing maximum usage for end users. The performance of cloud should remain intact even if there are increasing inflows of requests. The speed or effectiveness of the system should not be compromised.

iv. Stress Testing / Recovery Testing: In case a failure occurs, disaster recovery time should be as less as possible. Services must be retrieved online with minimum adverse effects on client's business.

v. Security Testing: Unauthorized access to data must be strictly prohibited. Shared data integrity and security must be maintained at all times, as client trusts the cloud platform for securing his information. The infrastructure requirement for test environment is another important consideration for cloud testing. The two possible options for choosing an adequate test environment are:

- Simulating in-house cloud test environment.
- Selecting an adequate cloud service provider.

Apart from identifying applicable testing types, testing team must also focus on the specific requirements of the application to be tested as enumerated below:

i. Supporting multiple browsers.

ii. User session management related issues.

iii. Test against security vulnerabilities.

iv. In a multi-tenant environment, restricting users to access their data only

v. Test engineer's skill.

This section highlights the disparity existing between traditional software testing and cloud-based testing. The next section elucidates the state-of-the art cloud-based testing models.

2.2 Cloud-based Testing Models: State of the Art

Several authors have proposed models for cloud-based testing. A comparative analysis of these techniques is presented in Table 2.2 and is also explained below:

- Virtualization-aware Automated Testing Service (VATS):- VATS is a framework that facilitates automated performance testing and configuration of services in Cloud computing environments. It executes tests, manipulates virtualized infrastructure, and collects performance information. VATS complements a Service Lifecycle Management system named SLiM. SLiM is a model-driven service for managing the configuration, deployment, and run-time management of services operating in Clouds. VATS works with SLiM and supports the testing of other services that are compatible with SLiM. VATS uses HP LoadRunner as a load generator and provides the foundation for an automatic performance evaluator for Cloud environments.

- York Extensible Testing Infrastructure (YETI):- YETI is a cloud enabled automated random testing tool with the ability to test programs written in different programming languages [90]. While YETI is one of the fastest random testing tools with over a million method calls per minute on fast code, testing large programs or slow code such as libraries uses memory intensively. It relies on the Hadoop package and it does map/reduce implementation to distribute tasks over potentially many computers. Cloud version of YETI can be distributed over Amazon's Elastic Compute Cloud (EC2).

- Model-Based Testing Using Symbolic Execution:- Symbolic execution [87] is a fully automatic technique for generating test cases to achieve high testing coverage. It is performed by executing programs with symbolic, rather than concrete inputs. The paths followed during symbolic execution form a symbolic execution tree, representing all the possible executions through the program. However, exploring all the possible program executions is generally infeasible, thus restricts the application of symbolic execution in practice. Scalability of symbolic execution can be addressed through parallelization as done in Cloud9 [88]. Cloud9 is an automated testing platform that employs parallelization to scale symbolic execution by harnessing the resources of commodity clusters. Cloud9 helps cope with path explosion. It can automatically test real systems, such as memcached, that interact in complex ways with their environment. Doing so without Cloud9 is hard, because single computers with enough CPU and memory to symbolically execute large systems either do not exist today or are prohibitively expensive. Besides single-threaded single node systems, Cloud9 also handles multi-threaded and distributed softwares. Cloud9 provides an easy-to-use API for writing "symbolic tests". Developers concisely specify the families of inputs and environment behaviors to test the target software without having any need to understand the working of symbolic execution.

- Model Based Testing Service using Labeled State Transition Systems (LSTSs) :- It is a model-based GUI testing service for Symbian S60. The server encapsulates the domain-specific test models and associated test generation heuristics. The testers, or test execution specialists, order tests from the server, and the test adapter clients connect to the phone targets under test. It is easy to deploy in industrial environments and tasks of the tester are minimized to specify the coverage requirement [205].

- D-Cloud:- Takayuki Banzai et al. [85] proposed a software testing environment for dependable parallel and distributed system using cloud computing technology, named D-Cloud. D-Cloud uses Eucalyptus as cloud management software and enables fault tolerance testing by causing device faults by virtual machine. The authors presented the concept and design of D-Cloud and described the procedure to specify test scenarios. D-cloud automates the system configuration and test procedures. It can also perform a number of test cases simultaneously and emulates the hardware faults flexibly. Furthermore, the preliminary test example depicting the software testing using D-Cloud has

been presented. The result shows that D-Cloud allows to set up the environment easily and effectively test the software for distributed system.

- AST :- It is based on the concepts of autonomic computing to test adaptive systems which is called as Autonomic Self-Testing (AST). It deploys test managers throughout the software to validate dynamic adaptations and updates. AST is designed with flexible strategies for incorporating the approach into systems with different performance and availability requirements. It supports replication with validation strategy that can provide a highly transparent runtime testing process in distributed environments. AST is supplemented with TSaaS that allows testing to cross administrative boundaries in the Cloud [213].

- Combinatorial Testing Strategies:- M.I Younis et al. [275] presented a new deterministic strategy called MultiCore Modified Input Parameter Order Generalized (MCMIPOG) based on earlier strategy, Input Parameter Order Generalized (IPOG). MCMIPOG is a parallel t-way test generation strategy for multicore systems and it adopted a novel approach by removing control and data dependency to permit the harnessing of multicore systems. Experiments were undertaken and they proved that MCMIPOG outperforms most existing strategies (IPOG, IPOF, IPOF2, IPOG-D, ICH, TConfig, Jenny and TVG) in terms of test size and execution time. MCMIPOG is capable of supporting higher interaction strength of t>6. Authors also investigated (GMIPOG) [276], the Grid based strategy for generating t-way test set in which they redesigned and distributed the test generation process (called GMIPOG) on the GRID. Their practical results were encouraging particularly in terms execution time, while keeping the optimized test set size. Experimental results demonstrated that GMIPOG scales well against the sequential strategy IPOG with an increase in the number of computational nodes.

- IBM Development and Test Cloud:- IBM Smart Business Development and Test Cloud facilitates evaluation, planning, designing and implementation of a dynamically scalable, virtualized development and testing private cloud environment. A cloud model enables users to access IT resources for development or testing. Unlike traditional methods, this model can dramatically reduce many manual steps and processes required to set up a production-like development and testing environment. With an on-demand provisioning of virtual and physical resources, it helps in reducing capital investments while offering the elastic scalability to handle fluctuating business needs. It also provides an automatic provisioning and configuration to meet challenging development and testing cycle times. Moreover, IT labor costs and configuration errors are also reduced. It facilitates a quicker return on investment without placing the mission-critical applications at risk.

- PREFAIL :- It is a programmable failure injection tool that supports failure abstractions and executions profiles that helps testers to write policies to prune down large spaces of multiple-failure combinations. It facilitates an automatic sorting of failed experiments

depending upon the bugs that caused them and parallelization of test workflow for further speedup. PreFail has been integrated to three Cloud software systems like HDFS, Cassandra and Zookeeper [277].

- FATE and DESTINI:- It is a testing framework which has been integrated with several Cloud system like HDFS, for Cloud recovery which consists of different modules: FATE (Failure Testing Service) and DESTINI (Declarative Testing Specifications). FATE facilitates systematic multiple failure testing of recovery, whereas DESTINI specifies the way to recover from failures [278].

- Genetic Algorithm based Test data generation:- Linda Di et al. [135] employed traditional genetic algorithm using Hadoop Map Reduce to generate test data. The authors conducted preliminary analysis of the proposed algorithm to evaluate the speed-up with respect to the sequential execution. The experimental results depicted the saving of 50% of time when using parallel version of genetic algorithm with Hadoop Map Reduce.

- HadoopUnit:- In [105], authors designed HadoopUnit which performs unit testing using JUnit for distributed execution framework. It uses MapReduce primitives to distribute test cases execution over the cloud. The necessary files such as testing tool and test cases are uploaded to the DFS prior to its utilization by a Mapper node. Mapper nodes are instructed by a Master for execution of test cases and Reducer collects the report and stores it in the distributed file system. An elementary case study [105] done with HadoopUnit on a 150-node cluster showed speedup of 30x in execution time. These applications are basically focused on execution of already generated test data over multiple nodes and substantially lack the user support in term of automatic designing of intelligent test cases.

After a brief introduction to the existing cloud-based testing models, the subsequent section presents various corporate cloud-based testing providers.

2.2.1 Corporate Providers of Cloud-based Testing

The companies providing cloud testing are listed below:

(i) IBM :- IBM Company's Smart Business Development and Test service, offered by the "IBM Cloud", is aimed at developers who use IBM's Rational toolset. IBM's CloudBurst for Development and Test appliance can be used for on-premises testing. IBM also offers its Infrastructure Optimization Services such as IBM Smart Business Test Cloud that provides on-demand secure, dynamic and scalable virtual test server resources in a private test environment [279].

(ii) SauceLabs :- SauceLabs provides onDemand which is a software testing service based on Selenium that enables web applications to be tested across multiple browsers in the cloud. This Silicon Valley startup lets developers test Web apps using the Selenium test suites, with an emphasis on speeding up the process. Selenium is a platform for testing Web apps written in a variety of programming languages. SauceLabs, is hosted on Amazon Web Services and it allows customers to record video of browser actions to troubleshoot and document tests [197].

(iii) Skytap :- Skytap promotes a hybrid cloud model where customers use its testing services in combination with their own on-premises tools and processes. Features include a library of operating systems, databases, and other softwares for creating drag-and-drop test environments. Skytap along with Hewlett-Packard offers software testing through HP's Quality Center [198].

(iv) uTest :- uTest uses a community of professional testers, i.e. crowd-sourcing, to help developers identify software bugs and other issues. The types of testing include functional testing, load, performance, and usability testing. Customers can view backgrounds and profiles of 18,000 on-demand testers and grade the quality of their work [198].

(v) VMLogix :- VMLogix is a specialist in virtual machine management and provides a software, VMLogix LabManager, Cloud Edition that runs on Amazon's EC2. It can be used to create virtual labs for testing software stacks in the form of Amazon Machine Images, with a central repository, team capabilities, and audit trails. VMLogix also offers on-premises platform to create hybrid software lab environments [198].

(vi) Zephyr :- Zephyr provides "testing lifecycle management." Zephyr 2.5 platform provides testing desktops and dashboards served from a hosted Zephyr Management Platform, which incorporates testing applications and APIs. Their tool's capabilities include test case creation, test automation, defect tracking, collaboration, and project management [280].

(vii) SOASTA :- SOASTA CloudTest Platform offers a comprehensive set of components and access options that enable teams of any size to execute frequent, scalable and affordable performance testing of web and mobile applications from inside the lab through to live production ensuring high performance under normal as well as extreme traffic conditions [201].

(viii) Sogeti :- Sogeti provides STaaS-Software Testing as a Service tailored to provide the clients with a flexible, easily obtainable cost-effective service [178].

After presenting existing research in cloud-based testing in academia and corporate world, the following section would present the classification of existing models.

2.2.2 Classification of Cloud-based Testing Models

In this section, classification of cloud-based testing models has been done on the basis of testing technique adopted by various researchers. It has been classified into seven categories as shown in Table 2.3.

i. Model based Testing.

ii. Performance Testing.

iii. Symbolic Execution.

iv. Fault Injection Testing.

v. Random Testing.

vi. Privacy Aware Testing.

vii. Others.

As shown in Table 2.3, several researchers have worked in the area of above mentioned categories. Takayuki Banzai et al. [85] proposed a software testing environment, called D-Cloud using cloud computing technology and virtual machines with fault injection facility.
Sebastian Gaisbauer et al. [86] implemented VATS which uses HP LoadRunner as a load generator and provides the foundation for an automatic performance evaluator for cloud environments. Tariq M. King et al. [213] proposed a collaborative framework approach which combined the development of an automated test harness for a cloud service, with the delivery of test support as-a-service (TSaaS). Marco A. S. Netto et al. [207] uses a virtualized environment for load generation aimed at performance testing. Both Liviu Ciortea et al. [88] and Stefan Bucur et al. [87] contributed towards the development of Cloud9, the first symbolic execution engine that scales to large clusters of machines. Stefan Bucur et al. [87] extended the research of Liviu Ciortea by including a new symbolic environment model that supports all major aspects of the POSIX interface, such as processes, threads, synchronization, networking, IPC, and file I/O. Alexey Lastovetsky [217] worked towards parallel testing of computer software systems. Sasa Misailovit'c et al. [216] presented novel algorithms built on the Korat algorithm for constraint-based generation of structurally complex test inputs. W. K. Chan [179] proposed a metamorphic approach to online services testing. Manuel Oriol [90] developed YETI, a language agnostic random testing tool. Gary Wassermann [215] proposed automated input test generation algorithm which is a standard concolic framework for automated test execution in cloud computing environments. Beatriz Martin [212] investigated the testing challenges of future web applications and proposed a testing methodology by integrating search based testing, model-based

testing, oracle learning, concurrency testing, combinatorial testing, regression testing and coverage analysis. There are 2 whitepapers by CSS in which they discussed performance engineering of web applications and performance testing tools provided by the company.

The next section highlights the gaps of existing cloud-based testing models.

2.3 Challenges of Existing Cloud-based Testing Models

Several studies have been conducted that anticipate the tremendous increase in utilization of infrastructure as a service over internet. The study also indicated an increase in demand of cloud in the upcoming years [113] [114]. Software testing is one of the safest workload that can be put onto the cloud as it does not involve any business sensitive data [111]. Inspite of the above facts, very little work has been reported on cloud-based software testing frameworks.

Based on the literature survey, challenges of cloud-based testing are as follows:

(i) An efficient test data generation strategy with the capability to reveal maximum faults in minimum time: In academics, most of the researchers have worked on parallel execution of test data. There is comparatively lesser work in the field of automatic test data generation and most of the test data is designed manually. Quality of manually designed test case depends upon the expertise of test case designer. In the corporate world of software testing tools, automated test script execution is achieved with the help of capture and playback mode feature of the tools. This again depends on the human intelligence and their involvement and hence software testing is not completely automated. Also, the selection of test cases is decided by software tester and success of testing process depends upon the expertise and intelligence of a tester. This can be fully automated if test data is generated automatically by the software testing tool. Thus, an efficient automatic test data generation is one of the challenging areas of research in software testing.

(ii) Improper resource utilization: It is very important factor as cost of the testing job mainly depends upon the resources used. Existing cloud-based testing frameworks require clients to opt for the resources for their testing jobs. But, this is very tedious on the behalf of clients to decide about the best configuration of required computing resources to perform a particular testing job. Hence, it results in higher cost.

(iii) Lack of trust among clients: Existing models does not emphasize on the trust establishment. There should be some security mechanism that ensures the protection of data and code uploaded by different users.

(iv) Lack of transparent pricing model: Transparent pricing model increases the trust level and popularity of services among clients. But, in the existing cloud-based testing frameworks, this issue has not been addressed effectively. Transparent pricing model helps

in fulfilling the expectations of customers and also helps to maximize the net profit of service providers.

(v) Service Level Agreements (SLA): Terms and conditions of cloud services are often hard to understand, misleading and biased towards the vendor.

The gaps analysed above necessitate devising a framework for cloud-based testing with strong focus on QoS parameters such as fault detection capability, resource utilization, security and cost.

The next section highlights the various existing test data generation strategies.

2.4 Test Data Generation Strategies

Software organizations spend considerable portion of their budget in testing related activities. A well tested software system will be validated by the customer before acceptance. The effectiveness of this verification and validation process depends upon the number of errors found and rectified before releasing the system. This in turn depends upon the quality of generated test cases. Over the years, a number of different methods have been proposed for generating test cases. According to IEEE standards, a test case is "a documentation specifying inputs, predicted results, and a set of execution conditions for a test item". As the definition states, there are three required elements in a test case; test inputs or test data, predicted results and conditions of execution [44].

In order to generate test cases, test data must first be generated. Test data can be generated using different sources that describe the Software Under Test (SUT)and it's behavior or how it will be used. Test case generation techniques are usually named after the source of test cases such as specification-based or model-based.

Several approaches have been proposed for test case generation, mainly random, path-oriented, goal-oriented and intelligent approaches [51]. Random techniques determine test cases based on assumptions concerning fault distribution. Path-oriented techniques generally use control flow information to identify set of paths to be covered and generate the appropriate test cases for these paths. These techniques can further be classified as static and dynamic. Static techniques are often based on symbolic execution, whereas dynamic techniques obtain the necessary data by executing the program under test. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch, irrespective of the path taken. Intelligent techniques of automated test case generation rely on complex computations to identify test cases [51].

Some of the test case generation methods have been used extensively by various researchers and they are explained in the following sections.

2.4.1 Mutation Testing

Mutation Testing is a fault-based testing technique that provides a testing criterion called the "mutation adequacy score" [52]. The mutation score is ratio of the number of detected faults over the total number of seeded faults. The mutation adequacy score can be used to measure the effectiveness of a test set in terms of its ability to detect faults. The general principle underlying Mutation Testing work is that the faults used by Mutation Testing represent the mistakes that programmers often make. By carefully choosing the location and type of mutant, test adequacy criteria can be simulated. Such faults are deliberately seeded into the original program, by simple syntactic changes, to create a set of faulty programs called mutants, each containing a different syntactic change. To assess the quality of a given test set, these mutants are executed against the input test set. If the result of running a mutant is different from the result of running the original program for any test case in the input test set, the seeded fault denoted by mutant is detected.

Mutation Testing can be used for testing software at unit level, integration level and specification level. It has been applied to many programming languages as a white box unit test technique, for example, Fortran programs, Ada programs, C programs, Java programs, C sharp programs, SQL code and AspectJ programs [52]. Mutation Testing has also been used for integration testing [54] [55]. Besides using Mutation Testing at the software implementation level, it has also been applied at the design level to test the specifications or models of a program. For example, Mutation Testing has been applied to Finite State Machines [56] [57] [58] State charts [59], Estelle Specifications [60], Petri Nets [61], Network protocols [62], Security Policies [63] and Web Services [64].

Sasa Misailovic et al. [216] have presented two significant improvements of Korat: (i) a set of algorithms for parallel testing, and (ii) a methodology for generating fewer equivalent inputs. Korat [65]is a constraint-based test generation tool that performs a systematic search to generate all valid test inputs (within the bounds). Their algorithms for parallel testing splits test generation and execution across a number of worker machines, both off-line (when the inputs are saved on disk) and online (when execution immediately follows generation). They have also described a novel methodology for reducing the number of equivalent inputs that Korat generates. These improvements have been motivated by testing an application developed at Google. The experimental results on running Korat in parallel across up to 1024 computers on the Google's infrastructure show that test generation and execution can achieve significant speedup up to 543.55 times.

2.4.2 Specification Based Test Case Generation

Gustavo Cabral et al. [69] explored the usage of formal methods with the purpose of documenting system and consequently enabling the generation of test cases and other artifacts. The use

of formal methods is commonly related to the definition of a specification that can be refined and eventually mapped to code, thus guaranteeing the quality of the implementation. They focused on generating formal specifications through validation and processing of requirements at an early stage. They proposed a Controlled Natural Language (CNL), a subset of English, used to write use case specifications according to a template. From these use cases, a complete strategy and tools enabled the generation of algebraic formal models in the CSP notation. They defined templates that represent requirements at different levels of abstraction, capturing different views of the system behavior. Moreover, a refinement notion is defined to connect the generated CSP models through an event mapping relation between abstract and concrete models. The use of a Controlled Natural Language (CNL), and use case templates guaranteed requirements consistency.

M.V. Arkhipova et al. [70] presented an automatic method, named SemaTESK1, for generation of test sets for a translator front end. It implements specification-based testing approach for static semantics checker testing. The SemaTESK method provides the SRL language for writing formal specifications of static semantics in the form that yields efficient traceability and is suitable for semantics-directed automated generation of tests. The SemaTESK method is supplied by the corresponding test case generator STG that allows generating test sets that meet appropriate completeness criteria formulated on the basis of SRL specifications of a language under test. The STG generator takes SRL specifications as an input and automatically produces both semantically correct and semantically incorrect (with unambiguously stated kind of an incorrectness) tests. The SemaTESK method has been used in several case studies including testing static semantics checker of such a complex programming languages such as C and Java.

2.4.3 Model-Based Testing

Clementine Nebut et al. [71] proposed an approach for automating the generation of system test scenarios from use cases in the context of object-oriented embedded software and taking into account the traceability problems between high-level views and concrete test case execution. They concentrated on the use of Unified Modeling Language (UML) to support an object-oriented development process. The method developed by them is based on a use case model unraveling the many ambiguities of the requirements written in natural language. They build UML use cases enhanced with contracts as per their definition. Based on these formalized-but still high-level-requirements, they defined a simulation model of the use cases. The simulation model is also used to explicitly build a model of all the valid sequences of use cases, and extract relevant paths from it using coverage criteria. These paths are called test objectives. The test objectives generation from the use cases constitutes the first phase of their approach. The second phase aims at generating test scenarios from these test objectives. In standard development processes based on the UML, each use case should be documented with several sequence diagrams. Building on these existing sequence diagrams, they automated the test

scenarios generation by replacing each use case with a sequence diagram that is compatible in terms of static contract matching.

Ashalatha Nayak et al. [72] addressed the problem of generating test scenarios from UML activity diagrams. Their approach captured all control structures of an activity diagram and transformed them into a testable model called ITM. Representing the structure of an activity diagram with ITM has a number of advantages. The proposed approach provides a foundation and reference for managing any complex and arbitrary activity diagrams. The introduction of separate control flow graph for each control construct leads to a simple process of analysis and derivation of scenarios using coverage criteria. The impact of path explosion has been tackled by choosing representative path for a set of paths in the context of looping. Their approach does not demand any modification in the input design and hence it is suitable for any unified process development environment. Moreover, their approach is capable of managing activity diagrams of any complexity for generating test cases efficiently. They considered the activity diagram generated from use cases to represent system behavior.

Antti Jaaskelainen et al. [73] described an approach that is model-based graphical user interface (GUI) testing of smartphone applications for synthesizing test models from test cases. In addition, they presented the results of two small case studies where the approach was applied for creating test models from existing test cases in the domain of S60 GUI testing. The synthesis is semi-automatic and thus requires user interaction to achieve useful results. A tool supporting this interaction was also sketched. Their approach is domain-specific in the sense that the set of keywords and the corresponding weight values must be decided based on the domain knowledge. Their approach is semi-automatic requiring user interaction.

Matthias Riebisch et al. [74] introduced an approach for generating system-level test cases based on use case models and refined by state diagrams. These models are transformed into usage models to describe both system behavior and usage. The method is intended for an integration into an iterative software development process model. The resulting test cases are suited to be carried out in conventional ways, i.e., either manually or using test tools. The method is supported by a XML based tool for model transformation.

2.4.4 Soft Computing based Test Data Generation Strategies

Soft Computing can be defined as a 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 [139]. In other words, soft computing is a set of problem-resolution techniques regulated by approximate reasoning, optimization approximation methods, evolutionary and search methods. Furthermore, its goal is to exploit the tolerance for imprecision, uncertainty,

approximate reasoning and partial truth in order to achieve tractability, robustness and low-cost solutions.

Robust and fault free software development is the main target of any software development company. To achieve this, software companies invest a 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 customers after deployment of software products on the client side. This leads to huge expenditures and complications while correcting faults in the software. Hence, it is very necessary for the software development companies to identify and rectify faults in the software at early stages to reduce the overall software development cost. Test data is used to verify and validate the software, which is very important and often decides the success of software testing. Test data should be designed very carefully and intelligently and should not only cover the entire path but should also reveal faults in the code. Manual test case design consumes a lot of time and entails high level of expertise and experience of a software tester. Many times testers are not able to design the appropriate test cases which causes ineffective software testing and often leads to increase in the cost of software when bugs are found at later stages. The need of automatic software test data generation is rapidly increasing in software industries.

Metaheuristic search techniques (MHS) and Swarm Intelligence are amongst the most important constituent of soft computing. They are also termed as Search-based Software Testing (SBST) and have been extensively used to automate the process of generating test cases, thus providing solutions for a more cost-effective testing process [76] [77] [82] [239]. SBST attempts to solve a variety of software engineering problems by reformulating them as search problems and are extensively used to find optimal or near optimal solutions to problems that have large complex search spaces [78] [79] [80] [120] [281] [240]. The generation of test cases is reformulated as a search that aims at finding the required or optimal set of test cases from the space of all possible test cases. Mantere and Alander [81] discusses the use of MHS algorithms for software testing in general and McMinn [42]provides a survey of some of the MHS algorithms that have been used for test data generation. The most common MHS algorithms that have been employed for search-based software testing are evolutionary algorithms, simulated annealing, hill climbing, ant colony optimization, and particle swarm optimization [127] [253] [106]. Because of the non-linear nature of software resulting from control structures such as if-statements and loops, simple search strategies may not be sufficient and global MHS algorithms become a necessity as they implement global search and are less likely to be trapped into local optima [250] [251] [252] [253] [254] [255] [256]. Furthermore, software test data generation is an undecidable problem and 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 algorithms [262] [263] [264] [265] [266] [267]. Hence, automatic test data generation using the concepts of soft computing techniques (genetic and particle swarm optimization) has been proposed in this thesis.

2.4.4.1 Genetic Algorithm

Genetic Algorithms (GA) is an evolved metaheuristic optimization technique that works with a set of promising solutions tracking the optimization problem through encoding them on some data structures, called chromosomes. This technique copies principles of Darwian theory of biological evolution as it applies recombination and mutation operators on these chromosomes yielding one or many candidate solutions. GA throughout the solution process manages and maintains a set of candidate solutions. Initially, a set of chromosome is randomly generated and in next successive steps and a selection operator is employed choosing two best solutions from the current set. The selection is directed by the results given by fitness function used as quantifiers. Then, the crossover operator is applied 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 [250]

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

2.4.4.2 Basic Particle Swarm Optimization

Particle Swarm Optimization utilizes swarm-based meta-heuristic search technique given by Eberhart and Kennedy in 1995 [253]. It mimics social norms of bird flocking and animals herding. Similar to 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. The particle can be observed as an object which consists of position and velocity as two components. As particle moves, the position and velocity of particle is continually updated with every repetition of algorithm. Every particle keeps the record of particle's personal best position in the swarm and global best position. In the initialization phase, each particle possesses random position and velocity within the domain of the specific function. In the process of applying the evaluating function, particle modifies its velocity to be

placed 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

$$\begin{aligned} V_{j,d}(t) = wV_{j,d}(t-1) + cr_{j,d}(pBest_{j,d}(t-1) - X_{j,d}(t-1)) \\ + cr_{j,d}(lBest_{j,d}(t-1) - X_{j,d}(t-1)) \end{aligned} \tag{2.1}$$

$$X_{j,d}(t) = X_{j,d}(t-1) + V_{j,d}(t) \tag{2.2}$$

A Basic algorithm for PSO is as follows [253]

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
```

2.4.4.3 Existing GA and PSO based Strategies

The techniques based on GA and PSO that generate automatic test data for object oriented programs and structured programs have been illustrated in this section.

Zhang et al. [247] combined the ideas of GA and PSO for automatic generation of test data. The authors selected two population sets P1 and P2 and applied genetic operators and PSO calculation operators separately. Computation of fitness function values is carried out on population P1 and particles are sorted based upon their fitness function values in population P1. The best fitness particles are added to population P2 at ratio phi. PSO operations are applied to population P2 and updated fitness function of particles is computed. The particle is transferred to P2 and removed from P1 and new random particles are added to P1. The author tooks Triangle Classification problem as a case study and demonstrated four paths as test goal of his experiment. They compared GA, PSO and GA-PSO independently on each trail 100 times and observed that GA-PSO took lesser number of iterations than GA and PSO. The average execution time of GA-PSO is found to be more than PSO but lesser than GA. This technique generates tests that may be redundant and is more complicated as it applies GA after applying PSO and requires several transformations and exchanges in the different population sets. Moreover, the test adequacy criterion chosen is path coverage which consumes a lot of computational cost and time.

Li et al. [244] also suggested the test data generation strategy based upon GA and PSO where they used velocity and distance updates of PSO, instead of mutation operator of genetic algorithm. The chromosome representation used by them is a binary vector that divides the population into 'n' population set based upon the length of the binary vector. Singla et al. [245] proposed test data generation strategy based upon GA and PSO in which authors used velocity and distance updates of PSO to enhance the particles and genetic operations. MATLAB has been used for the implementation. It is somewhat similar to the approach proposed by Zhang et al. [247]. The fitness function is based upon 'definition computational' use and 'definition predicate use'. They tested their strategy only on some simple programs.
Kaur et al. [249] also proposed hybrid strategy based on genetic and particle swarm optimization algorithm in which authors used only the mutation operator of genetic algorithm to update the values of particles to solve prioritization problem in regression testing. The authors strategy depends on randomly selected mutant that caused larger execution time to find optimal solutions. Also, the algorithm has been tested on less number of simple programs. Wu et al. [246] suggested GAPSO hybrid strategy to overcome the optimization problem of both continuous and discrete parameters. The authors have applied genetic algorithm for discrete magnitude and PSO for continuous magnitude. They tested their approach by taking a quadratic equation and proved the effectiveness of their approach. Chen et al. [248] amalgamated PSO with genetic operator and tested the new approach for multimodal functions taken from the Black-Box Optimization Benchmarking (BBOB) functions [255]. The authors used crossover operations to the pbest particles to make it more explorative. They experimentally proved that this scheme is much more efficient than dividing the swarm into multi-swarm systems which periodically regroup in terms of being more exploitative and explorative. Chen et al. [248] modified the velocity and distance update equations by adding crossover component to it. This crossover

step sets the position of every particle to the midpoint of its pbest and a random pbest is drawn without replacement from the population of pbest particles.

Wapler et al. [259] proposed the approach of evolutionary class testing. Their approach is based on genetic programming system and utilized ECJ [260] to implement this approach. They also suggested fitness function and termed it as distance metrics in which they gave equal weightage to approach level and branch distance.

Windisch et al. [261] conducted experiments with 25 small artificial test objects and 13 complex industrial test objects and proved that PSO is more efficient and effective than GA. Wegner et al. [140] compared evolutionary testing and random testing for six C programs and found mean coverage achieved by evolutionary testing much better than Random Testing. They also concluded that although test data generated by RT is much higher than Evolutionary Testing but coverage achieved is not satisfactory.

Tonella et al. [117] also tested container classes using conventional genetic algorithm but made some changes to the original version of genetic algorithm to take care of the object oriented software testing. Authors applied one-point crossover and in addition defined three types of mutation operators that were selected randomly. The authors tested it for container classes. Wang et al. [262] also compared Hill Climbing, GA and MA and showed that MA is the best algorithm. They performed experiments for procedural functions. Arcuri et al. [263] proposed a metaheuristic based upon genetic algorithm and hill climbing known as Memetic algorithm. Authors applied hill climbing technique on each generation to find local optimum. They tested this strategy on container classes and empirically proved their strategy better than random search, hill climbing, simulated annealing and strategy based upon conventional genetic algorithm. Nayak et al. [264] adopted MATLAB to simulate their strategy for test data generation using data flow testing and found PSO outperformed GA in def-use coverage percentage. They tested on small 14 FORTRAN programs and concluded that PSO required less number of generations to achieve the same def-use coverage percentage. Ahmed et al. [265] adopted t-way interaction testing using PSO and compared their tool with other strategies like IPOG, WHITCH, Tenny, TConfig and TVG and observed remarkable test suite minimization. They carried out 20 independent test suite runs and proved that PSTG produced test data of optimal size. Li et al. [266] employed PSO for all-path automatic generation of test data. They performed the experimental results for benchmark Triangle problem & Binary Search program and did analysis on a number of iterations and time consumption and showed that their strategy is much efficient. Fraser et al. [267] employed GA and took branch coverage as test criterion initially and later it was generalized for any rest criterion. They developed their tool in Java & performed byte code instrumentation. To evaluate the strategy, one industrial project and five open source libraries had been chosen and observed. They compared whole test suite generation with single branch test case generation and observed small test suites generated. They repeated

their experiments 100 times with different seeds. Gong et al. [268] proposed automatic generation of test data for both path coverage and fault detection using genetic algorithm. Authors tested their strategy for real world programs and also compared with random and evolutionary optimization techniques. The search-based approaches employed for automated testing are very extensive. There are several metaheuristic search techniques like hill climbing, simulated annealing and evolutionary algorithms that have proved their elementary significance in generating test data automatically. Table 2.4 presents the comparison chart of the existing test data generation strategies.

After reviewing the existing test data generation strategies, the next section highlights the merits of soft computing based test data generation strategy.

2.5 Merits of Soft Computing based Test Data Generation Strategy

Based on the comprehensive literature survey, it has been discovered that there is dire need of a test data generation strategy that detects faults present in the software in minimum time and cost. PSO and GA are the types of soft computing technologies that are very effective for software test data generation [138] [241] [261] [140] [116] [171]. In addition, these algorithms are competent enough for the parallelization and thus can be used in cloud environment.

Extensive work has been found that blends genetic and particle swarm optimization algorithm to solve various research problems such as optimization of recurrent network design, optimization of multi-modal functions, Class responsibility assignment problem in object oriented analysis and design, economic dispatch problem, optimal tuning of the controller parameter, Document Clustering, force method-based simultaneous analysis and design problems for frame structures etc. [256] [257] [258]. Although, various researchers have advocated the hybrid approach and showed the effectiveness in the area of software test data generation by using simpler benchmark functions, still its application is limited in the software industry [138] [261] [140] [141] [142] [143] [144] because of the complexity and huge computational cost associated with these techniques.

Based on the findings of available literature, subsequent section formulates the problem and states the objectives for this research work.

2.6 Problem Formulation

This literature survey necessitates the migration of software testing to cloud because of various reasons like paradigm shift in the provision and use of computing services, reduction in cost

of software development (as there would be no need to install and maintain test environment), shorter development cycles, flexibility, on-demand basis and access to global markets for both providers and customers.

Cloud computing can provide testing infrastructure with quality attributes (like availability, reliability, security, performance, scalability and elasticity).

This necessitates development of a cloud-based testing framework that can provide on-demand operation and testing support. In this work, a framework would be developed for testing as a service using cloud computing. The research work in this thesis is focused on the design and development of cloud-based testing framework using soft computing methodology.

The main objectives of the proposed work are mentioned below:

(i) To study and analyze existing software testing techniques in cloud environment.

(ii) To design an automated test case generation technique for software application to be tested.

(iii) To design and develop a framework for software testing to be provided as a cloud service.

(iv) To test and validate the above framework in a cloud environment.

2.7 Conclusion

This section exhibits the disparity existing between traditional software testing and cloud-based testing. After a brief introduction, classification and gaps of existing cloud-based testing models, it illustrates existing test data generation strategies. Furthermore, the merits of soft computing based test data generation strategies are detailed. Finally, the problem has been formulated for this research work.

The next chapter presents proposed sequential and cloud-based software test data generation strategies.

Table 2.1: Cloud-based Testing

Category	Author	Year	Issue Addressed
Cloud-based Testing and issues	Somenath Nag [180]	2011	OnDemand Testing
	Leah Muthoni Riungu et al. [181]	2010	Online Software Testing
	Jerry Gao et al. [176]	2011	Cloud-based Application Testing
	IBM [162]	2011	IBM Testing Services for Cloud-Application Virtualization
	Sunil Joglekar [164]	2009	Agile Testing
	Sidharth Ghag [182]	2011	Testing on Azure
	Leah Muthoni Riungu et al. [177]	2010	Research Issues of Cloud Computing imposed on Software Testing
	Ali Babar et al. [183]	2011	engineering aspects of migrating SOA system
	K. Priyadarsini et al. [184]	2011	Architecture of Testing Service
	AppLabs [165]	2009	Benefits of Cloud Computing in the area of Testing
	Leo van der Aalst [166]	2010	Providor's Service Models and Process Models
	Impetus [49]	2011	Cloud utilization during Testing and its challenges
	Perfecto mobile [167]	2011	Cloud-based mobile testing
	Ganesh Moorthy [185]	2010	SOA testing automation framework
	Rajesh Mansharamani [186]	2008	Virtual Production Enviornment
	Ewald Roodenrijs [187]	2010	Cloud Testing benefits and challenges
	Jinesh Varia [188]	2010	Migration steps to Cloud
	CSC [168]	2011	Testing methodology adopted by CSC
	Tauhida Parveen et al. [175]	2010	Migration of Software Testing to Cloud
	Cognizant [48]	2011	Factors affecting Testing in the cloud
	Impetus [49]	2011	Cloud Computing in the product development life cycle
	TUI [50]	2011	Cloud Testing
Offerings by Potential Providors	Siemens [189]	2011	SiTEMPPO
	CSS [191]	2009	ClouTestGo
	Wipro [192]	2010	Assurance Services on the cloud
	HP [193]	2010	Accelerated Test Service, Performance Test Service, Application Security Testing and SAP Test Service
	Veracode [194]	2010	Cloud-based application security service
	ITKO [195]	2011	LISA DevTest CloudManager
	CloudTesting [142]	2011	Browser based testing
	PushToTest [143]	2011	Selenium based Functional Testing, Load testing
	IBM [196]	2010	OnDemand onPremises Testing
	SauceLabs [197]	2011	Selenium based onDemand Testing
	SkyTap [198]	2011	Development, Unit, Integration, System, User Acceptance and Performance Testing
	uTest [199]	2011	OnDemand load testing, security testing, functionality testing
	VMLogix [200]	2011	Desktop, Dashboards Testing, test creation, defect tracking, test automation
	SOASTA [201]	2011	CloudTest, Performance Testing of web and mobile application
	Sogeti [178]	2011	CloudTest, STaaS
	Rackspace [202]	2011	Test and Dev on Cloud

Table 2.2: Comparison Chart of Existing Cloud-based Testing Models

Testing Model	Strategy	Implementation	Parallelization Framework	Platform	Benefits	SUT
Yeti	Automated Random Testing	Java	Apache Hadoop MapReduce	Cloud/Amazon EC2	Test Execution Speedup	Java.lang
VATS	Performance Testing using HPLoadRunner	SmartFrog	Xen	Cloud	Improved Service Performance	SAP/R3 System
D-Cloud	Fault Injection	XML	QEMU and Eucalyptus	Cloud	Cost and Time	Parallel/ Distributed
Cloud-9	Symbolic Execution	Symbolic Tests API	-	Eucalyptus and Amazon EC2	on-demand Software Testing Service; Speedup	UNIX Utilities
GridUnit	Regression testing	-	-	Grid	Speedup	JUnit Test Case Execution
Joshua	Regression testing	-	-	Jini	Speedup	JUnit Test Case Execution
Korat	Constraint-based Testing	Java	Google MapReduce	–	Object graph visualization	Google Applications
NMS	Performance Testing	–	—	–	Less expensive and more scalable implementation	Simulation of large Scale Networks
TaaS	Prototype of TaaS Over Cloud	–	–	Cloud	Elastic resource infrastructure; provides various kind of testing services to users	Web Applications
HadoopUnit	Regression Testing		Apache Hadoop MapReduce	Hadoop Cluster	Speedup in Test Execution	JUnit Test Cases
Linda et.al.	Unit Testing	Java	Apache Hadoop MapReduce	Hadoop Cluster	Speedup	One open source library

Table 2.3: Testing Models

Approach	Author	Year	Issues Addressed
Model Based Testing	Hien Le et al [203]	2011	Model Testing for component based development
	W.K. Chan et al [204]	2009	Model based testing
	Antti Jaaskelainen et al [205]	2008	Model based Test generation algorithms
Performance Testing	Sebastian Gaisbauer [86]	2008	Automatic performance Testing;VATS
	CSS [206]	2009	On-demand performance testing tool
	Marco A. S. Netto [207]	2011	Virtualized enviornment for load generation aimed at performance testing
	Zohar Ganon [208]	2009	large-scale performance tests of Network Management Systems
	CSS [209]	2008	Performance Engineering
Symbolic Execution	Stefan Bucur [87]	2011	Symbolic Execution ; Cloud9
	Liviu Ciortea [88]	2009	Symbolic Execution ; Cloud9
Fault Injection Testing	Takayuki Banzai [85]	2010	Software Testing Enviornment based using fault injection;D-Cloud
Random Testing	Manuel Oriol [90]	2010	YETI;Language Agnostic random testing tool
Privacy Aware Testing	Lin Gu [210]	2009	Privacy Protected Testing
	Philipp Zech [211]	2011	Security Testing
Others	Beatriz Martin [212]	2011	Integration of search based testing, model-based testing, oracle testing, concurrency testing, combinatorial testing, regression testing and coverage analysis
	W. K. Chan [179]	2010	Metamorphic approach to online service testing
	Tariq M. King [213]	2010	Virtual Test Enviornment;AST and TSaaS
	T. Vengattaraman [214]	2010	Cloud computing Model for software testing
	Gary Wassermann [215]	2008	Concolic framework for automated test execution
	Sa.a Misailovic [216]	2007	parallel testing of code on the Korat algorithm
	Alexey Lastovetsky [217]	2004	Design and Implementation of Parallel Testing
	Linda et. al [135]	2012	MapReduce and Genetic Algorithm Test data Generation
	Parveen et al. [105]	2011	HadoopUnit

Table 2.4: The Comparison Table of Existing Test Data Generation Strategy

Sr. No.	Strategy	Technology Used	Test adequacy criteria	Performance Criterion	Case Study	Input Format
1	Windisch et al. [66]	PSO	Branch coverage	Fitness function evaluations Number of iterations	25 small artificial test objects and 13 complex industrial test objects	Instrumented source program in MATLAB GEATbx
2	Wegner et al. [67]	Evolutionary Testing	Mean coverage	Mean number of test data	Six C programs	Source program in C
3	Li et al. [266]	Hybrid GA and PSO	Path Coverage	Execution time	Triangle benchmark problems	C source code
4	Singla et al. [84]	Hybrid GA and PSO	dcu and dpu	Number of Test Cases and cover ratio percentage	Simple Programs	MATLAB
5	Zhang et al. [92]	GA and PSO	Path coverage	Number of iterations and average execution time	Triangle Classification problem	Instrumented source program in C
6	Kaur et al. [110]	Hybrid GA and PSO	Fault coverage	Execution time and APFD	Simple Programs	JAVA
7	Wapler et al. [259]	Genetic Algorithm	Distance metrics	Branch Coverage %	seven java classes	ECJ
8	eToc [117]	GA	Branch Coverage	Test cases, number of fitness evaluation and time	Container classes	Instrumented source code
9	Arcuri et al. [105]	GA and Hill Climbing	Branch Coverage	Time and coverage %	Container classes	Java; Instrumented code
10	Nayak et al. [264]	GA and PSO	dcu%	dcu%	Small 14 FORTRAN programs	Instrumented code; MATLAB
11	Ahmed et al. [91]	Combinatorial Testing using PSO	interaction elements	Test size	IPOG, WHITCH, Tenny, TConfig and TVG	
12	Li et al. [244]	PSO	Path coverage	Iteration time; Time consumption	Benchmark Triangle problem & Binary Search program	Instrumented code in C
13	Fraser et al. [267]	GA	Branch Coverage	Branch Coverage	Industrial project and five open source libraries	Bytecode;Java
14	GONG et al. [108]	GA	Path coverage	Fault detection and coverage	bubble sort program Siemens suite, print_tokens, schedule, replace, tcas,space, flex	Instrumented program in C
15	Traditional GA	GA	Branch Coverage	Branch Coverage	Container classes	Instrumented Source code in Java

3

Proposed Cloud-based Test Data Generation Technique

Cloud-based testing allows testing of applications with appropriate resources so as to obtain the finest quality results. It has been surveyed in the previous chapter that automatic test data generation in cloud computing environment has not been addressed to a large extent. To provide a solution to cloud-based test data generation, a soft computing based test data generation strategy has been proposed and designed in this chapter.

This chapter focuses on the design of software test data generation techniques for single machine and multiple machines in cloud environment by using the hybrid strategy of genetic and particle swarm optimization algorithms. Apache Hadoop MapReduce has been used for parallel implementation in the cloud environment. The concept of pareto front and pareto ranks has also been utilized to generate pareto-optimal solutions that satisfy multi-objective functions (fault detection, coverage and test case execution time).

This chapter initially presents the brief introduction of the test data generation and follows with the sequential software test data generation strategy. Further, it explicates the mathematical model and illustrates the design of the underlying test data generation strategy. Subsequently, it describes the complexity analysis and presents the differences between the single node and cloud-based test data generation strategies.

3.1 Software Test Data Generation

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.

The concept of pareto-optimality improves the quality of generated test data by identifying the solutions that satisfy multiple objectives. The fitness function evaluations use large population to obtain optimal and better solutions. It requires an enormous amount of computational power, time and storage space. This requirement can be fulfilled by using cloud computing that 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 contrary, MapReduce handles tedious tasks such as fault tolerance, load balancing and coordination of the execution of parallel tasks in the distributed environment. It is capable of distributing the task of test cases generation to several cheaper nodes available on the cloud easily and very effectively. Keeping this view in mind, a MapReduce based test data generation strategy has been designed in this work.

The next section describes sequential version of the proposed test data generation strategy.

3.2 PSOG : Sequential PSO and GA based Test Data Generation Strategy

In this section, the architecture designed to adapt PSO and GA for the automatic generation of test suites for the unit testing of classes of given Class under Test (CUT) has been explained. The model is shown in the Fig. 3.1. The model has been divided into four main modules as follows:

i. Instrumentor

ii. Optimizer

iii. Test Executor

iv. JUnit File Generator

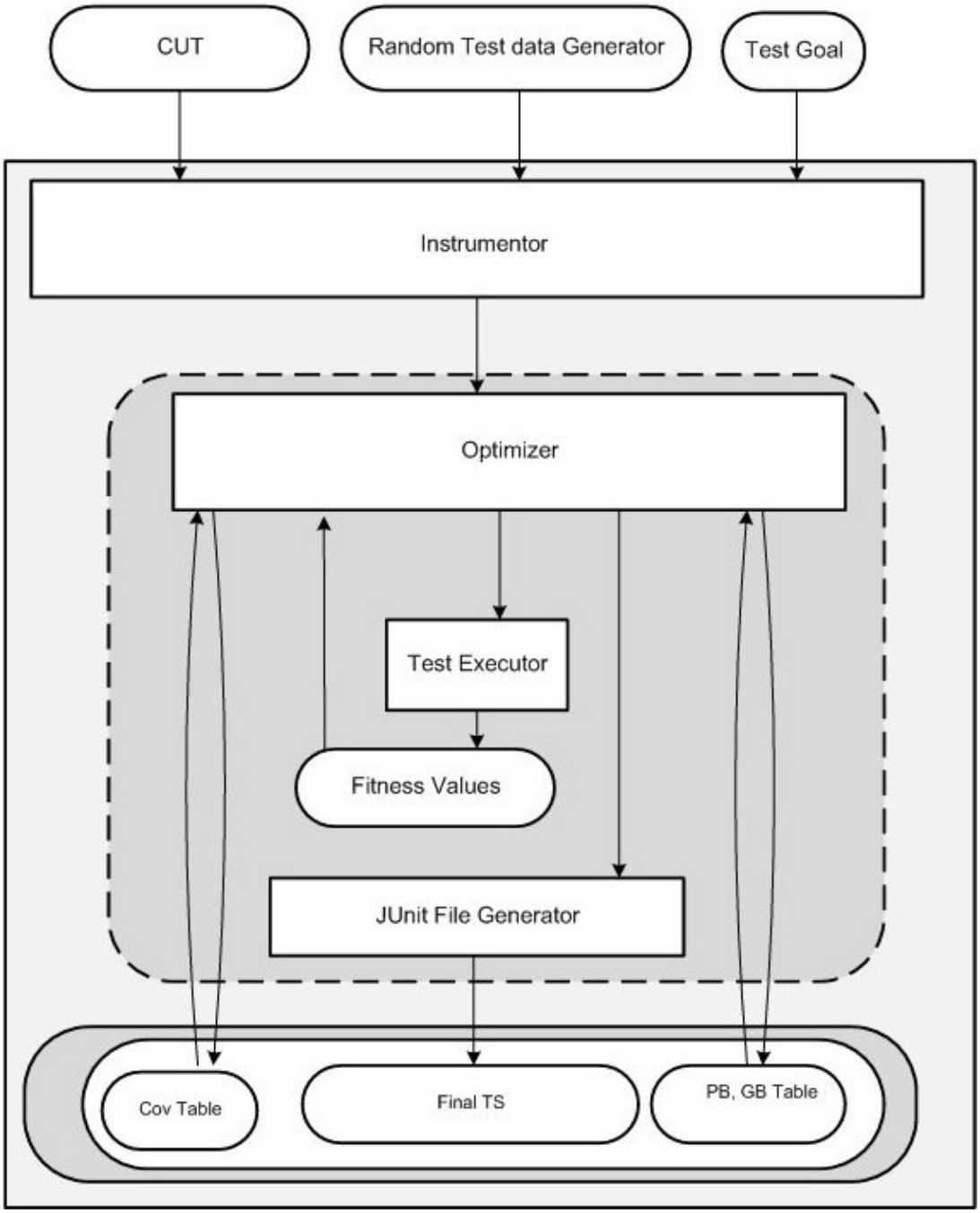

Figure 3.1: Model of Proposed Strategy

The proposed algorithm generates optimal test data for both the procedural programs and object oriented programs. The structure of a test case for both the programs consists of input data, parameter values of methods and the assertions which verify the pass and failure status of a test. Fig. 3.2 shows the format of the test case of procedural program containing only set of parameter values of the procedure call. The test case format of object oriented program also consists of sequence of constructor and method calls in addition to the above mentioned components. In object oriented paradigm, there may be multiple constructor invocations for object creation to test the class under test. In addition to this, the objects are required to be

put into special states to carry out the test scenario in the desired order. This is done with the aid of method calls on the required test objects. Fig. 3.3 represents the structure of test case of object oriented program containing two constructor invocations and parameter value of the method. The first constructor invocation corresponds to instance of the class under test and another constructor is required as a parameter for the creation of object under test. There is also an integer value required for method call.

```
float procedure( int x, float y)
{
    ...                                  Test data = ( 5, 4.02)
}
```

Figure 3.2: Test case format of procedural program

```
class F { ... }
class U
{
    ...
    // method under test

     public int add(F o1, int x)
          {
              if( o1.m1( ) == 0 )
                 {                       Test Data = (o2,o1,2)
                                         where o2 is object of
                                         class U
                     ...

                  }
           }
 }
```

Figure 3.3: Test case format of object oriented program

3.2.1 Instrumentor

The instrumentation of CUT is essential for the evaluation of fitness function. The fitness function used in this work is based on branch coverage per unit of time and branch distance function. The role of instrumentation is to modify the original executable image by inserting little instructions before every block to indicate that the block was executed. To evident the branch distances, extra instruction is added before every predicate. These instructions are responsible for the computation of branch distance and notification to the the testing environment. The

instrumentation can be carried out during runtime for java open source libraries. Jacaco open source tool has been used to analyze the execution flow of the source code of CUT. Jacaco performs code coverage analysis using standard technology in Java VM based environment [83]. It provides lightweight, flexible and well documented library for integration with various build and development tools. Jacaco can perform an efficient on-the-fly instrumentation and analysis of applications even in the absence of source code.

3.2.2 Optimizer

It is main component of the model which is responsible for all the operations related to Test data generation. Implementation has been carried out using the intelligence of genetic and particle swarm optimization algorithms. The best particle positions of particle swarm optimization and genetic operators like mutation and crossover have been retained to evolve better solutions. Genetic operators have been used for the modification of individuals in between the generations rather than using position and velocity update rules of particle swarm optimization. This is because these operators are more explorative. It also accommodates the object oriented structure of container classes as it contains not only parameter values but also contains constructor and method invocations. This has helped to achieve the increased population diversity of genetic algorithm and fast convergence rate of particle swarm optimization by selecting only the best individuals as the new candidate solution in the upcoming generation. The loop is iterated until maximum number of iterations have been carried out or the optimal solution is found with the best fitness values and coverage table has been formed as per the test goal received from the user. Optimizer also instructs JUnit Test data generator module to create JUnit file consisting of best individuals formed by the proposed algorithm. Algorithm 3 gives the macro steps of the proposed strategy. Particle representation and its corresponding test case format lead to a set of JUnit test cases for a given Class Under Test (CUT) as shown in the Fig. 3.4.

3.2.3 Detailed Explanation of PSOG

At the first step, type of program is taken as input from the tester and the appropriate random test data is generated. After initialization, instrumentation of Class under Test (CUT) is done where each condition is transformed into an expression that performs the same operations as the original program but also conveys the value taken by the reached condition. Jacaco is used as the instrumentor and it has been explained in the previous section. The set of test goals are also determined and saved in a variable TG. The algorithm is iterated for each and every identified TG until it is carried out for maxIterations. The maxIterations denotes maximum allowable test data generation time and has been set by the tester. The randomly generated test cases are executed on the CUT with an intention to cover all the test goals. The fitness function is computed for each and every particle (i.e. test case). The pbest variables store the particles

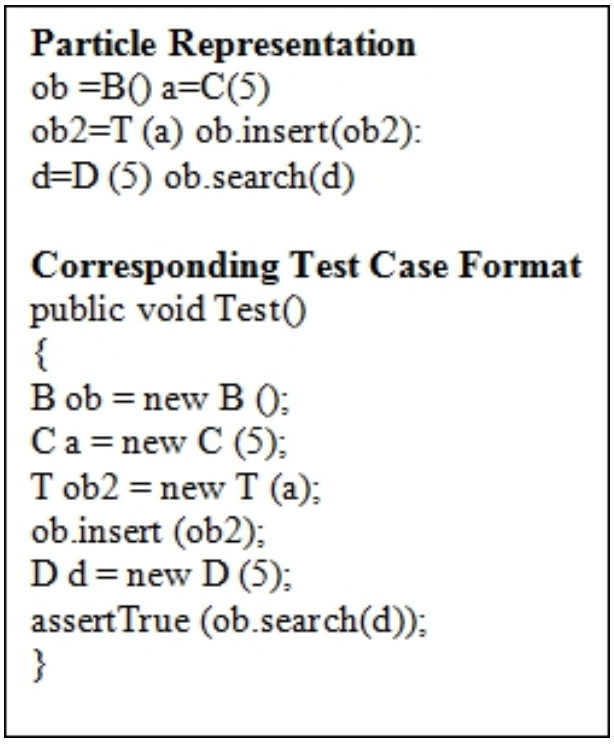

Figure 3.4: The template of Particle and Test Case Format

which have best fitness value when compared with its previous best fitness function values. The cpvector contains the current particle (i.e. solution) which is compared with previous best value. The better value (having better fitness function value) is stored in the variable pbest. In the similar fashion it executes for Cgbest and gbest. The gbest variable stores the global best test case value (i.e. particle) having best fitness function value in whole population at a given iteration. All the individuals possessing better fitness function values are stored in the PBest and Gbest files. The test cases (or a particle) that succeed to cover the targets are also saved in file named as FinalTestSuite. Crossover operation is applied between the gbest and pbest particles to uncover the new population individuals. Mutation is also applied to the newly formed particle to introduce the diversity into the population. This process is repeated until all the test goals are covered or it exceeds the maximum limit of execution time of test case generation. The set of test cases saved in FinalTestSuite can be redundant which is minimized by applying simple greedy algorithm by including the test cases which cover most of the test goals. CovTable will contain all the information about covered targets. The type of crossover and mutation operators used are same as that of used in MRPSOG strategy. Algorithm 3 gives the macro steps of the PSOG. The proposed hybrid approach using PSO and GA has been implemented in Java programming language. The operative principle of the proposed approach as described above is also depicted through flowchart shown in Fig. 3.5. The objective function of test data generation optimization problem is computed by branch coverage achieved by a test case and it is analogous to the quality or fitness of the associated solution.

The fitness function chosen for this research work is branch coverage. The fitness function

ALGORITHM 3: Proposed Strategy

```
input  : Source code of the Class
output: A JUnit Class
Generate random population of 'n' particles.;
Initialize CovTable with results of random test data generation.;
Instrument the SUT;
Instrument the SUT;
Create TestGoals;
for TG ← 1 to n do
    target=TG ;
    while counter ≤ maxIterations and target not satisfied do
        Evaluate fitness of each particle in the population based on branch coverage
          analysis;
        Emit(Particle, Fitness);
        for particle ← 1 to n do
            if cpvector > pbest then
                /* Compare Current value of particle with the previous best
                   particle                                                  */
                pbest=cpvector;
                /* Set particle best with the current value of the vector
                   */
            end
            if Cgbest > gbest then
                /* Compare Current value of particle global best with the
                   previous global best                                      */
                gbest=Cgbest;
                /* Set particle global best with current value of global
                   best                                                      */
            end
        end
        If all the individuals have been processed then write best individuals in PBest and
          GBest Tables;
        Apply Crossover operation on each particle stored in PBest with GBest;
        Apply mutation over Gbest;
        if its fitness value greater than GBest set it as new GBest;
        Execute the instrumented code for each particle to check if there any other untested
          branches being tested;
        Update CovTable, TesGoal, TestSuite;
    end
end
```

is the ratio of edges covered during the execution of a test case to the total number of edges that leads to the given test goal or target in a control flow graph. Thus, the fitness value would be close to 1 for the test cases which traverse most of the control edges, while it will be close to zero when the execution path does not coincide with the the edges leading to the test goal or target. Approximation level and branch distance are also appended to the fitness function in order to guide the test cases (or particles) towards the better optimal solution set.

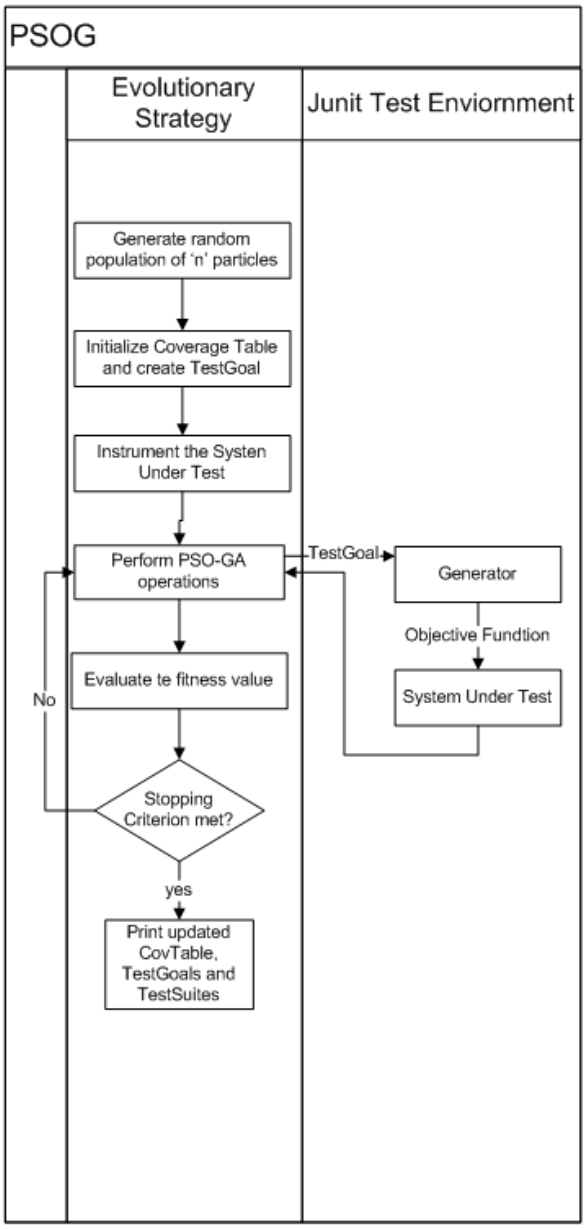

Figure 3.5: Flowchart of the Proposed Strategy

3.2.4 Fitness Function

Search based algorithms like genetic algorithms, particle swarm optimization depend highly on the fitness function to find out the optimal solution set. Multi-objective problem has been formulated in which branch coverage and computational cost have been chosen as its constituent objectives. This necessitates to maximize branch coverage whereas cost should be minimized. The quality of branch coverage of a particle (or test case) is measured in terms of approximation level and branch distance.

$$BC = A + normalize(B) \tag{3.1}$$

Where BC is Fitness function for branch coverage, A and B represents approximation level and branch distance respectively. For a given path that does not cover the target branch, the approximation level can be defined as number of branching nodes that comes between nodes covered by the individual and the target node. Branch distance defines the extent of closeness of the input that is supposed to satisfy the condition of the last predicate but went wrong. The approximation level is required to point the search towards the target branch. Branch distance directs the exploration of the search algorithm towards optimal solution of the problem. It acts on the branch predicates and can be calculated with the application of recursive rules after performing instrumentation of the class under test as explained earlier in the MRPSOG section and defined in the Table 3.1 [101]. The normalization has been achieved with the following formula [269]:

$$normalize(B) = 1 - 1.001^{-B} \tag{3.2}$$

$$F(TS_j) = Coverage(TS_j) + BC(TS_j) \tag{3.3}$$

$$TS_j = \{TC_1 \ldots TC_n\}$$

where TC_k are Test cases in a TS_j in a particular ordering; $i = 1 \ldots n$; The aim is to find a suitable ordering of the sequence of Test cases that achieves maximum coverage in minimal time. To decide if $TS_i$is better than TS_j , it can be written as

$$\left\{ \begin{array}{l} F(TS_k) > F(TS_j) \vee \\ F(TS_k) = F(TS_j) \wedge Time(TS_k) < Time(TS_j) \end{array} \right\}$$

F(TS) consists of two components: Coverage of a particular sequence which is to be maximized in minimal time and second component is the sum of approximation level and branch distance which is to minimized and is only used to guide the search. Here, the concept of Test Prioritization has been utilized as defined by Rothermel et al. [270] to prioritize the minimal test suite which covers all the branches and 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 of PT real numbers R.

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. Time taken can be computed using the method System.currentTimeMillis(). Before executing a particular test suite on the given CUT, current time has been stored in a variable startTime and after execution of the CUT current time has been stored into another variable endTime. Execution time can be found by subtracting startTime with endTime. The objective function of test data generation optimization

problem is computed by branch coverage achieved by test case and it is analogous to the quality or fitness of the associated solution.

3.2.5 Representation

The architecture used for a particle is not just a combination of parameter values rather it is a sequence of constructor and method calls with the associated parameter values. Combination of parameter values can be used for procedural programs but not for object oriented software. The representation is shown below:

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

Here 'P' represents particle, 'B' represents sequence of constructor and function instantiation, V represents actual parameters passed to the function, c represents class name and f represents function name.

3.2.6 Test Executor

Test case executor component is responsible for fitness function evaluation by executing CUT with the initial random test data generator and after that with the possible candidate produced by Optimizer. Its main goal is to report the best solution having optimum fitness function values which is then used by the optimizer to initiate JUnit File Generator for the generation of JUnit file.

After explanation of the first phase of the research work, next section explains the second stage that focuses on the design and development of cloud-based test data generation framework.

3.3 MRPSOG Algorithm: MapReduce based Test Data Generation Algorithm

The main goal of the MapReduce based Test Data Generation Algorithm is to generate test data of SUT that not only ensures maximal branch coverage but also detects maximum faults that may result in failure of software in minimum possible time. A strategy has been designed by using particle swarm optimization and genetic algorithm in which gbest and pbest particles are computed using the concept of pareto optimality approach. Pareto optimality helps in finding the optimal solution with the best fitness value of multiple objectives. Mutation and crossover is further applied to these selected best particles. Only best solutions are retained and carried over to the next generation. This helps in finding the best test data covering the entire feasible path and detects maximum faults in the software.

The process of test data generation begins with the formation of class control flow graph followed by derivation of the optimal test inputs that results into execution of the desired paths. The suitability of test inputs is decided by the fitness function and desired path can be specified by 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. 3.6.

The class control flow diagram shown in the above figure shows that upon the call of a method, the control flow graph is created which is same as that of control flow graph of a method of procedural program.

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 bytecode of the SUT is loaded on the HDFS. It extracts all the names of classes present in the API in a file on HDFS.

ii. The MapReduce job is created and MRJob and Mapper classes are set.

iii. The number of Reduce tasks is set to zero in the configuration, so that, the job becomes Map only job and Mapper does all its task with InputSplit and no job is specified for the Reducer. In this case, the number of output files is equal to number of mappers and are written on the HDFS. This has helped us in reducing the overhead associated with the

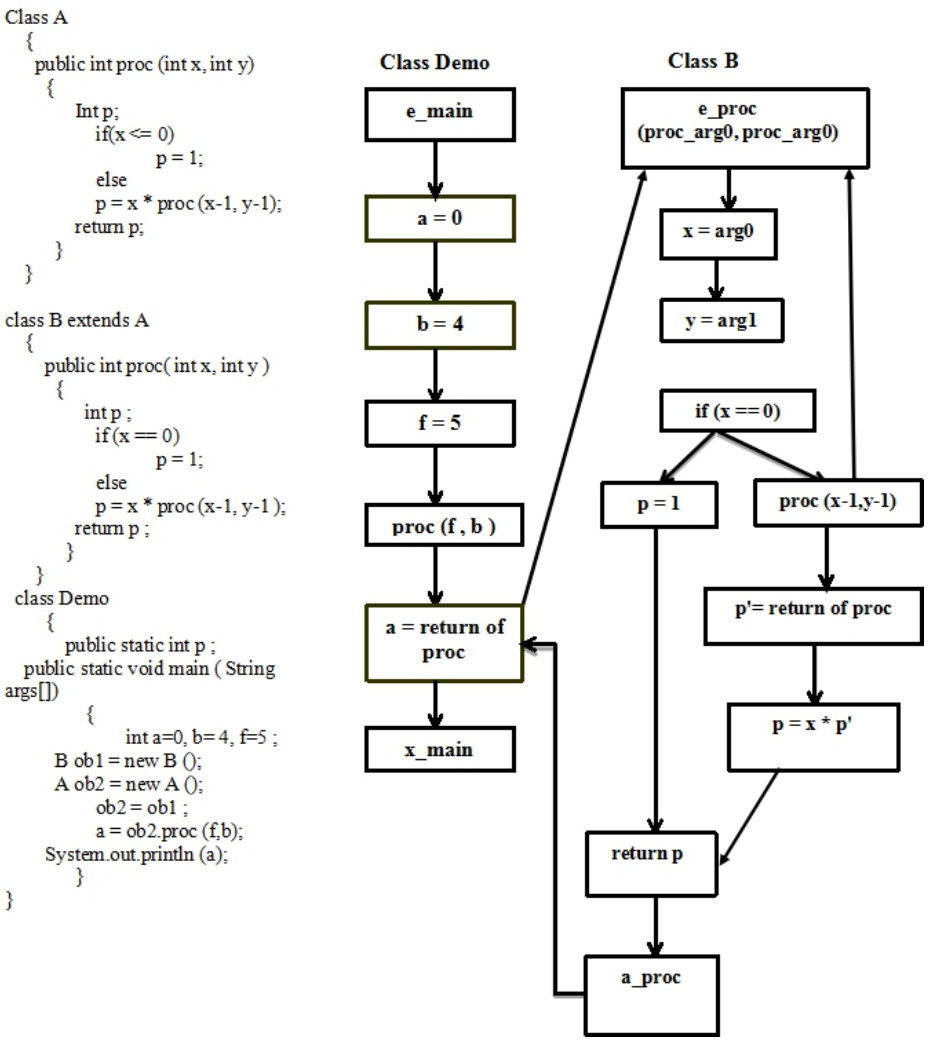

Figure 3.6: The Class Control Flow Diagram

data movements to the Reducer. All the necessary operations have been performed in parallel with the Mapper.

iv. Output Key Class and Output Value Class are initialized.

3.3.1 Mapper

Mapper 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 performs 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.

ALGORITHM 4: 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;

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 form of JUnit file.

v. The JUnit files are written to the HDFS by calling context.write() method.

ALGORITHM 5: 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);

3.3.2 Representation

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 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 a 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 have been explained in this section.

The first step to find the solution of a problem by applying soft computing technique is to encode the candidate solution. The encoding can be done with much precision with the help of illustration of the grammar. The grammar is a set of productions which defines the syntax of the candidate solution. In this 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. The same representation has been used as described in Section 3.2.5.

3.3.3 Fitness Function

The proposed strategy 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 adopted fitness function is as follows:

$$fitness = (f1, f2) \tag{3.4}$$

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

$$f_1 = |S| - |S_T| + \sum_{b_i \in B} \frac{Dmin(b_i,t)}{1 + Dmin(b_i,t)} where 0 < t \leq CT \tag{3.5}$$

First part $\rightarrow$ It deduces approach level of the test suite where S and S_T denote method sequence and traversed methods respectively.
Second Part $\rightarrow$ 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.

Branch coverage and faults detection should be maximized whereas time elapsed 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 obtain 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 between nodes covered by the individual and target node. The approach level directs the search towards the target branch whereas branch distance guides the exploration of 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 3.1 [42]. This has been implemented by making modifications to the source code of Jacaco.

Table 3.1: Branch Distance Evaluation Rules [42]

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

$$f2 = \sum_{j=1}^{|z(x)|} q_j \tag{3.6}$$

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 this work, the unchecked exceptions reported on artificial seeding of the bugs in the SUT have been termed as faults. 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 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 and 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 on its own. Error is the exception specified by class Error and its subclasses. Examples of Errors are OutOfMemoryError and java.io.IOError.

The test data of the proposed framework is generated by calling the method getBestPopulation(). It uses the concepts 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. In 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 has been executed. Though the modified code still performs the same operations as the original program but also conveys the value taken by the reached condition. The

instrumentation of the SUT is essential for evaluation of fitness function. To evident 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. Jacaco has been used as an instrumentor and it performs code coverage analysis using standard technology in Java VM based environment [83]. It can be integrated easily with several development tools because of being lightweight, flexible and having well documented library. Jacaco can perform an 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. The 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 performs 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. Three objective functions (branch coverage, faults detected and execution time) have been used 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 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. The 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 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. 3.7 and the algorithm is described in Algorithm 7 [110].

$$p_i \geq p_j = \begin{cases} if\ rank(p_i) < rank(p_j)\ or \\ (rank(p_i) = rank(p_j))\ and \\ (distance(p_i) > distance(p_j)) \end{cases} \tag{3.7}$$

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 figures 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 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 6) 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 unchecked targets.

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

xvii. The above operations are repeated until optimal solution is found or threshold of iterations is exhausted.

3.3.4 Minimization and Prioritization

The module minTestCases() performs minimization of the test cases and eliminates redundant test cases. It also updates the TestReport, FinalTS and Resource Utilization Table on HDFS. The priortizeTestCases() module prioritizes the test cases such that fault detection rate increases. To achieve this, the concept of Test Prioritization has been utilized that is defined by Rothermel [270] for the prioritization of the test suite. The prioritized test suite covers all the branches and detects maximum possible faults in the SUT. 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 is applied to any such ordering, yields an award value for that ordering.

3.3.5 GBest Evaluation

Pareto ranking methodology has been utilized 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 [134] [136]. On the contrary, it has been used to form a set of promising solutions (Gbest particles set) which not only satisfy coverage criteria but also detect 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:

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

The chosen objective functions comprises of code coverage and fault detection and they should be maximized for a given time. Time should be minimized for a given traversed path. When the optimization problem is instantiated to satisfy above three objectives, it should retain the following properties:

Suppose, the population of a subset of test suite T has been achieved 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 a set of optimal solutions rather than just a single solution

obtained through single objective that merely approximates the global optimum solution. Each member of 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 [134].

$$p_i \geq p_j = \begin{cases} if\ rank(p_i) < rank(p_j)\ or \\ (rank(p_i) = rank(p_j))\ and \\ (distance(p_i) > distance(p_j)) \end{cases} \tag{3.8}$$

Crowding distance function shown in Algorithm 8 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.

The next section computes and presents the complexity analysis of the cloud-based test data generation algorithm.

3.3.6 Complexity Analysis

The worst case time complexity of an algorithm depends upon the basic operations performed by it. The basic operations performed by proposed algorithm and the associated complexity are discussed below:

i. Fitness function evaluations: The number of fitness function evaluations (fe) depends upon population size n and number of generations (gen_i). Here , i represents the number of generations.

$$fe = n \times gen_i \tag{3.9}$$

ii. Evaluation of pareto fronts: Pareto front computations involve two loops. The first loop computes the number of particles which dominates p_i (n_p) and the set of particles which particle pi dominates (T_p). This process takes $O(mn^2)$ comparisons. Here m is the number of objectives. The other loop computes the remaining fronts by taking newly formed front as current front. Each such iteration is associated with at most $O(n)$ computations and there are at most N fronts. Hence the complexity of this loop is $O(n^2)$.

iii. Crowding distance computation: The complexity of this sub-module is same as that of sorting algorithm. In the worst case, all the solutions would be found in one front and complexity would be observed as $O(mn\log n)$.

iv. Sorting of pareto fronts: This sub-module makes a selection between two solutions. The solution which belongs to lower rank is given preference over the other. If both the solutions belong to same rank then the solution located in less dense area is given preference over the other. The worst complexity comes out to be $O(n\log n)$.

ALGORITHM 6: 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 7: genParetoFront(IP)

input : Population IP
output: Non-dominated Fronts

for $p_i \in IP$ **do**
  for $q_i \in IP$ **do**
    if $p_i \succ q_i$ **then**
      T_p=$T_p \cup \{q_i\}$
    end
    if $q_i \succ p_i$ **then**
      n_p=$n_p + 1$;
    end
    if $n_p = 0$ **then**
      PF_1=$PF_1 \cup \{p_i\}$;
    end
  end
end
c=1;
while $PF_i \neq \emptyset$ **do**
  L=$\emptyset$;
  for $p_i \in PF_i$ **do**
    for $q_i \in T_p$ **do**
      n_q=$n_q - 1$;
      if $n_q = 0$ **then**
        L=$L \cup \{q_i\}$;
      end

      $c = c + 1$

      PF_i=L;
    end
  end
end

ALGORITHM 8: 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);
$I[1]_{dis}$=$I[l]_{dis} = \infty$;
for $i = 2$**to** $(l-1)$ **do**

$[i]_{dis} = I[i]_{dis} + (I[i+1].m + I[i-1].m)$

end

v. New population generation: It computes the particle of the new generation by performing crossover and mutation operations. There can be at most N particles in the new upcoming population. Hence, the worst case complexity is $O(n)$.

Thus, the worst case complexity of the entire algorithm in a single iteration is proportional to $O(n^2)$.

The next section presents an illustrative example of cloud-based test data generation strategy.

3.4 Illustrative Example

In this section, the working of the mapper algorithm has been explained by taking an example of a population of six particles. The first step evaluates the fitness values of each particle as described in the previous section. The fitness values of each particle in the format of (%branch coverage, %fault detection, time) are as under:
P1=(60,70,3)
P2=(58,69,5)
P3=(54,65,4)
P4=(75,82,4)
P5=(70,80,3)
P6=(75,83,4)
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 3.2
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 3.7 clearly shows that distance (T4) $<$ distance (T6).
Hence, Gbest = $\{P6, P4, P5, P1, P2\}$

The next step is application of crossover operation on the elements of GBest. Subsequently, set PBest is formed by comparing the ranks of initial value of GBest and modified value

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 3.2: Non-dominated Sorts

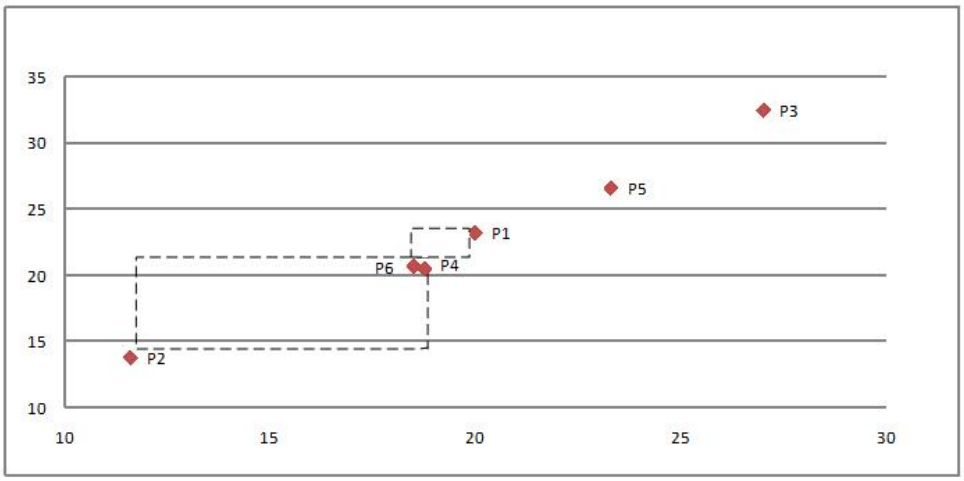

Figure 3.7: Crowding Distance

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. The type of crossover and mutation operators used in this work are explained below:

i. One-point crossover: Crossover operator 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 the help of an example in Figure 3.8.

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

- Mutation of parameter values: Parameter values of methods are changed with the another value of same type. The Figure 3.8 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 ex-

ample shown in Figure 3.8.

- 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 3.8.

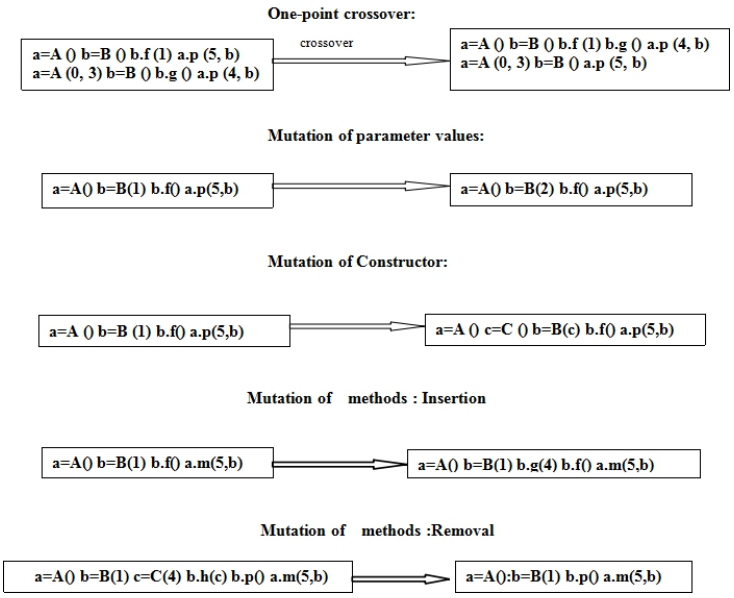

Figure 3.8: Example demonstrating Crossover and Mutation

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

The next section presents the differences between PSOG and MRPSOG.

3.5 MRPSOG versus PSOG : Modifications done in MRPSOG from PSOG

MRPSOG is the Map Reduce Version of the PSOG. In addition to parallelization of the test data generation strategy, some more features are added to it. They are listed below:

i. The concept of pareto-optimality has been used for the evaluation of GBest. It has helped in finding the best optimal solution satisfying multiple objectives such as fault detection and coverage in much lesser time.

ii. Crowding Distance is also applied for the better exploration of the solution set. The solution belonging to less crowded region is selected for the next upcoming generation set.

iii. PBest particles are computed with rank comparisons of previously generated particles. The better individual is selected and mutation is applied over it. The better evolved individual is selected otherwise older particle is retained.

iv. Crossover is applied between the GBest and PBest for the new elite generation.

v. The logic of the strategy is divided between Map and Reduce functions. Map function performs all the functions related to the particle fitness evaluation and computation of next generation particles until best optimal set is evolved. The reducer finally prioritizes the set of test cases in a sequence or in other words, it does the appropriate ordering for further increase in percentage of fault detection.

3.6 Conclusion

This chapter discussed the proposed sequential and cloud-based test data generation strategies. The representation of the test case format and the formulation of objective function for both the strategies have been discussed. The new proposed GBest evaluation method using the concept of pareto-optimality has been explained with the help of an illustrative example. Further, the differences between PSOG and MRPSOG have also been laid out. In the next chapter, cloud-based test data generation service framework has been proposed.

4

Proposed Framework of Testing As a Service

The previous chapter elaborated the design of cloud-based test data generation technique and sequential test generation algorithm. The new approach for the gbest evaluation using pareto-optimality has been explained. The differences between the proposed sequential and cloud-based test data generation strategies have also been laid out.

This chapter presents the testing as a service framework that is based on Apache Hadoop MapReduce in which user can submit the testing job and can obtain test data after completion of the job. The framework generates test data automatically and efficiently and considers the QoS requirements of both the service providers and service consumers. MapReduce job profiling and prediction techniques have been utilized to forecast the optimal resource requirements for different size of applications. The security model has also been designed for safety assurance of the data uploaded by clients. In addition, transparent pricing model has also been formulated for mutual benefits of clients and service providers and to offer testing as a service.

This chapter initially describes the design of the proposed framework. Further, it presents mathematical model, goals and features of the proposed framework. Finally, the chapter formulates the architecture and execution flow of the proposed framework.

4.1 Proposed Framework

The proposed framework is based on Apache Hadoop MapReduce in which user submits job for testing and after successful completion of MapReduce job, a JUnit file is saved on HDFS that can either be retrieved manually or can be sent to client through an e-mail. The proposed service framework focuses on automatic test data generation which is an important aspect of software testing as an optimal and intelligent test data can significantly reduce the cost as well as time of testing.

The main aim of the generated test suites is to reveal maximum faults in code in minimum possible time and cost. The framework has been implemented on Java platform that takes byte-code of Software Under Test (SUT), type of testing and goal of testing as input from the user. The underlying strategy for test data generation of the proposed framework is called by Map-only job. As explained in the previous chapter, it is designed by amalgamation of Genetic and PSO algorithm. Genetic operators have been applied for updating and evolving the particles. The reason behind this is that 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 constructor and method invocations. This also helped us to achieve the increased population diversity of Genetic Algorithm and fast convergence rate of PSO by selecting the best individuals as new candidate solution in the upcoming generation. The two component fitness function has been designed which computes the branch coverage as well as number of faults lying in the traversed paths. The optimal test data that reveals more faults in minimum time is given priority than the others.

The Gbest particle set is formed from 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 SUT and type of testing.

The proposed framework is generic enough to use 'test oracle' from the available code contracts [145]. Fault is said to occur when an exception is generated without any violations of preconditions of the contract. However, in this work all the unchecked exceptions reported on artificial seeding of the bugs in the SUT have been termed as faults.

Cloud computing services face significant obstructions to its acceptance due to lack of trust among potential users. To build and sustain trust, security techniques at various layers are implemented and role based access control is provided. The proposed security mechanism ensures protection of data and code of different users that will certainly build confidence of users for using the service.

The service providers aim to provide application access to user over the internet without other concerns of procurement of software licenses and upgrades. Minimization of costs and

maximization of customer satisfaction level should be achieved by providers to become popular among potential users. To address this challenge, mathematical pricing model has been designed to fulfill the expectations of customers and to maximize net profit of service providers. Cloud service request model has also been proposed that postulates Service Level Agreements (SLA) between customers and service providers. Another most important feature of the proposed framework is to provide and guide users with most appropriate cluster configuration parameters like number and type of VMs and MapReduce configuration parameters like number of mappers and reducers per VM. This has helped to achieve better performance in a cost-effective manner. The existing cloud-based testing frameworks depend on the customer's specifications that often leads to poor resource utilization and higher cost.

The next section explicates the mathematical model of the proposed framework.

4.1.1 Mathematical Model

A client submits his testing job for a particular SUT in the form of tuple(d_j, b_j, n_j, t_j), where d_j is the deadline to submit test report of the submitted job, b_j is the maximum budget quoted by client, n_j is the number of VMs client can afford and t_j represents the type of VMs. For simplicity, it can be assumed that clients have capability to specify their processing requirement. However, the mechanism of profiling and prediction has been used that prompts the users or clients to opt for an optimal hadoop cluster configuration. But, the final decision of cluster configuration selection is taken by clients only. This provision has been implemented in order to increase the transparency of pricing model and for building trust among users or clients. The proposed framework aims the following:

i. **Maximization of profit**: The profit of providers is computed by finding the difference between Quoted Price(QP) and Total Cost(TC) incurred . This can be represented mathematically as follows:

$$\textbf{Maximize}\,(Profit) = QP - TC \tag{4.1}$$

Here, QP represents the price quoted by the provider and TC is the total cost incurred. From the above equation, the profit of the provider can only be maximized when TC is minimized and the difference between QP and TC is as large as possible. This infers that if cost incurred to process a request is minimized without compromising the quality of service then it leads to client's cost benefits. It has been assumed that provider specified a minimum expected return and provider accepts job only if his investment return is greater or equal to minimum expected return.

ii. **Profile and Predict job's performance**: The cost of job can be reduced by minimizing the required number and types of VMs. To help the client in making this smart decision, the profiling and prediction module has been inculcated in the proposed framework. An

open source tool Starfish [112] has been used to create profiles of the job which helps in forecasting the resource needs. The proposed framework prompts the users or clients for selecting an optimal hadoop cluster configuration.

iii. **Minimization of cost to clients**: The cost to client can be minimized if provider quotes lesser price for the service as compared to other service providers. It means provider should be able to provide service at minimum cost without compromising quality of service or minimum expected return. The job will be accepted only if investment return is greater or equal to minimum investment return.

iv. **Minimization of job completion time**: Job completion time is the sum of time spent while processing the job($(Time)_R$), data transfer time ($(Time)_{DT}$), time spent to deliver test report to the client ($(Time)_D$) and waiting time spent during payment to gateway service ($(Time)_{WT}$). Hence, $(J_{CT})_j = ((Time)_R) + ((Time)_{DT}) + ((Time)_D) + ((Time)_{WT})$. The cost of job can be reduced by minimizing the job completion time:
$(J_{CT})_j$ i.e. $(J_{CT})_j < d_j$

In this section, mathematical model has been laid out, the next section discusses the features of proposed framework.

4.1.2 Features of proposed framework

The features of the 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 PSO algorithm.

iv. Easier parallel implementation through Apache Hadoop MapReduce.

v. Prompt mechanism for guiding the customers in selecting best cluster configuration to achieve minimized cost.

vi. The security mechanism that ensures protection of data and code of different users.

vii. The mathematical pricing model that fulfills the expectation of customers and maximizes net profit of service providers. Cloud service request model has also been designed which postulates Service Level Agreements (SLA) between customer and service providers.

The next section describes the architecture overview of the proposed framework.

4.2 Architecture Overview of Proposed Framework

In this section, we have described the architectural overview of the proposed framework and is illustrated in Figure 4.1. As depicted, the proposed framework is composed of the following components:

i. **Starter**: It is the front-end of the framework which takes an input SUT, test-goal, testing type, deadline, budget, number and types of VMs from the user and transforms it in the form of text file named as Test Description File (TDF). This file is stored in HDFS. It also 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.

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

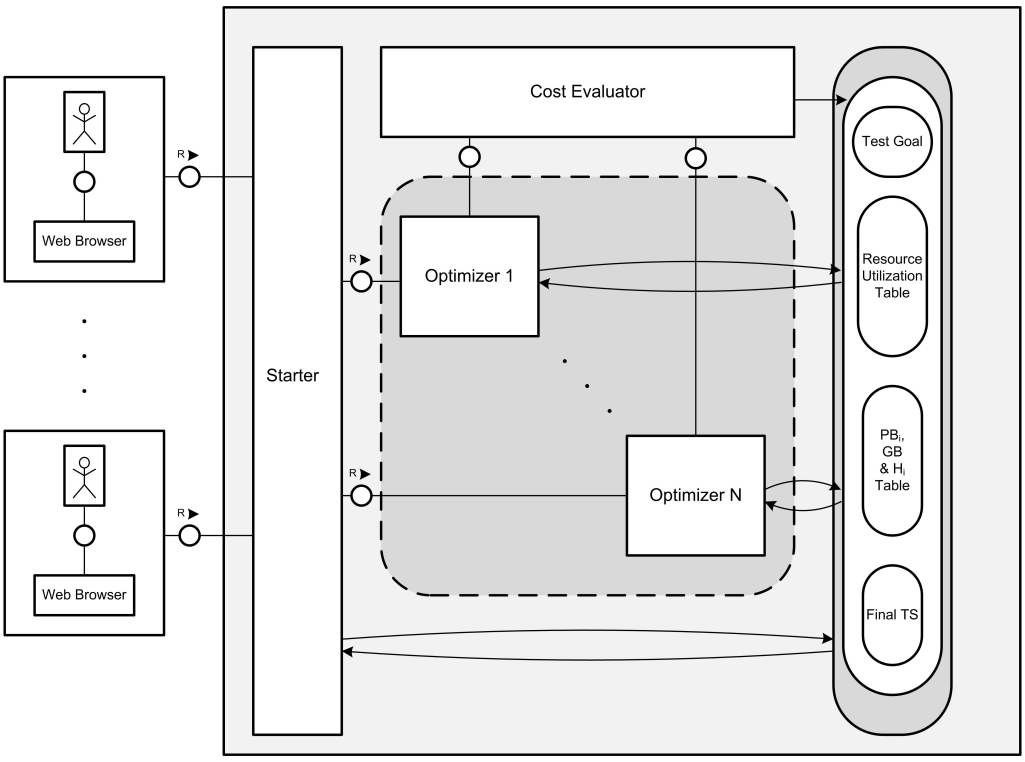

Figure 4.1: Framework of Cloud-based Testing

The proposed framework offers Testing as a Service as per the following steps. It is also depicted in Figure 4.2:

i. The user submits the test request to the Starter. The test request consists of bytecode of the SUT, type of testing, test goal etc. in the form of Test Description File (TDF). TDF is

saved on the Hadoop Distributed File System (HDFS) in the form of 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 to 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 an intelligent test data generation. It also performs the minimization and selects 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 notify about their 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.

4.2.1 Starter

It is the front end component which takes SUT, budget, time deadline etc. from the client as input. These inputs are then transformed to the form of text file named as Test Description File (TDF) which is stored on HDFS. It also invokes the MR_PSOG in Optimizer. The constituent modules are Security and Authentication, Job Profiling and Prediction, Admittance Strategy and Test Report Submission.

i. **Security and Authentication:** The security challenges of the proposed framework have been addressed by implementation of security mechanisms at manifold stages and providing role based access control for the protection assurance of code and data of clients. Role-based access control is a unified model that supports real-world access control requirements for large scale authorizations [118] [53]. Two modes of client roles have been

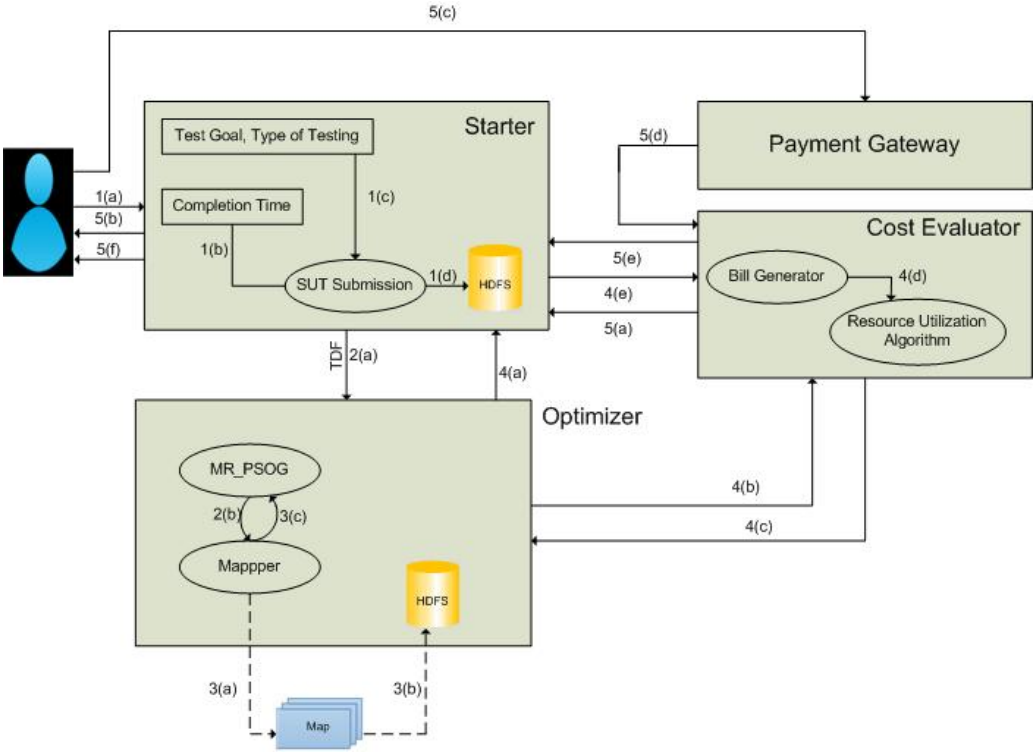

Figure 4.2: Execution Flow

defined: Owner and Reliable Associate (RA). The client who has submitted the job for test data generation is treated as Owner. The Owner may share the test data with other groups of community such as testers, developers and research groups for consultation on specific issues. Such group of people have been termed as Reliable Associate (RA). The process of creation, modification and removal of RA is named as MngRA. RA has been divided into Tester, Developer and Researchers classes based on the permissions granted to them. The Tester has been given the permissions of editing the code uploads whereas developer is allowed to view the test data and researchers are allowed to download the strategy of test data generation and view the test data. The security system has been devised by incorporating symmetric cryptosystem for authentication and role based access control for authorization. Identification of clients is carried out by their user name and password. Since cloud data security is the foremost apprehension for the customers using services provided by the service providers, so there may be lack of trust between the provider and the customers [121] [104]. Therefore, to enhance the trust relationship between the parties, Trusted Third Party security support is used to assure that only registered users are allowed to access the data [119]. Private Key delivered by Trusted Third Party has been used to encrypt the passwords of the users. The authenticated users are allowed to access the resources based upon the authorization roles associated with user id. The user possessing the role of Owner can create a number of RA and assigns different authorizations to them labeled as (RA_T, RA_D, RA_R). The single private key is being shared between the Owner and RA under him. This key can be used to encrypt and decrypt the

confidential information stored on the cloud. The model has been shown in the Figure 4.3

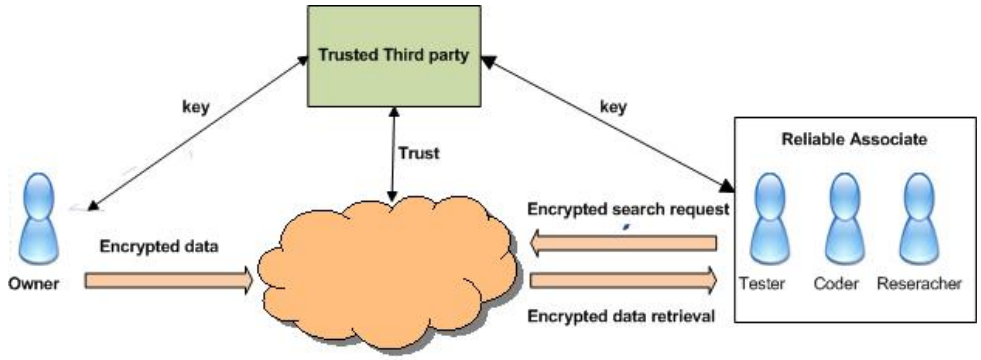

Figure 4.3: Security Model of the Proposed Framework

ii. **Job Profiling and Prediction**: It prompts the client for most appropriate hadoop and cluster configurations. This is achieved by utilizing the job analysis and prediction toolkit named as Starfish [112]. This tool leverages MapReduce profiling and create profiles for forecasting optimal resource needs to run the job over hadoop cluster. This helps user in making cost-effective decisions while selecting hadoop cluster configuration.

iii. **Admittance Strategy**: The job is submitted to the optimizer only if it is approved by the admittance module of the starter. The job is admitted only if service provider is gaining profit by executing the job. The profit is calculated with the help of pricing model explained in the upcoming section. If the job is allowed for submission, TDF is stored in HDFS and then it invokes the Optimizer.

iv. **Test Report Delivery**: The generated test suite is delivered to the user in the form of JUnit test file by the starter upon receiving the payment from the customer.

4.2.2 Optimizer

It is the main component which is responsible for computing the test data of the submitted application. JUnit file is generated comprising of optimal individuals formed by the proposed algorithm. It also evaluates the cost associated as per the pricing model explained in the section 5. The system architecture is shown in the Fig. 4.4. It consists of following two main sub-modules:

i. **MR_PSOG Algorithm**: This module manages the overall execution of MR_PSOG and has been explained in the previous chapter. It performs all the task related to initializations and once it terminates, it returns test suites for SUT.

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 prioritization of the resulting test suite. It also updates TestReport, FinalTS and Resource Utilization Table on HDFS. This has also been elaborated in detail in the previous chapter.

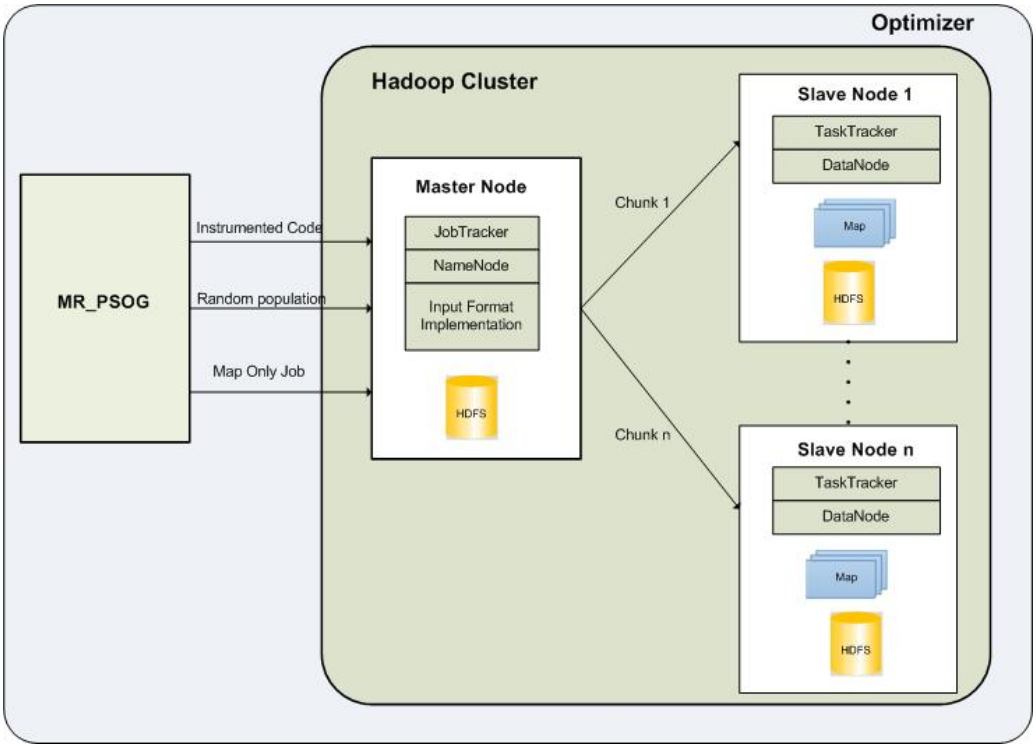

Figure 4.4: System Architecture of Optimizer

The operative principle of the proposed approach as described above is also depicted through flowchart shown in Figure 4.5.

4.2.3 Cost Evaluator

The proposed 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 SUT 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

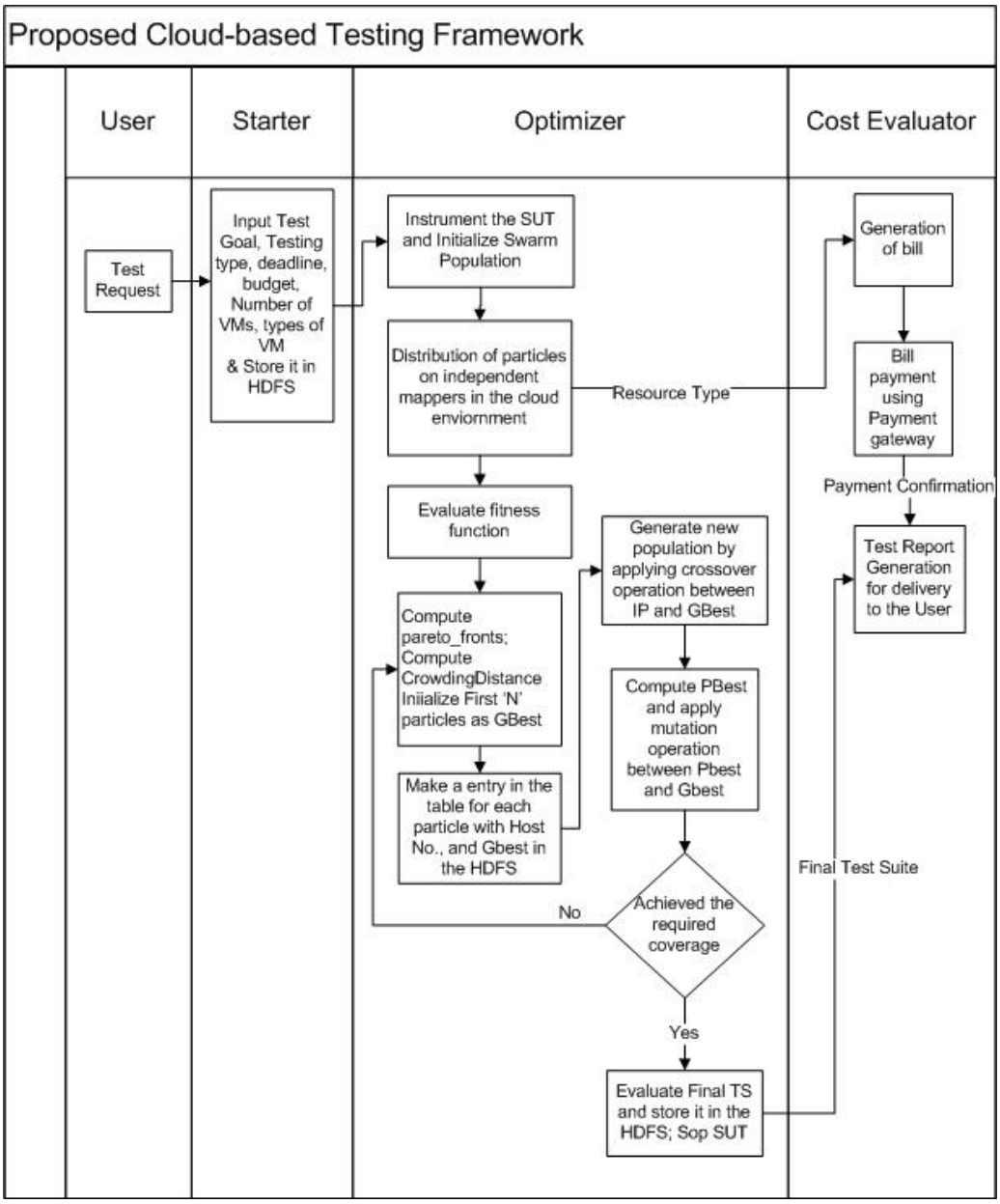

Figure 4.5: Flowchart of the Proposed Framework

testing strategies. Presently, this work is focussed on unit testing as the type of testing offered to the client.

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

i. Test goal of 90% of branch coverage and generation of test cases for crashes.

ii. Cost budget is $500.

iii. Size of the SUT 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 charged for each crash inducing defects. Both $x and $y will be proportional to the size of the SUT.

$$Bill = n_1 * x + n_2 * y \tag{4.2}$$

where n_1 denotes the percentage coverage of SUT, n_2 denotes the number of faults detected and $Bill <=$ Cost Budget

After the completion of test data generation task, Cost Evaluator generates the bill. It reads the data stored in the files related to number and type of resources used. The cost associated with the utilized computing resources is evaluated with the help of pricing model as explained in the next section. Then, it sends bill to the starter module. The starter module sends 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.

4.2.3.1 Pricing Model

Pricing is the process to determine the amount of money received by service provider from the user in exchange for his services. Pricing can be fixed or dynamic. The factor that affects the pricing in cloud computing are :

- Initial cost
- Lease period
- QoS
- Types of resources
- Maintenance Cost
- Comparison with other service providers
- Supply and demand of services

A customer evaluates the pricing model mainly by pricing, QoS and usage type. The pricing can be fixed regardless of volume. In the fixed price plus per unit charges' the customer pays a fixed price plus a unit rate. In Assured purchase volume plus per unit price rate, customer pay fixed price for certain quantity and pays per unit if usage exceeds the limit. In per unit rate with a ceiling customer pays per unit rate with a certain limit. In per unit price, customers pay per unit

price of resources. The QoS requirements include availability, security, privacy, scalability and integrity of the service provider. The service provider with high level of QoS gets an increased number of loyal customers. The usage period defines the time period during which a customer can utilize the services as per SLA between the two parties. The usage period can be perpetual or by subscription or pay-per-use.

The most common pricing model used by service providers is pay-per-use model, in which user is charged a fixed price for each hour Virtual Machine (VM) usage. In recent years, cloud computing has driven shift in the computing paradigm which allows service providers at different layers (application, platform and infrastructure) to offer computing services on demand and pay per use. Nowadays, customers using the Software as a Service (SaaS) model need not to worry about purchasing a license from a software company for installing, upgrading and maintaining the software. Rather, customer buy services on rent from the service providers and rely on them for the acceptable QoS requirements, up gradations and maintenance. To fulfill the expectations of customers and maximize the net profit of SaaS providers, cloud service request model needs to be designed in such a way that it postulates Service Level Agreements (SLA) between clients and service providers. SLA defines the officially authorized agreement between a service provider and a user that states quality of service and its analogous proceedings. It also states the response time, customer budget and penalty in case of performance breakdown among other things. Considering the above explanation and various work related to design of profit model of service providers [66] [67] [68] [75], User Service Request (USR) of the proposed framework is as follows:

f(USR)=f(B, T_{CL}, T_{DL}, β, TC)

The parameters to the $f(USR)$ are Maximum Price Allocation (B), Job completion Time (T_{CL}), Penalty Delay Time (T_{DL}), Penalty Rate (β) and Total Cost Incurred (TC). The pricing model of the proposed framework is given below:

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

- Maximum price (Budget) represented as B
- Deadline represented as T_{DL}
- Penalty rate represented as β

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

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

$$Profit = B - TC_i \tag{4.3}$$

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

$$TC = C_R + C_{DT} + C_{VM} + C_{PD} \tag{4.4}$$

The request processing cost (C_R) is dependent on request processing time (T_R) and hourly price of VM (HP_{VM}).
Thus (C_R) is given by the following:

$$C_R = T_R * HP_{VM} \tag{4.5}$$

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

$$C_{DT} = D_{in} \times P_{in} + D_{out} \times P_{out} \tag{4.6}$$

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

$$C_{VM} = P_{VM} \times T_{VM} \tag{4.7}$$

The penalty delay cost (C_{PD}) represents the penalty levied to service provider for violating the SLA, It is dependent on penalty rate β and penalty delay time period (T_{PD}).
Let T_{CL} is the job completion time. The T_{PD} is defined as :

$$T_{PD} = T_{CL} - T_{DL} \tag{4.8}$$

and

$$C_{PD} = T_{PD} \times \beta \tag{4.9}$$

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

$$T_{CL} = T_{VM} + T_R + T_{DT} + T_{PD} \tag{4.10}$$

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

$$T_{DT} = T_{in} + T_{out} \tag{4.11}$$

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

$$IR_i = Profit_i \div T_{CL} \tag{4.12}$$

The (IR) can be maximized by maximizing profit which in turn can be achieved by minimizing

TC. Minimization of TC is subject to following constraints:

i. $T_D \rightarrow 0$, which can be achieved by job profiler by optimizing the time required for job completion.

ii. C_{DT} can be minimized either by storing local data on HDFS or by using $S3$ bucket which is nearer to the location of hadoop cluster. Hence, TC primarily depends upon C_R and C_{VM}.
Therefore,

$$TC_i = \sum_{i=1}^{l} \sum_{j=1}^{m} T_{R_j} \times HP_{VMj} + P_{VMi} \times T_{VMi} \tag{4.13}$$

iii. Minimize job completion time: The cost of job can be reduced by minimizing the required number and types of VM_i. To help the client in making this smart decision, starfish tool has been inculcated in the proposed framework.

The job will be accepted only if IR is greater than or equal to minimum investment return. If the job is accepted, it will be executed as per the sequence defined in the proposed framework. After completion of test data generation task, Cost Evaluator generates bill. It reads the data stored in the files related to number and type of resources used. The bill is generated based upon the resources used to produce the test data of SUT. Client pays the bill through payment gateway and sends a message to the starter which reads the final test suites from HDFS, and submits a report to the client.

4.3 Conclusion

This chapter presented the framework of testing as a service. The mathematical model of the framework along with its features has been discussed. This is followed by architecture overview and execution flow of the proposed framework. The pricing model has also been designed that considers the QoS interests of both providers and customers by minimizing the total cost incurred. Further, security and authentication mechanism that ensures the security of the data uploaded by the customers has been discussed. The next chapter discusses the verification and validation of the proposed framework and the underlying test data generation strategies. It also lays the empirical results of the hybrid and other existing soft computing based test data generation strategies.

5

Verification and Validation of the Proposed Solution

The previous chapter presented the design layouts of proposed framework of testing as a service.

This chapter focuses on the verification and validation of the proposed framework. The proposed framework has been implemented in Java programming language. It has been verified by conducting experiments on local Apache Hadoop cluster and cluster of AWS EC2 machines. Statistical and practical differences between proposed algorithm and existing algorithms have been assessed with a two-tailed Mann-Whitney U-test. The magnitude of improvement in the proposed algorithm is quantified with the Vargha and Delaney's standardized effect size $\hat{A}_{12}$. Vargha and Delaney's $\hat{A}_{12}$ statistics is a standardized effect size that acts as an objective and standardized measure of the magnitude of observed effect.

At the outset, this chapter outlines the details of verification and implementation of the proposed work. The accuracy of the proposed framework has been assessed with respect to various perspectives used in the existing frameworks such as fault detection capability, branch coverage, resources utilized, time taken etc. The results obtained have been compared with other related cloud-based testing models such as YETI, Cloud9, HadoopUnit etc. Finally, the chapter concludes with validation of the proposed solution.

5.1 Phases of research work

The research method presented in this thesis has been accomplished in three major phases as shown in Table 5.1. In the first phase, the sequential test data generation strategy using GA and PSO algorithm has been designed and implemented. The experimental results have been obtained on single machine using container classes as case study subjects. The reason for selecting the container classes is that these classes are mostly used in the software development. PSOG has been compared with the other existing hybrid strategies, memetic algorithms, singular genetic and particle swarm optimization. The empirical results proved the superiority of the hybrid strategy proposed in this thesis. The second phase implements the map reduce version of the above said strategy and is named as MapReduce based test data generation strategy (MRPSOG). The concept of pareto-optimality has also been used to compute the pareto front which helps to identify optimal test suites with the assurance of maximum branch coverage and maximum fault detection in the software in minimum time. The efficacy of MRPSOG has been verified on a hadoop cluster of ten nodes. It has also been compared with other existing cloud-based test data generation strategies. In the third phase, the framework of testing as a service has been designed and compared with other existing frameworks. The proposed framework has been evaluated and analyzed statistically using real world open source libraries and experimental results proved that it significantly outperforms other existing cloud-based testing models.

Table 5.1: Phases of Research Work

Scenario	Description
Phase 1	PSOG implemented on single node and compared with other existing hybrid strategies like memetic algorithms and traditional genetic and particle swarm optimization algorithms.
Phase 2	Map reduce version of the PSOG implemented with additional features
Phase 3	Implementation of Testing as a service framework.

The comparison of results is also classified into three categories, according to the progress of the research work carried out in three phases presented by three different Sections.

i. Section 5.2 outlines the comparative analysis of PSOG with existing soft computing based strategies.

ii. Section 5.3 discusses the findings of MRPSOG and its comparison with existing map reduce based GA strategy with respect to parameters like resources utilized, time taken, fault detection capability and branch coverage.

iii. Section 5.4 presents the experimental results of the proposed framework and its comparison with other existing cloud-based testing frameworks such as YETI, Cloud9, Hadoop-Unit etc.

5.2 Experimental Evaluation of PSOG

The proposed strategy has been evaluated in terms of improvement in the efficiency of test suite generation. The metrics used to measure the efficiency is code coverage per unit time.

5.2.1 Test Adequacy Criteria

Test adequacy criteria measures the completeness of the software. The software is said to be adequate w.r.t criterion C if it satisfies it. Test adequacy criteria are categorized based on the type of testing. In black box testing, test adequacy criterion is defined in terms of the coverage of requirements of the software. A test set is said to be adequate if all the identified specifications have been covered and SUT is completely exercised. White-box testing postulates testing requirements in terms of program under test. Test set is said to be adequate when the software under test is completely exercised. Test adequacy criterion can be further divided based upon the underlying testing approach. Testing requirements are specified in terms of coverage of particular set of elements (such as branch coverage, path coverage, statement coverage) in structure of the program in structural testing.

In fault-based testing, test adequacy criterion refers to the measurement of fault detection capability. Fault-based adequacy criteria measure the quality of a test set according to its effectiveness or ability to detect faults. Error seeding is a fault-based testing originally proposed to estimate the number of faults that remain in software. By this method, artificial faults are introduced into the program under test in some suitable random fashion unknown to the tester. It is assumed that these artificial faults are representative of inherent faults in the program in terms of difficulty of detection. Then, the software is tested and the inherent and artificial faults discovered are counted separately. Let r be ratio of the number of artificial faults found to the number of total artificial faults. Then the number of inherent faults in the program is statistically predicted with maximum likelihood to be f/r, where f is the number of inherent faults found by testing. This method can also be used to measure the quality of software testing. The ratio r of the number of artificial faults found to the total number of artificial faults can be considered as a measure of the test adequacy. If only a small proportion of artificial faults are found during testing, the test quality must be poor. In this sense, error seeding can show weakness in the testing. Branch coverage can be defined as follows:
A set P of execution paths satisfies the branch coverage criterion if and only if for all edges e in the flow graph, there is at least one path p in P such that 'p' contains the edge 'e'. Branch coverage is stronger than statement coverage because if all edges in a flow graph are covered, all nodes are necessarily covered. Therefore, a test set that satisfies the branch coverage criterion must also satisfy statement coverage. Such a relationship between adequacy criteria is called the subsumes relation.

5.2.2 Case Study

Ten Java classes from standard Java API 1.6, package java.util have been taken as subject to carry out the experiments. Table 5.2 summarizes their features. The classes that are chosen for testing covers varied range of lines of code i.e. from 100 to 1600 lines of code. Moreover, these classes Under Test (CUTs) have varied internal complexity that can be seen from lines of code. Fitness function evaluations, coverage and time taken for the generations of test suites have been obtained on an Intel Core i5-3210M CPU notebook having 2.5GHz processor and 4 GB of RAM.

Table 5.2: Features of Test Object

Test Object	LOC	No. of Branches
TreeList	901	23
BitSet	606	156
Stack	136	20
StringTokenizer	195	55
Vector	1019	100
LinkedList	708	84
BinaryHeap	334	61
TreeMap	1636	191
BinomialHeap	335	79
BinTree	154	37

5.2.3 Experimental Results

Statistical and practical differences have been computed using two-tailed Mann-Whitney U-test as per the guidelines in [271] [108]. Vargha and *Delaney's* $\hat{A}_{12}$ statistics has been used as standardized effect size [273] [274] in conjunction with Null Hypothesis Significance Testing (NHST). An effect is an objective and standardized measure of the magnitude of observed effect. Null Hypothesis for the statistical test states that there is no difference between novel strategy and existing strategy. Alternative hypothesis states Novel Strategy is better than the existing strategy. Pair wise comparisons have been performed between the novel strategy and all other existing strategies. For the comparisons, it was decided to choose Percentage coverage achieved per unit of execution time (C) as a criterion. As coverage should always be maximized whereas execution time should be minimized, so percentage coverage achieved per unit of execution time should be maximized for a better strategy. All the strategies have been executed 30 times to gather sufficient information on the probability distribution of C, performance of all the compared strategies is shown in Table 5.3. The level of significance is set to 0.05. Vargha and *Delaney's* $\hat{A}_{12}$ statistics measure the probability that executing strategy X yields superior C values than executing another strategy Y. It can be computed easily as per the following formula [271]:

$$\hat{A}_{12} = (R1/p - (p+1)/2)/q \tag{5.1}$$

where R1 is the rank sum of the first data group that is being compared. In the above formula, p is the number of observations in the first data sample, whereas q is the number of observations in the second data sample. Results in Table 5.4 exhibit positive results with high statistical

Table 5.3: Comparison of the different Optimization algorithms on the container cluster. Each algorithm has been stopped after evaluating up to 10,000 solutions. The reported values are calculated on 30 runs of the test.

Case Study	Strategy	Minimum	Median	Maximum	Mean	Std. Deviation	Std. Error
BinaryHeap	GA	0.8304	0.8519	0.8624	0.8469	0.01128	0.00206
	ETOC	1.179	1.192	1.22	1.197	0.0172	0.003139
	MEM	4.963	5.222	5.588	5.258	0.2608	0.04762
	PSOG	5.765	6.094	6.6	6.194	0.3401	0.0621
	PSO	0.8868	0.8962	0.9048	0.8975	0.006453	0.001178
	GAPSO	1.220	1.227	1.240	1.227	0.007303	0.001333
TreeList	GA	0.5029	0.5145	0.5244	0.5139	0.008948	0.001634
	ETOC	0.8571	0.9427	0.9977	0.9354	0.04088	0.007464
	MEM	5.5	5.765	6.188	5.844	0.2602	0.04751
	PSOG	6.667	7.071	7.654	7.131	0.4121	0.07524
	PSO	0.5993	0.6050	0.6133	0.6058	0.004685	0.0008554
	GAPSO	0.9567	0.9636	0.9853	0.9685	0.01239	0.002262
Bitset	GA	0.4501	0.4559	0.4673	0.4577	0.007263	0.001326
	ETOC	0.5621	0.5647	0.5655	0.5641	0.001455	0.000266
	MEM	1.153	1.193	1.207	1.184	0.02338	0.004268
	PSOG	1.269	1.282	1.286	1.279	0.007188	0.001312
	PSO	0.5200	0.5295	0.5338	0.5278	0.005862	0.001070
	GAPSO	0.6901	0.6986	0.7071	0.6984	0.006191	0.001130
Vector	GA	3.043	3.197	3.41	3.217	0.153	0.02793
	ETOC	5.389	6.188	6.533	6.037	0.4874	0.08899
	MEM	11	12.38	12.5	11.86	0.7053	0.1288
	PSOG	14.29	19.8	20	18.3	1.815	0.3313
	PSO	3.770	3.881	4.133	3.913	0.1462	0.02670
	GAPSO	4.600	4.947	5.278	4.943	0.2378	0.04342
StringTokenizer	GA	4.261	4.543	4.796	4.56	0.1845	0.03369
	ETOC	12.13	12.38	16.5	13.2	1.433	0.2616
	MEM	16.33	19.8	33	22.31	6.242	1.14
	PSOG	19.8	24.88	33.33	25.76	4.99	0.9111
	PSO	4.335	4.605	4.861	4.639	0.1862	0.03400
	GAPSO	9.700	10.34	11.00	10.34	0.5676	0.1036
TreeMap	GA	5.326	5.683	5.755	5.694	0.07773	0.01419
	ETOC	12.13	12.38	16.5	13.2	1.433	0.2616
	MEM	24.5	32.67	33	29.05	4.181	0.7633
	PSOG	19.8	33.08	33.33	31.07	3.971	0.725
	PSO	6.555	6.640	7.193	6.804	0.2765	0.05049
	GAPSO	10.78	11.00	12.25	11.30	0.6323	0.1154
Hashtable	GA	4.261	4.543	4.796	4.56	0.1845	0.03369
	ETOC	12.13	12.38	16.5	13.2	1.433	0.2616
	MEM	33.17	49.75	50	46.55	6.765	1.235
	PSOG	19.8	49.5	50	39.74	12.63	2.305
	PSO	4.300	4.619	4.901	4.644	0.1877	0.03427
	GAPSO	13.86	16.33	19.80	16.48	2.013	0.3675
Linked List	GA	4.261	4.543	4.796	4.56	0.1845	0.03369
	ETOC	12.13	12.38	16.5	13.2	1.433	0.2616
	MEM	33	50	49.75	46.51	6.762	1.235
	PSOG	33	99	100	76.97	28.77	5.252
	PSO	4.400	4.684	4.944	4.722	0.1905	0.03478
	GAPSO	13.86	16.17	19.80	15.83	1.849	0.3375
Stack	GA	6.082	6.16	7.108	6.318	0.3033	0.05537
	ETOC	25	25	33.33	27.22	3.748	0.6843
	MEM	33.33	33.33	50	41.11	8.457	1.544
	PSOG	25	33.33	50	40	9.129	1.667
	PSO	7.750	8.165	8.591	8.166	0.3680	0.06718
	GAPSO	16.67	20.00	20.00	19.11	1.499	0.2737
BinaryTree	GA	6.373	6.52	7.523	6.667	0.3261	0.05954
	ETOC	25	25	33.33	27.22	3.748	0.6843
	MEM	33.33	33.33	50	41.11	8.457	1.544
	PSOG	25	33.33	50	40	9.129	1.667
	PSO	16.67	16.67	20.00	18.00	1.661	0.3032
	GAPSO	25.00	25.00	33.33	27.22	3.748	0.6843

confidence which certainly answers RQ. The proposed novel strategy is better than all other strategies except MEM in few cases. The novel strategy PSOG is 100% times better than MEM in case of BinaryHeap, TreeList, BitSet and Vector. The performance of PSOG in case of TreeMap and that of StringTokenizer and LinkedList is 80% and 76% times better than MEM respectively. This is due to the following reasons:

i. The strategy has retained only those particles intelligently that possess better fitness function values resulting from crossover and mutation operators. The other particles were rejected and not included in the population. The proposed strategy is able to generate particles with best fitness function values in much less iterations of genetic and particle swarm optimization algorithm. This has resulted in fastest convergence rate. On the other hand, the memetic algorithm is quite complex as it utilized hill climbing metaheuristic for local optimization and it involves much compute intensive operations. Furthermore, the combination of genetic operators and traditional particle swarm optimization algorithm has proved to be more exploitative and explorative. Whereas the elitism rate and population set of memetic algorithm are set to one and ten respectively.

ii. The test cases are prioritized and minimized effectively. The redundant test cases are deleted. Thus, generated test suites have been compact and takes less time while execution.

The performance of PSOG has not shown remarkable results in case of Hashtable, Stack and BinaryTree. MEM is equivalent or slightly better than the proposed strategy in case of Stack, BinaryTree and Hashtable. This may be due to the fact that in these classes there is comparatively lesser number of branches and hence the effectiveness of the proposed strategy could not produce remarkable difference.

Fig. 5.1 shows the box plots showing comparison of all the strategies w.r.t CUT which clearly presents the performance of novel strategy better than others.

Fault seeding experiments have been performed with the aid of JACA [107], a fault injecting tool to assess the quality of test data generated by the proposed strategy. Ten faults have been injected to simulate classic programming errors like use of wrong relational operator, wrong parameter value and missing condition etc. Out of the ten case study subjects, test data of seven case studies were able to detect 100% faults injected in the code, which is highly optimal. In the case of other three case study subjects, the assertion failures were found out to be 90%. It has been shown in the Table 5.5. This was mainly due to the lack of required data flows associated with some attributes of an object.

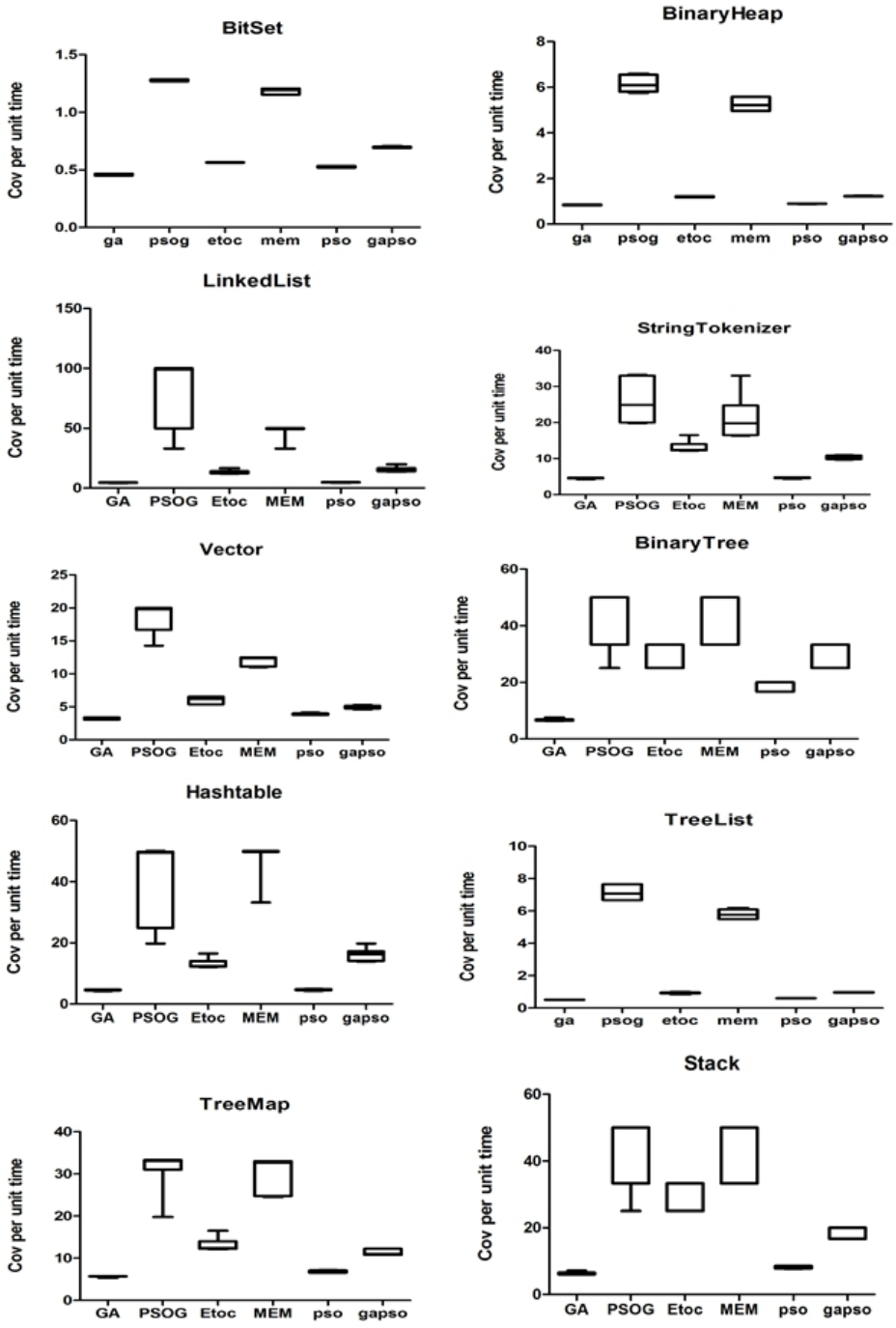

Figure 5.1: Boxplots percentage coverage per unit time for all the container classes.

Table 5.4: Comparison of percentage coverage per unit time obtained by the novel strategy PSOG and by Memetic algorithm MEM. Significant values of Vargha and Delaney's effect size ($\hat{A}_{12}$) are shown in bold.

CUT	Strategy	Sum of ranks	$\hat{A}_{12}$
BinaryHeap	MEM	465	1
	PSOG	1365	
TreeList	MEM	465	1
	PSOG	1365	
Bitset	MEM	465	1
	PSOG	1365	
Vector	MEM	465	1
	PSOG	1365	
StringTokenizer	MEM	675	0.76667
	PSOG	1155	
TreeMap	MEM	639.5	0.80667
	PSOG	1191	
Hashtable	MEM	1112	0.28167
	PSOG	718.5	
LinkedList	MEM	675.5	0.76556
	PSOG	1154	
Stack	MEM	946	0.46556
	PSOG	884	
BinaryTree	MEM	946	0.46556
	PSOG	884	

Table 5.5: Experimental results of Fault Seeding

Case Study Subjects	Assertion Failures
BinaryHeap	10/10
TreeList	10/10
BitSet	9/10
Vector	10/10
StringTokenizer	9/10
TreeMap	9/10
Hashtable	10/10
LinkedList	10/10
Stack	10/10
BinaryTree	10/10

5.2.4 Validation of PSOG

The proposed strategy of test data generation on single node has been evaluated by performing experiments with ten container classes from the Java standard library. The algorithm has been analysed statistically with branch coverage as test adequacy criterion. Percentage coverage per unit of time and percentage of faults detected by the generated test data have been taken as performance adequacy criteria. Figures (in experimental results section)showed that the test case generation is efficient in the proposed work. The proposed test data generation strategy has been validated against the features of existing heuristic based upon GA, PSO, existing hybrid strategies based on GA and PSO and Memetic algorithm. The comparision table has been depicted in Table 5.6.

5.3 Performance Evaluation of Proposed Framework

The designed three test cases for evaluation of proposed framework have been stated below:
Test Case 1: Fault detection and coverage capabilities of underlying map reduce based test data generation strategy.
Test Case 2: Performance comparison of map reduce based strategy and sequential strategy.
Test Case 3: Performance comparison in terms of time taken and resources utilization.

5.3.1 Case Study

Five open source libraries provided by Apache have been taken as the case study subjects as shown in the Table 5.7. The subjects chosen for testing covers varied range of lines of code i.e. from 6337 to 19041 lines of code. Moreover, these case study subjects have varied internal complexity that can be seen from lines of code. The strategy has been evaluated over the Apache Multi-node Hadoop cluster setup in the LAN. Ten nodes have been taken in the cluster, each having the hardware configuration as Intel Core i5-3210QM CPU as the processor with 2.5GHz processor, 4 GB of RAM and software configuration as Hadoop version of 1.04 and Ubuntu 12.04 on the machines. 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 to the 30 Mappers in parallel.

5.3.2 Experimental Design, Results & Analysis

In order to answer the research questions, numerous experiments have been conducted. The design of experiments is planned as follows as per the guidelines in [271] [272]:

i. Performing Null Hypothesis Significance Testing by defining Null Hypothesis for the comparison of two strategies.

ii. Two-tailed Mann-Whitney U-test for performing statistical analysis in conjunction with NHST.

iii. Evaluation of Vargha and *Delaney's* $\hat{A}_{12}$ statistics as standardized effect size which acts as an objective and standardized measure of the magnitude of observed effect.

5.3.2.1 Test Case 1: Fault detection and coverage capabilities

To answer the RQ1, experiments have been performed to gather data of traditional genetic algorithm based strategy and the proposed novel strategy with respect to Average Percentage of Faults Detected (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

$$APFD(T,S) = 1 - \sum_{i=1}^{l} detect(i,T) \div nl + 1 \div 2n \tag{5.2}$$

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 percentage increase in APFD score is shown in the box plots in the Figure 5.4 and found that on an average percentage increase in APFD score comes out to be 4.86. The seeded faults simulate classic programming errors like use of wrong relational operator, wrong parameter value and condition missing etc.

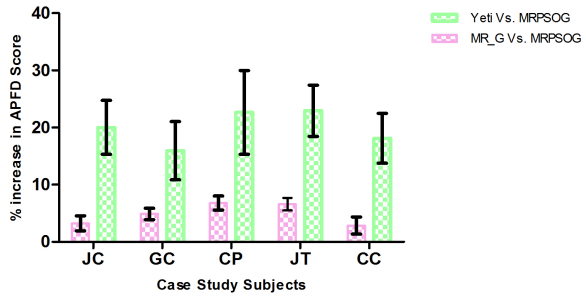

Figure 5.2: Boxplots of percentage increase in APFD for the case study subjects.

Percentage Coverage achieved per unit of execution time(C) has been chosen as a criterion for performance comparison. As coverage should always be maximized whereas execution time should be minimized, so percentage coverage achieved per unit of execution time should be maximized for a better strategy. The strategies have been executed 30 times to gather sufficient information on the probability distribution of C. Performance of the compared algorithms is shown in Table 5.8 and is also shown graphically in Figure 5.3.

Figure 5.3 shows the box plots showing comparison of all the strategies w.r.t SUT which clearly presents that the C score of novel strategy is better than others. Results in Table 5.8 are shown where each algorithm has been stopped after evaluating up to 200,000 solutions. Vargha and *Delaney's* $\hat{A}_{12}$ statistics have also been computed by using the formula stated in Equation 5.3. It measures the probability that running algorithm A yields higher C values than running another algorithm B. The reported values are calculated on 30 runs of the test. It clearly exhibits positive results with high statistical confidence that certainly answers RQ1. The average percentage increase in coverage per unit of time comes out to be 48.45. Hence the proposed novel strategy is better than the other strategies.

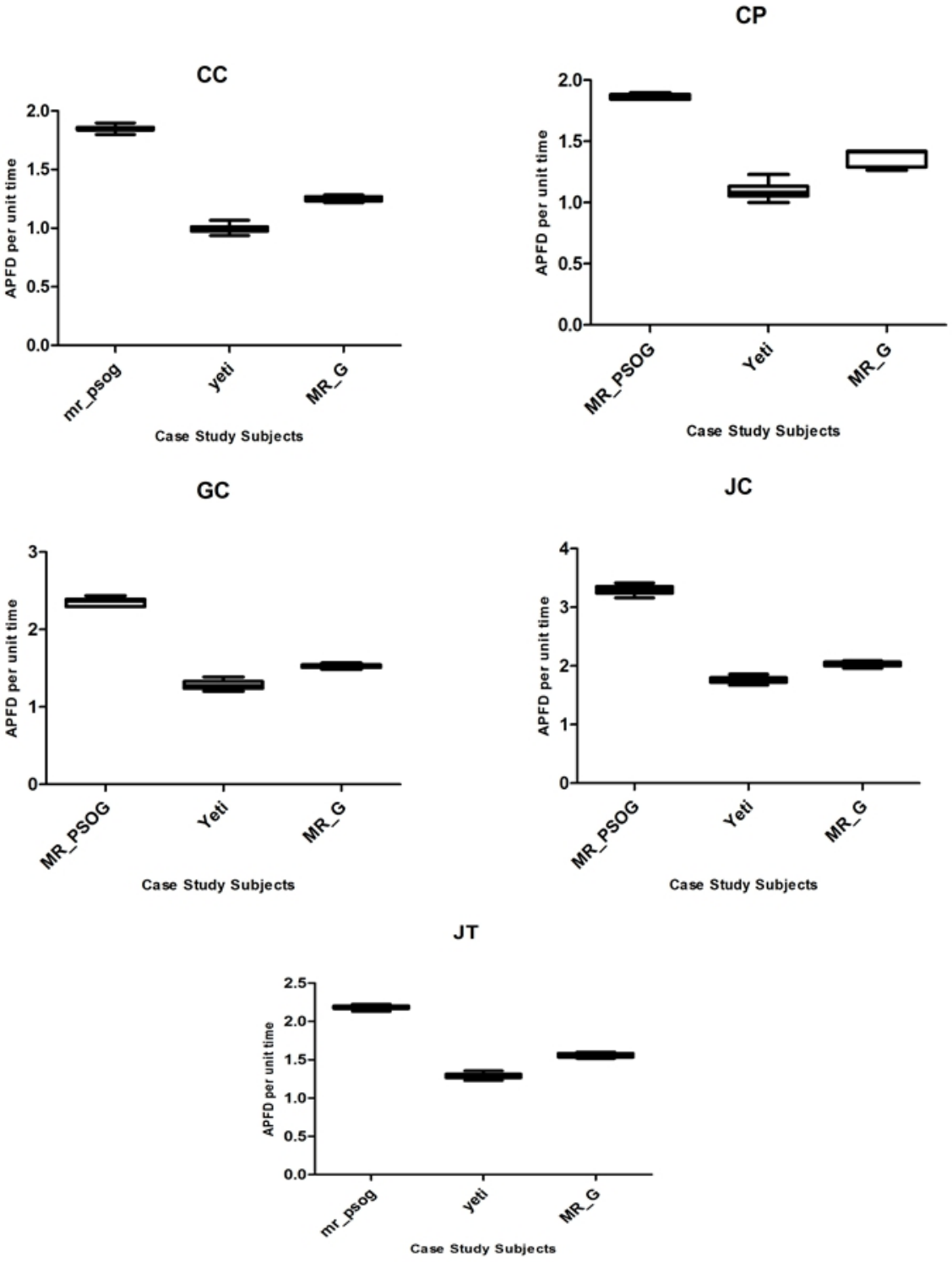

Figure 5.3: Boxplots of percentage coverage per unit time (C) for all case study subjects.

5.3.3 Test Case 2: Performance Comparison of Map-Reduce based strategy and the sequential strategy

The Apache Map Reduce based strategy achieves the same APFD and Coverage that has been achieved 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 Figure 5.4. In order to provide a more concrete quantitative analysis of the answers to RQ2, the results obtained have been compared using Two-tailed Mann-Whitney U-test for statistical significance.

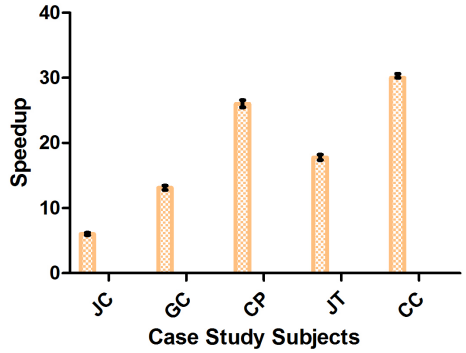

Figure 5.4: Boxplots of speedup for all case study subjects.

5.3.4 Test case 3: Performance Comparison in terms of time taken and resources utilization.

The performance of proposed hybrid strategy has been compared with traditional genetic algorithm based strategy. The metrics used for performance comparison are number of resources used and time taken to generate test cases. The resource requirement has been measured by measuring the number of mappers. It has been found that the proposed strategy is able to produce the required results in much less time and resources than the traditional genetic algorithm based strategy. The proposed strategy is entirely different from the work proposed in [107] [115] in a way that this strategy has restricted all the operations related to fitness evaluations and genetic operations to the Mappers and few work is left for reducers. On the other hand, the authors in [107] [115] performed fitness evaluation through Mappers whereas other genetic operations were performed over Reducer. This results in delay in 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. Extensive experiments have been performed and it has been found that the proposed framework is on the average 31.08% faster than traditional genetic algorithm based test data generation. These results have been shown in Figure 5.5.

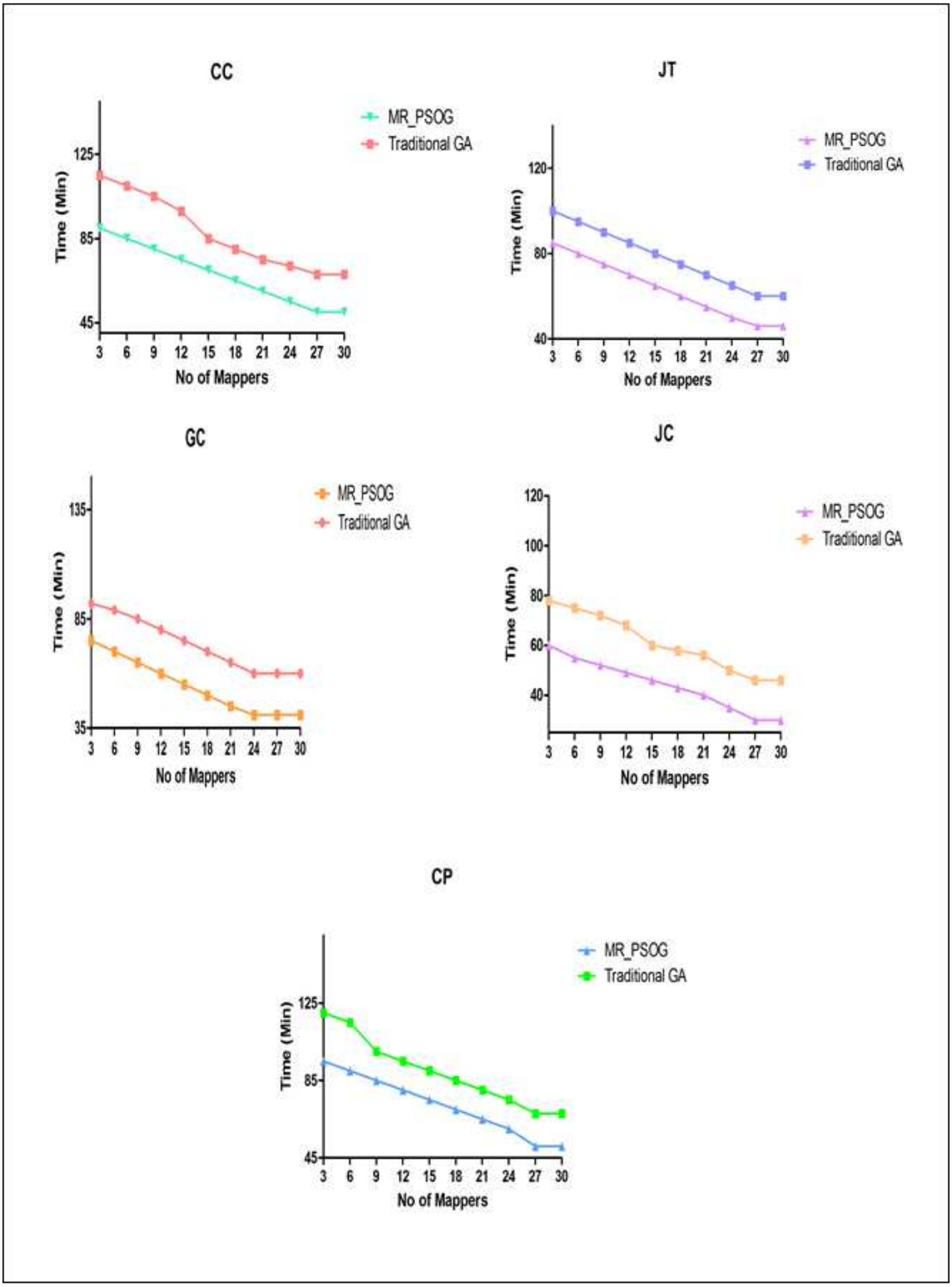

Figure 5.5: Comparison of models from resource requirement aspect

The above results have been obtained with the parameter settings as shown in the Table 5.10.

5.3.5 Validation of MRPSOG

In the proposed strategy of Apache Hadoop MapReduce based test data generation, the test adequacy criteria chosen by us are branch coverage and faults detection. The performance adequacy criteria 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. Figures (in experimental results section) proves the efficacy of the proposed solution on Hadoop Cluster with ten nodes. The proposed test data generation strategy has been validated against the features of the existing cloud-based test data generation frameworks and comparison chart has been depicted in Table 5.11.

5.4 Experimental Evaluation of Proposed Framework

In this section, the performance and benefits of the proposed framework have been evaluated. The experimental evaluation has been partitioned into two segments. First segment contains the detailed analysis of the proposed framework on the test data generation effectiveness and second segment details the investigation of its overall performance.

5.4.1 Experimental Setup

Two environments have been used in the evaluation, public cloud hadoop cluster on AmazonEC2 [109] and local hadoop cluster.

Public Cloud: The type of EC2 instances launched by us for ten node cluster is the t2.medium type that is defined as 2 vCPU Intel Xeon Processors operating at 2.5GHz with Turbo up to 3.3GHz, 4GiB memory with EBS as local storage and running on Ubuntu 14.04 LTS 64 bit platform.

Local Cluster Setup: Ten nodes in the cluster have been taken having the hardware configuration of Intel Core i5-3210QM CPU as the processor with 2.5GHz processor and 4 GB of RAM. The network is 1 Gbps and the nodes are connected through a single switch. Hadoop version of 1.04 and Ubuntu 14.04 on the machines have been installed. The job of test data generation of a jar file is distributed to the 30 Mappers in parallel.

5.4.2 Case Study

To validate the effectiveness of testing strategy, case study subjects opted in this work are shown in the Table 5.9. The source lines of code of the case study subjects varies between 60LOC to 296LOC. The MRPSOG is configured with a population size of 200. The maximum limit for test case size is positioned to 60 statements and the number of solutions evaluations has been set to 200,000. The crossover probability is set to 0.8 and mutation probability to 0.2 with equal probabilities for insertion, deletion and change. The experiments are repeated 30 times with different random fault seeds.

5.4.3 Metrics

To evaluate the performance and cost effectiveness of the framework, the following metrics have been employed:
Testing effectiveness: To measure the effectiveness of testing, the metrics used are Percentage coverage achieved and percentage of faults detected.
Framework effectiveness: Cost and Time (comparison with cloud and local cluster) metrics are utilized to measure the performance of the proposed framework.

5.4.4 Experimental Results

5.4.4.1 Testing Effectiveness

Experimental results shown in Table 5.12 clearly depict improvement in structural coverage for all the considered case study subjects by using MRPSOG. The increase in median coverage varies from 4.6% for the case of Rational to 10.15% for Option. The minimum branch coverage and method coverage achieved in the case for Option are 82% and 94% respectively with MRPSOG. On the other hand, the maximum branch coverage and method coverage achieved with test suites by using traditional genetic algorithm are 88.57% and 92.73% respectively. Unpaired t-test has been carried out for finding the statistical significance of the results obtained and the p-value comes out to be less than 0.001 which clearly increases the statistical confidence of the obtained results. For DocType and ArrayIntList the median coverage increase comes out to be 9.34% and 8.2% respectively. However, there are relatively lesser gains in the case of achieved branch coverage which needs further investigation. But, for all the case study subjects the fault detection ability is quite significant except DocType. Thus, the conclusion can be made on the overall better test effectiveness of the proposed strategy.

5.4.4.2 Framework Effectiveness

The effectiveness of framework is assessed by the cost and time taken by framework to perform certain task of test data generation. The cost required to set up cloud based framework has been evaluated. The associated cost of equivalent local set up in the lab has also been evaluated.

Cost Comparisons

To perform the experiments, *t*2.*medium* AWS instance [110] has been selected. The price of t2.medium = $0.052/hour. Hence the Cost of ten AWS instance is:

$$C_{aws} = \$0.52 \tag{5.3}$$

In the case of local cluster setup, the cost of machine (C) is a factor of Initial cost of machine (i_c), Cost of Maintenance (c_m) and Cost of Installations (c_i).

$$C = i_c + c_m + c_i \tag{5.4}$$

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

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

$$c_m = c_s + c_e + c_c + c_f + c_p \tag{5.5}$$

Cost of installations (c_i) is a function of Cost of license (c_l) and Cost of expertise of personnel(c_{ep}).

$$c_i = c_l + c_{ep} \tag{5.6}$$

When TC is compared with the machine setup on AWS, the TC is strictly greater in many folds than the Cost of ten AWS instances. Hence, the cost of local cluster is much higher than the cost of AWS machines used over internet as well as there are no hassles of installations, procurement and maintenance. The AWS machines are released after the completion of task. This allows considerable amount of savings.

Time Comparisons

In this section, the time requirements for the local cluster setup and AWS machines have been

compared.
Total time for AWS cluster setup $Time_{aws}$ is a sum of Time for installations (T_i), Time for cluster setup (T_{cs}) and Time for performing experiment (T_{pe}).

$$Time_{aws} = T_i + T_{cs} + T_{pe} \tag{5.7}$$

The time requirements for local cluster setup $(Time_{lcs})$ depends on Time for procuring machines(T_{pm}), Time for installations on local cluster$(T_{il}$, Time for local cluster setup T_{lcs} and Time for performing experiments(T_{pe}).

$$Time_{lcs} = T_{pm} + T_{il} + T_{lcs} + T_{pe} \tag{5.8}$$

It can be easily proved that $Time_{lcs}$ is greater than $Time_{aws}$ as T_{pm} is usually at higher end as compared to the machine allocations on AWS.
From the above mathematical equations, it is clearly depicted that the proposed framework is efficient and cost effective.

5.4.5 Validation of Proposed Framework

In the proposed framework, cloud-based testing has been designed with stronger focus on QoS parameters such as fault detection capability, resource utilization, security and costs which were not considered traditionally. Figures (in experimental results section) show that proposed framework is comprehensive, cost effective and efficient automatic software test data generation framework. It has been validated against the features of the existing cloud-based testing models and corresponding results have been depicted in Table 5.13.

5.5 Conclusion

In this chapter, verification details, experimental setup and testing results of the proposed work have been demonstrated. The experimental results have been illustrated for the first stage test data generation strategy using genetic and particle swarm optimization algorithm (PSOG) by executing it using single node environment. Comparative analysis of PSOG with existing soft computing based strategies has been outlined in terms of achieved code coverage per unit of time. The performance of MapReduce based test data generation framework (MRPSOG) has been compared with the existing MapReduce based genetic strategy with respect to parameters like resources utilized, time taken, fault detection capability and branch coverage. The comparative analysis has been done with other existing frameworks like YETI, Cloud9, HadoopUnit etc. The experimental results demonstrate that the proposed solutions are working and can be effectively used to address cloud-based test data generation. The next chapter concludes the thesis and also discusses the future directions.

Table 5.6: The Comparison Table of Existing Test Data Generation Strategy

Sr. No.	Strategy	Technology Used	Test adequacy criteria	Performance Criterion	Case Study	Input Format
1	Windisch et al. [261]	PSO	Branch coverage	Fitness function evaluations Number of iterations	25 small artificial test objects and 13 complex industrial test objects	Instrumented source program in MATLAB GEATbx
2	Wegner et al. [140]	Evolutionary Testing	Mean coverage	Mean number of test data	Six C programs	Source program in C
3	Li et al. [266]	Hybrid GA and PSO	Path Coverage	Execution time	Triangle benchmark problems	C source code
4	Singla et al. [245]	Hybrid GA and PSO	dcu and dpu	Number of Test Cases and cover ratio percentage	Simple Programs	MATLAB
5	Zhang et al. [247]	GA and PSO	Path coverage	Number of iterations and average execution time	Triangle Classification problem	Instrumented source program in C
6	Kaur et al. [249]	Hybrid GA and PSO	Fault coverage	Execution time and APFD	Simple Programs	JAVA
7	Wapler et al. [259]	Genetic Algorithm	Distance metrics	Branch Coverage %	seven java classes	ECJ
8	eToc [117]	GA	Branch Coverage	Test cases, number of fitness evaluation and time	Container classes	Instrumented source code
9	Arcuri et al. [263]	GA and Hill Climbing	Branch Coverage	Time and coverage %	Container classes	Java; Instrumented code
10	Nayak et al. [264]	GA and PSO	dcu%	dcu%	Small 14 FORTRAN programs	Instrumented code; MATLAB
11	Ahmed et al. [265]	Combinatorial Testing using PSO	interaction elements	Test size	IPOG, WHITCH, Tenny, TConfig and TVG	
12	Li et al. [244]	PSO	Path coverage	Iteration time; Time consumption	Benchmark Triangle problem & Binary Search program	Instrumented code in C
13	Fraser et al. [267]	GA	Branch Coverage	Branch Coverage	Industrial project and five open source libraries	Bytecode Java
14	GONG et al. [268]	GA	Path coverage	Fault detection and coverage	bubble sort program Siemens suite, print_tokens, schedule, replace, tcas,space, flex	Instrumented program in C
15	Traditional GA	GA	Branch Coverage	Branch Coverage	Container classes	Instrumented Source code in Java
16	The Proposed Strategy	GA and PSO	Branch coverage per unit time	Time, No. of test cases, Fault Detection and Coverage per unit time	Container classes	Bytecode or Source code

Table 5.7: 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

Table 5.8: Comparison with Existing Traditional Genetic Algorithm based Testing Models in Cloud Computing Environment.

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	
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	
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	
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	
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	

Table 5.9: Case Study Subjects

Case Study Subjects	**LOC**	**Methods**	**Branches**
Option (Commons CLI)	155	42	96
Rational(Math4J)	61	19	36
DocType(XOM)	296	26	242
ArrayIntList(CP)	65	12	28

Table 5.10: Parameters Settings

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

Table 5.11: Comparison Chart of Existing Cloud-based Testing Models

Testing Model	Strategy	Implementation	Parallelization Framework	Platform	Benefits	SUT
Yeti	Automated Random Testing	Java	Apache Hadoop MapReduce	Cloud/Amazon EC2	Test Execution Speedup	Java.lang
VATS	Performance Testing using HPLoadRunner	SmartFrog	Xen	Cloud	Improved Service Performance	SAP/R3 System
D-Cloud	Fault Injection	XML	QEMU and Eucalyptus	Cloud	Cost and Time	Parallel/ Distributed
Cloud-9	Symbolic Execution	Symbolic Tests API	-	Eucalyptus and Amazon EC2	on-demand Software Testing Service; Speedup	UNIX Utilities
GridUnit	Regression testing	-	-	Grid	Speedup	JUnit Test Case Execution
Joshua	Regression testing	-	-	Jini	Speedup	JUnit Test Case Execution
Korat	Constraint-based Testing	Java	Google MapReduce	–	Object graph visualization	Google Applications
NMS	Performance Testing	–	—	–	Less expensive and more scalable implementation	Simulation of large Scale Networks
TaaS	Prototype of TaaS Over Cloud	–	–	Cloud	Elastic resource infrastructure; provides various kind of testing services to users	Web Applications
HadoopUnit	Regression Testing		Apache Hadoop MapReduce	Hadoop Cluster	Speedup in Test Execution	JUnit Test Cases
The Proposed Framework	Unit Testing	Java	Apache Hadoop MapReduce	Hadoop Cluster	Speedup, better code coverage and Fault detection	5 open source library
Linda et.al.	Unit Testing	Java	Apache Hadoop MapReduce	Hadoop Cluster	Speedup	One open source library

Table 5.12: Statistical results of all the Case Study Subjects

Case Study Subjects	Statistics	%Branch Coverage			%Statement Coverage			%Method Coverage			%Faults detected		
		MRPSOG	GA	p value	MRPSOG	GA	p value	MRPSOG	GA	p value	MRPSOG	GA	p value
Option	Min	82	80		90	86.36		94	90.48		50	44.5	
	Median	88.5	85.71		96.23	91.82		97	94.05		59	51.69	
	Mean	88.7	82.26	0.0009	95.08	92.05	0.0004	96.92	95.24	0.002	58.57	41.27	0.0001
	Variance	4.5	2.93		3.78	2.21		2.23	1.77		5.841	3.78	
	Max	94	88.57		100	92.73		100	95.24		66	55.66	
Rational	Min	80	75		95	90.89		93.47	89.47		45	15.74	
	Median	87	85		97	96.3		97	96.5		68.5	52.32	
	Mean	85.73	83.33	0.0386	97.4	96.05	0.0185	96.57	95.12	0.0009	67.7	53.7	0.0047
	Variance	3.79	4.92		2.05	2.26		2.6	3.04		15.05	21.3	
	Max	90	90		100	100		100	100		85	82.41	
DocType	Min	78	59.23		88	71.66		94	84.62		40	30	
	Median	88	84.62		95	89.04		99	98		80	45.31	
	Mean	85.8	81.41	0.0626	93.2	86.36	0.0001	97.93	96.08	0.8712	64.4	60.62	0.3299
	Variance	5.61	10.38		3.71	7.84		2.3	4.49		17.35	11.96	
	Max	92	92.31		98	95.19		100	100		80	70.13	
ArrayIntList	Min	69	66.67		80	71.43		91	91.67		30	10.47	
	Median	80	77.78		85	79.79		96	95.23		39	30.2	
	Mean	80.8	78.7	0.3191	86.6	80.21	0.0005	96.23	96.53	0.761	39.33	26.56	0.0001
	Variance	9.33	6.63		5.66	7.59		3.24	4.29		6.18	10.02	
	Max	93	88.89		95	92.86		100	100		50	37.21	

Table 5.13: Comparison Chart of Existing Cloud-based Testing Frameworks

Testing Framework	Testing Approach	Automated Test Data Generation	Parallelization Framework	Test Bed	SUT	Performance Adequacy Criteria	Test Adequacy Criteria	SLA	Pricing Model	Prediction Mechanism	Validation
Yeti	Automated Random Testing	Yes	Apache Hadoop	Amazon EC2	Java.lang	Speedup	Bugs detection	No	No	No	No
VATS	Performance Testing using HPLoadRunner	No	No	Xen	SAP/R3	Service Performance	—	No	No	No	No
D-Cloud	Fault Injection Testing	No	No	QEMU and Eucalyptus	Distributed Application	Cost ; Time	—	No	No	No	No
Cloud-9	Symbolic Execution	Yes	No	Amazon EC2	UNIX utilities	Speedup	Line Coverage	No	No	No	No
GridUnit	Regression Testing	No	No	Grid	JUnit Test case Execution	Speedup	No	No	No	No	No
Joshua	Regression Testing	No	No	Jini	JUnit Test case Execution	Speedup	No	No	No	No	No
Korat	Constraint Testing	Yes	Google MapReduce	No	Google Application	Object Graph Visualization	Constraint satisfaction	No	No	No	No
NMS	Performance Testing	No	No	No	Simulated networks	Scalability; Time	No	No	No	No	No
TaaS	Prototype of Taas Over Cloud	No	No	Cloud	Web Application	Elasticity	No	No	No	No	No
HadoopUnit	Regression Testing	No	Apache Hadoop	Hadoop Local Cluster	JUnit Test case Execution	Speedup	No	No	No	No	No
Linda et al.	Unit Testing	Yes	Apache Hadoop	Hadoop Local Cluster	One Open Source Library	Speedup; Coverage	Line Coverage	No	No	No	No
CISTS The 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	Yes	Yes	Yes	Yes

6

Conclusions & Future Directions

Automatic test data generation remains a fundamental problem in traditional software testing techniques with service providers striving hard to effectively manage large scale jobs, requiring significant computing resources and lengthy execution times. Cloud-based testing provides an opportunity to overcome these hurdles. The investigation of existing cloud-based testing models uncovers the shortcomings like improper resource utilization, lack of trust among clients, lack of transparent pricing model and an unbiased Service Level Agreement (SLA).

This research work is set out to investigate issues related to automatic software test data generation in traditional software testing techniques to advocate the need of moving software testing to cloud. This thesis efficaciously addresses the need to design a framework for cloud-based testing with strong focus on QoS parameters such as fault detection capability, resource utilization, security and cost. Further, the current scenario demands a paradigm shift to a flexible Op-Ex by offering a test data generation strategy that detects maximum number of faults in the software in minimum time and cost. This thesis proposes a software test data generation technique by using hybrid strategy of GA and PSO algorithm with Apache Hadoop MapReduce for parallel implementation in the distributed environment. Further, the framework for testing as a service has been designed, developed and validated in this thesis.

In this final chapter, concluding remarks on this research work have been given by describing the advancements made towards the objectives of the research. This chapter details the outcome of each chapter and later highlights the contributions of the proposed test data generation framework over cloud. Furthermore, it discusses the directions for future research.

6.1 Conclusion

The aim of this research work has been to design and develop a framework for cloud-based test data generation which has been addressed by the proposed framework.

Literature Review as presented in Chapter 2 highlights the existing cloud-based testing frameworks and various automatic software test data generation strategies. After an exhaustive review, it becomes apparent that academic research in this area is scarce and no existing cloud-based testing model addresses the complete concerns of clients as well as service providers with respect to all the necessary QoS parameters. Furthermore, software test data generation is an undecidable problem and it is termed as NP Complete problem because of its size. The complexity of software is measured in terms of its non-linear structure resulting from control structures such as if-statements and loops. Therefore, simple search strategies may not be sufficient. It was concluded through literature that such problems can be efficiently solved by utilizing soft computing techniques such as Genetic Algorithms and PSO algorithm.

Despite of the above efforts, application of soft computing techniques is limited in software industry. Based on these observations, automatic test data generation strategy and cloud-based test data generation technique have been proposed and designed in Chapter 3. The mathematical model to produce optimal test data has been described in this chapter. The main goal of the framework is to test computer programs that ensures maximum branch coverage and fault detection in minimum time and cost. The proposed strategy has been designed and implemented by using PSO and Genetic Algorithm in which gbest and pbest particles are computed using the concept of pareto optimality. Optimization technique to optimize branch coverage and fault detection of SUT through a fitness function has been discussed in detail in this chapter. The challenge is to ensure the safety of code and data uploaded by developers, tester and other users. In addition to it, need of transparent pricing model is required to establish the trust between the parties. To address these issues, the framework of testing as a service has been proposed in Chapter 4.

A mathematical model of the framework has been described in this chapter. The QoS requirements of both resource providers and resource consumers have been addressed by postulating service request model. The architecture overview, features and goal of the proposed framework have been presented in this chapter. The design of security and pricing model have also been designed in the same chapter. Chapter 5 provides the verification details of the proposed framework while it goes through different phases of the research. The framework has also been validated by comparing it with existing frameworks. The experimental results have been shown and the solution has been presented as a novel approach to address cloud based testing challenges. The proposed algorithm has been evaluated empirically and percentage increase in APFD score, percentage increase in coverage per unit time and the percentage decrease in time have been observed while using the same allotted computational resources. Statistical analysis

of the results has also been done in this chapter.

The contributions of this thesis are as follows:

(i) An exhaustive review of the existing work in the area of cloud-based testing and classification into seven categories has been done on the basis of testing technique adopted by various researchers.

(ii) Exploration of areas focussed by existing researchers, gaps and untouched areas of cloud-based testing.

(iii) Sequential automatic test data generation strategy using the concept of PSO and Genetic Algorithm has been designed.

(iv) Hadoop Mapreduce based automatic test data generation strategy has been devised that facilitates optimal generation of test suites for SUT.

(v) Framework of testing as a service with strong focus on QoS parameters such as fault detection capability, resource utilization, security and cost has been designed.

(vi) The QoS is promised by the service request model which postulates Service Level Agreements (SLA) between the customer and service providers.

(vii) Inculcating of prediction mechanism that prompts users of most valid and appropriate Hadoop and cluster configuration for best performance in minimum time and cost.

(viii) Implementation of security techniques at various layers and provision of role based access control. It ensures the protection of data and code of different users.

(ix) Implementation of transparent cost model for the benefits of clients and service providers.

6.2 Future Directions

In this thesis, the need of migration of software testing to cloud has been discussed. It also introduces cloud-based framework for testing as a service and the underlying soft computing based test data generation strategy. The proposed framework has given emphasis on QoS parameters such as fault detection rate, branch coverage, response time and resources used. Additionally, QoS is promised by service request model which postulates Service Level Agreements (SLA) between the customers and service providers. It demonstrated benefits of the proposed framework and algorithm in terms of user's QoS satisfaction and provider's profit function. However, the work can be further enhanced and extended in future. Some of the future directions are:

- The framework can be further enhanced by introducing system and integration testing after performing unit testing of SUT.

- The existing solutions have been designed and tested only for a single SUT for automatic test data generation at a time. In future, this can be extended for accommodating multiple SUTs from the user at the same time.
- It will be interesting to enhance the framework to support different types of testing such as Black-Box Testing, Combinatorial Testing and Regression Testing.
- The proposed framework in this thesis can also be enhanced by implementing and analysing different security models to select the best model that successfully establishes the trust among users.
- Designing and implementation of transparent and strategic pricing model can be carried out to effectively build trust of the clients.
- Test data generation strategy can be further extended by automatic generation of assertions in the JUnit file with the help of test oracles.
- The framework can be further enhanced by using the checked coverage as test adequacy criteria.
- Framework can also be extended for testing of services that are provided to users available on the fly.

Bibliography

[1] P. Mell and T. Grance. NIST definition of cloud computing. National Institute of Standards and Technology, 2011. Available from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf[last Accessed October 2015].

[2] B. Furht, "CloudComputing Fundamentals," in Handbook of Cloud Computing (B. Furht and A. Escalante, eds.), pp. 3-19, Springer US, 2010.

[3] H. Jin, S. Ibrahim, L.Qi, H. Cao, S, and X. Shi, "Tools and Technologies for Building Clouds," in Cloud Computing: Principles, Systems and Applications (N. Antonopoulos and L. Gillam, eds.), pp. 3-20, Springer, 2010.

[4] A. Talukder, L. Zimmerman, and H. Prahalad, "Cloud Economics: Principles, Costs and Benefits," in Cloud Computing: Principles, Systems and Applications (G. N. and eds L.,eds.), pp. 343-360, Springer, 2010.

[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu:, "Cloud Computing and Grid Computing 360-Degree Compared," in Grid Computing Environments Workshop, pp. 1-10, GCE'08, 2008.

[6] M. A. Murphy, L. Abraham, M. Fenn, and S. Goasguen, "Autonomic Clouds on the Grid.," Journal of Grid Computing, vol. 8, no. 2, pp. 1-18, 2010.

[7] W. Voorsluys, J. Broberg, and R. Buyya, CloudComputing: Principles and Paradigms. Wiley Press, New York, USA 2011. Introduction to Cloud Computing.

[8] S. Kawasaki, M. Niwa, T. Kamina, D.-A. Wu, H. Murakami, and M. Ohashi, "A Study on Formulation of the Ubiquitous Cloud Model," in Mobile Data Management, 2006. MDM 2006. 7th International Conference on, pp. 148, May 2006.

[9] L.J. Zhang and Q. Zhou, "CCOA: Cloud Computing Open Architecture," inWeb Services, 2009. ICWS 2009. IEEE International Conference on, pp. 607-616, july 2009.

[10] O. C. M. Group, "Open Cloud Manifesto," September 2011. Available at http://www.opencloudmanifesto.org/Open%20Cloud%20Manifesto.pdf [Last Accesed: September 2016].

[11] M. Ali, "Green Cloud on the Horizon," in Cloud Computing (M. Jaatun, G. Zhao, and C. Rong, eds.), vol. 5931 of Lecture Notes in Computer Science, pp. 451-459, Springer Berlin / Heidelberg, 2009.

[12] J. M. Nick, D. Cohen, and B. S. Kaliski, "Key Enabling Technologies for Virtual Private Clouds," in Handbook of Cloud Computing (B. Furht and A. Escalante, eds.), pp. 47-63, Springer US, 2010.

[13] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in Cloud Computing (M. Jaatun, G. Zhao, and C. Rong, eds.), vol. 5931 of Lecture Notes in Computer Science, pp. 626-631, Springer Berlin / Heidelberg, 2009.

[14] Q. Xue, X. Zhou, and J. Ma, "A preliminary Study of Spatial Cloud Computing," in Computer Application and System Modeling (ICCASM), 2010 International Conference on, vol. 8, pp. 576-581, oct. 2010.

[15] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy, Survey, and Issues of Cloud Computing Ecosystems," in Cloud Computing (N. Antonopoulos and L. Gillam, eds.), vol. 0 of Computer Communications and Networks, pp. 21-46, Springer London, 2010.

[16] D. Oliveira, F. A. Baico, and M. Mattoso, "Towards a Taxonomy for Cloud Computing from an e-Science Perspective," in Cloud Computing (N. Antonopoulos and L. Gillam, eds.), vol. 0 of Computer Communications and Networks, pp. 47-62, Springer London, 2010.

[17] L. Youseff, M. Butrico, and D. Da Silva, "Toward a Unified Ontology of Cloud Computing," in Grid Computing Environments Workshop, 2008. GCE'08, pp. 1-10, Nov. 2008.

[18] J. Voas and J. Zhang, "Cloud computing: New wine or just a new bottle?," IT Professional, vol. 11, no. 2, pp. 15-17, 2009.

[19] SIEGELE, L. Let It Rise: A Special Report on Corporate IT. The Economist (October 2008).

[20] Washington Post Case Study: Amazon Web Services [online]. Available from: http://aws.amazon.com/solutions/case-studies/washington-post/.

[21] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. In OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation (Berkeley, CA, USA, 2004), USENIX Association, pp. 10-10.

[22] BIALECKI, A., CAFARELLA, M., CUTTING, D., AND O'MALLEY, O. Hadoop: a framework for running applications on large clusters built of commodity hardware. Wiki at http://lucene. apache. org/hadoop.

[23] RUBiS: Rice University Bidding System. http://rubis.ow2.org/index.html

[24] MediaWiki. http://www.mediawiki.org.

[25] Ang Li, Xuanran Zong, Ming Zhang, Srikanth Kandula and Xiaowei Yang, "CloudProphet: towards application performance prediction in cloud": ACM SIGCOMM Computer Communication Review - SIGCOMM'11, Volume 41(4), pp 426-427, August 2011.

[26] Montage. http://montage.ipac.caltech.edu

[27] Taylor, I., Deelman, E., Gannon, D. and Shields, M. eds. (2006).Workflows in e-Science, Springer.

[28] Stevens, R. D., Robinson, A. J. and Goble, C. A. (2003). myGrid: per sonalised bioinformatics on the information grid. In Bioinformatics (11th International Conference on Intelligent Systems for Molecular Biology), vol. 19.

[29] Oinn, T., Li, P., Kell, D. B., Goble, C., Goderis, A., Greenwood, M.,Hull, D., Stevens, R., Turi, D. and Zhao, J. (2006). Taverna/myGrid: aligning a workflow system with the life sciences community. In Workflows in e-Science, edited by I. Taylor, E. Deelman, D. Gannon and M. Shields.Springer.

[30] Deelman, E., Callaghan, S., Field, E., Francoeur, H., Graves, R., Gupta, N., Gupta, V., Jordan, T. H., Kesselman, C., Maechling, P., Mehringer, J., Mehta, G., Okaya, D., Vahi, K. and Zhao, L. (2006a). Managing large-scale workflow execution from resource provisioning to provenance tracking: the CyberShake example. In E-SCIENCE '06: Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing, p. 14.

[31] Brown, D. A., Brady, P. R., Dietz, A., Cao, J., Johnson, B. An McNabb, J. (2006). A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis. In Workflows for e-Science, edited by I. Taylor, E. Deelman, D. Gannon and M. Shields, Springer.

[32] Piccoli, L. (2008). Lattice QCD workflows: a case study. In SWBES08: Challenging Issues in Workflow Applications, Indianapolis, IN.

[33] Google AppEngine https://developers.google.com/appengine

[34] Amazon Web Services aws.amazon.com/

[35] Microsoft Azure www.windowsazure.com

[36] Enki http://www.enki.co/

[37] XCalibre FlexiScale www.flexiscale.com

[38] RackSpace www.rackspace.com

[39] RightScale www.rightscale.com/

[40] Terremark Worldwide www.terremark.com

[41] Engine Yard Cloud www.engineyard.com

[42] McMinn P. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 2004; 14(2):105-156.

[43] M. J. Harrold. Testing: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. ACM: New York, 2000; 61-72.

[44] "IEEE Standard Glossary of Software Engineering Terminology," tech. rep., 1990.

[45] S. Ahamad. Studying the Feasibility and Importance of Software Testing: An Analysis. ETRI Journal, 2009; 1(3):119-128.

[46] Myers J. Glenford, The Art of Software Testing. New York: Wiley, 2nd ed., 2006.

[47] AppLabs, "Approach to Cloud Testing." Applabs Whitepaper. Available at http://www.applabs.com/html/ [Last Accesed: May2016].

[48] V. K. Mylavarapu, "Taking testing to the cloud." Cognizant Whitepaper. Available at www.cognizant.com/taking-testing-to-the-cloud [Last Accesed: December 2015].

[49] V. K. Mylavarapu, "Leveraging cloud capabilities for product testing." Impetus Whitepaper. Available at www.impetus.com [Last Accesed: December 2015].

[50] TUI, "Cloud testing." TUI Infotec Whitepaper. Available at http://www.tui-infotec.com/global. [Last Accesed: December 2015].

[51] M.Prasanna, S. Sivanandam, R.Venkatesan, and R.Sundarrajan, "A Survey On Automatic Test Case Generation," Academic Open Internet Journal, vol. 15, no. 0, 2005. Available at http://www.acadjournal.com/2005/v15/part6/p4/ [Last Accessed : September 2011].

[52] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," Software Engineering, IEEE Transactions on, vol. PP, no. 99, pp. 1-31, 2011.

[53] Rakesh Kumar and Hardeep Singh. 2013. "A qualitative analysis of effects of security risks on architecture of an information system". SIGSOFT Softw. Eng. Notes 38, 6 (November 2013), 1-3.

[54] M. E. Delamaro, J. C. Maldonado, A. Pasquini, and A. P. Mathur, "Interface Mutation Test Adequacy Criterion: An Empirical Evaluation," Empirical Software Engineering, vol. 6, pp. 111-142, 2001.

[55] M. Delamaro, J. Maidonado, and A. Mathur, "Interface Mutation: An Approach for Integration Testing," Software Engineering, IEEE Transactions on, vol. 27, pp. 228-247, mar 2001.

[56] S. S. Batth, E. R. Vieira, A. Cavalli, and M. U. Uyar, "Specification of Timed EFSM Fault Models in SDL," in Proceedings of the 27th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems, FORTE'07, (Berlin, Heidelberg), pp. 50-65, Springer-Verlag, 2007.

[57] M. Delamaro, J. Maidonado, and A. Mathur, "Interface Mutation: An Approach for Integration Testing," Software Engineering, IEEE Transactions on, vol. 27, pp. 228-247, March 2001.

[58] M. Trakhtenbrot, "New Mutations for Evaluation of Specification and Implementation Levels of Adequacy in Testing of Statecharts Models," in Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, pp. 151-160, September 2007.

[59] G. Fraser and F. Wotawa, "Mutant Minimization for Model-Checker Based Test-Case Generation," in Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, (Washington, DC, USA), pp. 161-168, IEEE Computer Society, 2007.

[60] S. D. R. S. De Souza, J. C. Maldonado, S. C. P. F. Fabbri, and W. L. De Souza, "Mutation Testing Applied to Estelle Specifications," Software Quality Control, vol. 8, pp. 285-301, December 1999.

[61] S. C. P. F. Fabbri, J. C. Maldonado, P. C. Masiero, M. E. Delamaro, and E. Wong, "Mutation Testing Applied to Validate Specifications Based on Petri Nets," in Proceedings of the IFIP TC6 Eighth International Conference on Formal Description Techniques VIII, (London, UK, UK), pp. 329-337, Chapman & Hall, Ltd., 1996.

[62] G. Vigna, W. Robertson, and D. Balzarotti, "Testing Network-Based Intrusion Detection Signatures using Mutant Exploits," in Proceedings of the 11th ACM conference on Computer and communications security, CCS'04, (New York, NY, USA), pp. 21-30, ACM, 2004.

[63] E. Martin and T. Xie, "A Fault Model and Mutation Testing of Access Control Policies," in Proceedings of the 16th international conference on World Wide Web, WWW'07, (New York, NY, USA), pp. 667-676, ACM, 2007.

[64] W. Xu, J. Offutt, and J. Luo, "TestingWeb Services by XML Perturbation," in Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, (Washington, DC, USA), pp. 257-266, IEEE Computer Society, 2005.

[65] C. Boyapati, S. Khurshid, and D. Marinov, "Korat: Automated Testing based on Java Predicates," SIGSOFT Softw. Eng. Notes, vol. 27, pp. 123-133, July 2002.

[66] Yeo, C.S., Buyya, R. Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. In *Proceedings of the 7th IEEE International Conference on Cluster Computing* (Cluster 2005), Bostan, MA, USA.

[67] Rana, O. F., Warnier, M., Quillinan, T. B., Brazier, F., Cojocarasu, D. Managing Violations in Service level agreements. In *proceedings of the 5th International Workshop on Grid Economics and Business Models*(GenCon 2008), Gran Canaris, Spain; 349-358.

[68] Irwin, D.E., Grit, L.E., Chase, J.S. Balancing Risk and Reward in a Market-based Task Service. In *Proceedings of the 13th International Symposium on High Performance Distributed Computing* (HPDC 2004), Honolulu, HI, USA.

[69] G. Cabral and A. Sampaio, "Formal Specification Generation from Requirement Documents," Electron. Notes Theor. Comput. Sci., vol. 195, pp. 171-188, January 2008.

[70] M. V. Arkhipova and S. V. Zelenov, "Directed Generation of Test Data for Static Semantics Checker," in Leveraging Applications of Formal Methods, Verification and Validation (T. Margaria and B. Steffen, eds.), vol. 17 of Communications in Computer and Information Science, pp. 753-768, Springer Berlin Heidelberg, 2009.

[71] C. Nebut, F. Fleurey, Y. Le Traon, and J.-M. Jezequel, "Automatic Test Generation: A use case driven Approach," Software Engineering, IEEE Transactions on, vol. 32, pp. 140-155, march 2006.

[72] A. Nayak and D. Samanta, "Synthesis of Test Scenarios using UML Activity Diagrams," Software and Systems Modeling, vol. 10, pp. 63-89, 2011.

[73] A. Jaskelainen, A. Kervinen, M. Katara, A. Valmari, and H. Virtanen, "Synthesizing Test Models from Test Cases," in Proceedings of the 4th International Haifa Verification Conference on Hardware and Software: Verification and Testing, HVC'08, (Berlin, Heidelberg), pp. 179-193, Springer-Verlag, 2009.

[74] M. Riebisch, I. Philippow, and M. GÃűtze, "UML-Based Statistical Test Case Generation," in Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, NODe'02, (London, UK, UK), pp. 394-411, Springer-Verlag, 2003.

[75] Wu L, Garg SK, Buyya R. SLA-based admission control for a software-as-a-service provider in cloud computing environments. Journal of Computer and System Sciences, 2012; 78(5):1280-1299.

[76] S. Ali, L. Briand, H. Hemmati, and R. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation," Software Engineering, IEEE Transactions on, vol. 36, pp. 742 -762, nov.-dec. 2010.

[77] J. Clarke, J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating Software Engineering as a Search Problem," Software, IEEE Proceedings, vol. 150, pp. 161-175, June 2003.

[78] R. Drechsler and N. Drechsler, Evolutionary Algorithms for Embedded System Design. New York: Kluwer Academic Publishers, first ed., 2002.

[79] D. A. Coley, An Introduction to Genetic Algorithms for Scientists and Engineers. New York: World Scientific Publishing Company, first ed., 1997.

[80] W. Afzal, R. Torkar, and R. Feldt, "A Systematic Review of Search-based Testing for Non-functional System Properties," Information Software Technology, vol. 51, pp. 957-976, June 2009.

[81] T. Mantere and J. T. Alander, "Evolutionary Software Engineering : a Review," Applied Software Computing, vol. 5, pp. 315-331, March 2005.

[82] Nitin Kumar, C. Ghanshyam, and Ajay K. Sharma. 2015. "Effect of multi-path fading model on T-ANT clustering protocol for WSN". Wirel. Netw. 21, 4 (May 2015), 1155-1162.

[83] Jacaco Web Page. Available from http://www.eclemma.org/jacoco/[last Accessed October 2015].

[84] Apache Hadoop Map Reduce. Available from http://www.hadoop.apache.org/mapreduce [last Accessed October 2015].

[85] Banzai T, Koizumi H, Kanbayashi R, Imada T, Hanawa T, Sato M. 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. IEEE Computer Society: Washington, DC, USA, 2010; 631-636.

[86] Gaisbauer S, Kirschnick J, Edwards N, Rolia J. VATS: Virtualized-Aware Automated Test Service. In *Proceedings of Fifth International Conference on Quantitative Evaluation of Systems*, QEST'08. St Malo, France, 2008;93-102.

[87] S. Bucur, V. Ureche, C. Zamfir, G. Candea. Parallel Symbolic Execution for Automated Real-World Software Testing. In *Proceedings of the sixth conference on Computer systems,* EuroSys'11. ACM: New York, NY, USA, 2011; 183-198.

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

[89] Alexey Lastovetsky. Parallel testing of distributed software.*Inf. Softw. Technol.*2005; 47(10):657-662.

[90] M. Oriol, F. Ullah. Yeti on the Cloud. In *Third International Conference on Software Testing, Verification, and Validation Workshops* (ICSTW), 2010. IEEE Computer Society Washington, DC, USA, 2010; 434-437.

[91] L. Yu, W. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, W. Zhao. Testing as a Service over Cloud. In *Fifth IEEE International Symposium on Service Oriented System Engineering*. IEEE: Nanjing, 2010; 181âĂŞ188.

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

[93] S.R. Balasundaram and B. Ramadoss. Issues of information communication technology (ICT) in education. Kanishka Publishers Distributors, New Delhi, INDIA, 2009.

[94] Nabil Sultan. Cloud Computing for education: A new Dawn? International Journal of Information Management, 30(2):109-116, 2010.

[95] Tuncay Ercan. Effective use of cloud computing in educational instututions. Procedia-Social and Behavioral Sciences, 2(2):938-942,2010. Innovation and Creativity in Education.

[96] Ctlin Boja, Paul Pocatilu and Cristian Toma. The economics of cloud computing on educational services. Procedia - Social and Behavioral Sciences, 93(0);1050-1054, 2013. 3rd World Conference on learning, Teaching and Educational Leadership.

[97] Xun Xu. From cloud computing to cloud manufacturing. Robotics and Computer Intergrated Manufacturing, 28(1):75-86, 2012.

[98] Yuanjun Laili, Fei Tao, Lin Zhang and Bhaba R Sarkar. A study of optimal allocation of computing resources in cloud manufacturing systems. The International Journal of Advaced Manufacturing Technology, 63(5-8):671-690, 2012.

[99] Wu He and Lida Xu. A state-of-the-art survey of cloud manufacturing. International Journal of Computer Integrated Manufacturing, 0(0);1-12,0.

[100] Fei Tao, ying Cheng, Li da Xu, Lin Zhang and Bo Hu Li. CCIoT-CMfg: Cloud computing and internet of things-based cloud manufacturing service system . Industrial Informatics, IEEE Transaction on, 10(2);1435-1442, May 2014.

[101] 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.

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

[103] Duarte A, Wagner G, Brasileiro F, Cirne W. Multienvironment software testing on the grid. In *Proceedings of the 2006 workshop on Parallel and distributed systems: testing and debugging*. ACM:USA, 2006;61-68.

[104] Raman Kumar, Harsh Kumar Verma, and Renu Dhir. 2015. Analysis and Design of Protocol for Enhanced Threshold Proxy Signature Scheme Based on RSA for Known Signers. Wirel. Pers. Commun. 80, 3 (February 2015), 1281-1345.

[105] Parveen T, Tilley S, Daley N, Morales P. Towards a distributed execution framework for JUnit test cases. In *Proceedings of International Conference on Software Maintenance, 2009.* ICSM 2009. IEEE: USA, 2009;425-428.

[106] A. Wahid, X. Gao, and P. Andreae, "Multi-view clustering of web documents using multi-objective genetic algorithm," in Evolutionary Computation (CEC), 2014 IEEE Congress on. IEEE, 2014, pp. 2625-2632.

[107] Geronimo LD, Ferrucci F, Murolo A, Sarro F. 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.

[108] Ali S, Briand LC, Hemmati H, Panesar-Walawege RK. A systematic review of the application and empirical investigation of search-based test-case generation. IEEE Transactions on Software Engineering, vol. 99, no. PrePrints, 2009.

[109] Amazon Web Service,http://www.aws.org/instance [last Accessed October 2015].

[110] Amazon Web Service,http://www.aws.org/instance/price [last Accessed October 2015].

[111] 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 Accessed October 2015].

[112] H. Herodotou and S. Babu On Optimizing MapReduce Programs Hadoop Jobs. In *Proceedings of the the 38th International Conference on Very Large Data Bases* (VLDB'12), August 2012.

[113] Gartner Report,http://www.gartner.com/newsroom/id/2613015 [last Accessed October 2015].

[114] Merill Lynch Report,http://readwrite.com/2009/11/25/merrill-lynch-cloud-computing [last Accessed October 2015].

[115] S. Martino Di, Ferrucci F, Maggio V, Sarro F. Towards migrating genetic algorithms for test data generation to the cloud. IGI Global, 2012, ch. 6, pp. 113-135.

[116] Fraser G, Arcuri A. Evolutionary Generation of Whole Test Suites. In *11th International Conference on Quality Software*(QSIC) 2011; 31-40.

[117] Tonella P. Evolutionary testing of classes. In *Proceedings of the International Symposium on Software Testing and Analysis*(ISSTA),2004; 119-128.

[118] Ravi Sandhu, David Ferraioli, and Richard Kuhn.The NIST model for role-based access control:towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control* (RBAC'00). ACM:New York,NY,USA: 47-63.

[119] Syed Rizvi, Katie Cover, Christopher gates. A Trusted Third-party (TTP) based Encryption Scheme for Ensuring Data Confidentiality in Cloud Enviornment. Procedia Computer Science, 2014;36:381-386.

[120] Vijay Kumar, Arun Sharma, Rajesh Kuma, "Applying Soft Computing Approaches to Predict Defect Density in Software Product Releases: An Empirical Study". Computing and Informatics 32(1): 203-224 (2013).

[121] Patel H, Patel D. A Review of Approaches to Achieve Data Storage Correctness in Cloud Computing Using Trusted Third Party Auditor. In *International Symposium on Cloud and Services Computing* (ISCOS), 2012:84-87.

[122] Maenhaut PJ, Moens H, Ongenae V and De Turck F. Migrating legacy software to the cloud: approach and verification by means of two medical software use cases. Software: Practice and Experience 2015. DOI: 10.1002/spe.2320.

[123] Ferrer J, Chicano F, Alba E. Evolutionary algorithms for the multi-objective test data generation problem. Software Practice Experience November 2012;42(11):1331-1362.

[124] 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.

[125] 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.

[126] 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

[127] 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.

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

[129] 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.

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

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

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

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

[134] 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.

[135] 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.

[136] 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.

[137] Jaca Web Page *http://www.ic.unicamp.br/ eliane/JACA.html*.

[138] Pargas R P, Harrold M J, Peck R R. Test-data generation using genetic algorithms. Software Testing Verification Reliability, 1999, 9(4): 263-282.

[139] Prateek Pandey, Shishir Kumar, Sandeep Shrivastava,"A fuzzy decision making approach for analogy detection in new product forecasting." Journal of Intelligent and Fuzzy Systems 28(5): 2047-2057 (2015).

[140] Wegener J, Baresel A, Sthamer H. Evolutionary test environment for automatic structural testing. Information and Software Technology, 2001, 43(14): 841-854.

[141] Jones B F, Sthamer H, Eyres D E. Automatic test data generation using genetic algorithms.Software Engineering Journal, 1996, 11(5): 299-306.

[142] 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.

[143] Harman M, Jones B. Search-based software engineering. Information and Software Technology, 2001 43(14): 833-839.

[144] 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.

[145] C4J Web Page *http://c4j-team.github.io/C4J/index.html*

[146] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multiobjective optimization test problems". In *Proc. of Congress on Evolutionary Computation*, 2002

[147] 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

[148] 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

[149] A. Verma, X. Llora, 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.

[150] C. Jin, C. Vecchiola, R. Buyya: MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms.*eScience* 2008: 214-221

[151] E. Starkloff, "Designing a parallel, distributed test system", Aerospace and Electronic Systems Magazine, IEEE, vol. 16, no. 6, pp. 3-6, jun 2001.

[152] 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.

[153] 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.

[154] 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.

[155] Ramler R, Wolfmaier K. Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. In: Proceedings of the International Workshop on Automation of Software Test. 2006, 85-91.

[156] Grottke M, Trivedi K S. 2007. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. IEEE Computer, 2007, 40(2): 107-109.

[157] A. Weiss,"Computing in the clouds", netWorker, vol. 11, pp. 16-25, Dec. 2007.

[158] A. Inc, "Cloudwatch: Monitoring for aws cloud resources". Available at http://aws.amazon.com/cloudwatch/ [Last Accesed: Dec 2011].

[159] G. Inc., "google apps: Reliable, secure, online applications". Available at http://apps.google.com [Last Accesed: Dec 2011].

[160] M. Corporation, "microsoft windows azure platform:operating system as an online service". Available at http://www.microsoft.com/windowsazure/ [Last Accesed: Dec 2011].

[161] G. Research, "gartner research, june 2007"." ID number: G00148987".

[162] IBM, "Ibm testing services for cloud application virtualization overview ibm". IBM Whitepaper. Available at http://www-935.ibm.com/services/us/en/business-services. [Last Accesed: Dec 2012].

[163] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC, NCM'09, (Washington, DC, USA), pp. 44-51, IEEE Computer Society, 2009.

[164] S. Joglekar, "A foray into cloud-based software testing". Patni Whitepaper. Available at www.igatepatni.com/ [Last Accesed: Dec 2012].

[165] AppLabs, "Testing the cloud." Applabs Whitepaper. Available at http://www.applabs.com/html/ [Last Accesed: December 2011].

[166] L. van der Aalst, "Leveraging cloud capabilities for product testing". Impetus Whitepaper. Available at www.impetus.com/Home/Downloads [Last Accesed: September 2011].

[167] P. Mobile, "Top 10 reasons why enterprises should adopt a cloud-based approach for mobile application testing". Perfecto Mobile Whitepaper. Available at www.perfectomobile.com/cloudbasedapproach.pdf [Last Accesed: December 2011].

[168] L. G. Briefing, "Testing applications in cloud." CSC Whitepaper. Available at http://www.assets1.csc.com [Last Accesed: December 2011].

[169] V. K. Mylavarapu, "Leveraging cloud capabilities for product testing." Impetus Whitepaper. Available at www.impetus.com [Last Accesed: December 2011].

[170] G. Candea, S. Bucur, and C. Zamfir, "Automated Software Testing as a Service" in Proceedings of the 1st ACM symposium on Cloud computing, SoCC '10, (New York, NY, USA), pp. 155-160, ACM, 2010.

[171] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," vol. 2, no. EBSE 2007-001, p. 65, 2007.

[172] M. Petticrew and H. Roberts, "Systematic Reviews in the Social Sciences: A Practical Guide", vol. 54. Wiley-Blackwell, 2006.

[173] T. Dyba, "Applying systematic reviews to diverse study types : An experience report," Search, no. 7465, pp. 225-234, 2007.

[174] Spirent, "The ins and outs of cloud computing and its impact on the network." Spirent Whitepaper. Available at: http://www.spirent.com/media/White[Last Accesed:Dec 2011].

[175] T. Parveen and S. Tilley, "When to Migrate Software Testing to the Cloud?" in Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW '10, (Washington, DC, USA), pp. 424-427, IEEE Computer Society, 2010.

[176] J. Gao, X. Bai, and W.-T. Tsai, "Cloud testing-issues, challenges, needs and practices," Software Engineering: An International Journal, vol. 1, pp. 9-23, September 2011.

[177] L. M. Riungu, O. Taipale, and K. Smolander, "Research issues for software testing in the cloud," in Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CLOUDCOM '10, (Washington, DC, USA), pp. 557-564, IEEE Computer Society, 2010.

[178] L. van der Aalst, "Software testing as a service STaaS." Sogeti Whitepaper. Available at www.sogeti.com/staas[Last Accesed: March 2010].

[179] W. Chan, S. Cheung, and K. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," Int. J. Web Service Res., vol. 4, no. 2, pp. 61-81, 2007.

[180] S. Nag, "Business case for cloud based-testing."BlueStar Whitepaper. Available at http://www.bsil.com/Resource-Center/ [Last Accesed: Dec 2012].

[181] L. M. Riungu, O. Taipale, and K. Smolander, "Software testing as an online service: Observations from practice," in Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW '10, (Washington, DC, USA), pp. 418-423, IEEE Computer Society, 2010.

[182] S. Ghag, "Software validations of application deployed on windows azure." Infosys Whitepaper. Available at www.infosys.com/cloud/ [Last Accesed:Dec 2011].

[183] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations," in Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, SECLOUD '11, (New York, NY, USA), pp. 50-56, ACM, 2011.

[184] K. Priyadarsini, "Cloud testing as a service,"IJAEST, vol. 6, pp. 173-177, September 2011.

[185] G. Moorthy, "Test automation framework for soa applications." Infosys Whitepaper. Available at isqtinternational.com/Ganesh[Last Accesed: December 2011].

[186] R. Mansharamani, "Virtual production environments for software development and testing." TCS Whitepaper. Available at http://www.tcs.com/SiteCollectionDocuments [Last Accesed: December 2011].

[187] E. Roodenrijs, "Testing on the cloud." Sogeti Whitepaper. Available at http://www.isqtinternational.com [Last Accesed: December 2011].

[188] J. Varia, "Migrating your existing applications to the aws cloud." Amazon Web Services Whitepaper. Available at http://www.media.amazonwebservices.com [Last Accesed:December 2011].

[189] Siemens, Sitemppo."Siemens Whitepaper". Available at https://www.cee.siemens.com/SiTEMPPO [Last Accesed: December 2011].

[190] J. Janeczko, "Journey into the cloud." TUI Whitepaper. Available at http://www.tui-infotec.com/global [Last Accesed: December 2011].

[191] CSS, "Cloud-testgo-cloud based performance testing." CSS Whitepaper. Available at csscorp.com/services/cloud-services [Last Accesed: December 2011].

[192] Wipro, "Wipro's cloud testing service." Wipro Whitepaper. Available at taas.wipro.com/ [Last Accesed: December 2011].

[193] P. Ashwood, "Why your it organization should move from traditional application testing to testing-as-a-service." HP Whitepaper. Available at http://h20195.www2.hp.com/ [Last Accesed: December 2011].

[194] Veracode, "Agile security." Veracode Whitepaper. Available at http://www.espiongroup.com [Last Accesed: December 2011].

[195] J. Michelsen, "Service virtualization and the devtest cloud." iTko Whitepaper. Available at http://www.itko.com/resources/ [Last Accesed: December 2011].

[196] IBM, "Development and testing using cloud computing." IBM Whitepaper. Available at http://www-935.ibm.com/services [Last Accesed: December 2011].

[197] SauceLabs, "onDemand Cloud Testing Tool." Available at http://saucelabs.com/ [Last Accesed: September 2011].

[198] Skytap, "SkyTap Cloud Testing Tool." Available at http://skytap.com/ [Last Accesed: September 2011].

[199] uTest, "uTest Cloud Testing Tool." Available at http://utest.com/ [Last Accesed: September 2011].

[200] VMLogix, "VMLogix LabManager Cloud Testing Tool." Available at http://vmlogix.com/ [Last Accesed: September 2011].

[201] SOASTA, "Software testing for startups." SOASTA Whitepaper. Available at http://www.cloudconnectevent.com/downloads/ [Last Accesed: December 2011].

[202] RackSpace, "Test and dev cloud." RacKSpace Whitepaper, December 2011. Available at http://www.rackspace.com/testdev/ [Last Accesed: December 2011].

[203] H. Le, "Testing as a service for component-based developments," in Proceedings of The Third International Conference on Advances in System Testing and Validation Lifecycle, VALID 2011, pp. 46-51, IARIA, 2011.

[204] W. Chan, L. Mei, and Z. Zhang, "Modeling and testing of cloud applications," in Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pp. 111-118, dec. 2009.

[205] A. Jaaskelainen, M. Katara, A. Kervinen, H. Heiskanen, M. Maunumaa, and T. Paakkonen, "Model-based testing service on the web," in Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems: 8th International Workshop, TestCom '08 / FATES '08, (Berlin, Heidelberg), pp. 38-53, Springer-Verlag, 2008.

[206] CSS, "Cloud-based Performance Testing." CSS Whitepaper. Available at csscorp.com/services/cloud-services [Last Accesed: December 2011].

[207] M. A. S. Netto, S. Menon, H. V. Vieira, L. T. Costa, F. M. de Oliveira, R. Saad, and A. Zorzo, "Evaluating load generation in virtualized environments for software performance testing," in Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW'11, (Washington, DC, USA), pp. 993-1000, IEEE Computer Society, 2011.

[208] Z. Ganon and I. E. Zilbershtein, "Large-scale performance tests of network management systems,"

[209] CSS, Performance engineering of web applications. CSS Whitepaper. Available at http://www.csscorp.com/downloads/whitepapers/ [Last Accesed: December 2011].

[210] L. Gu and S.-C. Cheung, "Constructing and testing privacy-aware services in a cloud computing environment: challenges and opportunities," in Proceedings of the First Asia-Pacific Symposium on Internetware, Internetware '09, pp. 2:1-2:10, ACM, 2009.

[211] P. Zech, "Risk-based security testing in cloud computing environments," in Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, ICST'11, (Washington, DC, USA), pp. 411-414, IEEE Computer Society, 2011.

[212] B. Marin, T. Vos, G. Giachetti, A. Baars, and P. Tonella, "Towards testing future web applications," in Research Challenges in Information Science (RCIS), 2011 Fifth International Conference on, pp. 1-12, may 2011.

[213] T. M. King and A. S. Ganti, "Migrating autonomic self-testing to the cloud," in Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW '10, (Washington, DC, USA), pp. 438-443, IEEE Computer Society, 2010.

[214] T. Vengattaraman, P. Dhavachelvan, and R. Baskaran, "A model of cloud based application environment for software testing," CoRR, vol. abs/1004.1773, 2010.

[215] G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su, "Dynamic test input generation for web applications," in Proceedings of the 2008 international symposium on Software testing and analysis, ISSTA '08, (New York, NY, USA), pp. 249-260, ACM, 2008.

[216] 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.

[217] A. Lastovetsky, "Parallel testing of distributed software".Inf. Softw. Technol., vol. 47, pp. 657-662, July 2005.

[218] R. Guha and D. Al-Dabass, "Impact of web 2.0 and cloud computing platform on software engineering," in Proceedings of the 2010 International Symposium on Electronic System Design, ISED '10, (Washington, DC, USA), pp. 213-218, IEEE Computer Society, 2010.

[219] D. Chantry, "Mapping applications to the cloud," The Architecture Journal, vol. 19, pp. 2-9, September 2011.

[220] E. P. Mancini, M. Rak, and U. Villano, "Perfcloud: Grid services for performance-oriented development of cloud computing applications," in Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE '09, (Washington, DC, USA), pp. 201-206, IEEE Computer Society, 2009.

[221] J. Yang, J. Qiu, and Y. Li, "A profile-based approach to just-in-time scalability for cloud applications," in Proceedings of the 2009 IEEE International Conference on Cloud Computing, CLOUD '09, (Washington, DC, USA), pp. 9-16, IEEE Computer Society, 2009.

[222] Y. Khalidi, "Building a cloud computing platform for new possibilities," Computer, vol. 44, pp. 29-34, March 2011.

[223] B. Chhabra, D. Verma, and B. Taneja, "Software engineering issues from the cloud application perspective," International Journal of Information Technology and Knowledge Management, vol. 2, no. 2.

[224] S. Hosono, H. Huang, T. Hara, Y. Shimomura, and T. Arai, "A lifetime supporting framework for cloud applications," in Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD '10, (Washington, DC, USA), pp. 362-369, IEEE Computer Society, 2010.

[225] G. Shroff, Dev 2.0: "model driven development in the cloud," in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, SIGSOFT '08. FSE-16, (New York, NY, USA), pp. 283-283, ACM, 2008.

[226] P. Bhattacharya and I. Neamtiu, "Dynamic updates for web and cloud applications," in Proceedings of the 2010 Workshop on Analysis and Programming Languages for Web Applications and Cloud Applications, APLWACA '10, (New York, NY, USA), pp. 21-25, ACM, 2010.

[227] J. S. Rellermeyer, M. Duller, and G. Alonso, "Engineering the cloud from software modules," in Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, CLOUD '09, (Washington, DC, USA), pp. 32-37, IEEE Computer Society, 2009.

[228] P. Yara, R. Ramachandran, G. Balasubramanian, K. Muthuswamy, and D. Chandrasekar, "Global software development with cloud platforms," Software Engineering Approaches for Offshore and Outsourced Development, pp. 81-95, 2009.

[229] R. Sen, "Developing parallel programs," The Architecture Journal, vol. 19, pp. 17-23, September 2011.

[230] O. Taipale, J. Kasurinen, K. Karhu, and K. Smolander, "Trade-off between Automated and Manual Software Testing," International Journal of Systems Assurance Engineering and Management Springer, pp. 1-12, 2011.

[231] R. Ramler and K. Wolfmaier, "Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost," in Proceedings of the 2006 in-

ternational workshop on Automation of software test, AST '06, (New York, USA), pp. 85-91, 2006.

[232] S. Berner, R. Weber, and R. K. Keller, "Observations and Lessons learned from Automated Testing," in Proceedings of the 27th international conference on Software engineering, ICSE'05, pp. 571-579, 2005.

[233] CSS, "A framework for test automation." CSS Whitepaper. Available at http://www.utest.com/landing-interior/framework [Last Accesed: December 2011].

[234] K. Z. Zamli, N. Ashidi, M. Isa, M. Fadel, and J. Klaib, "A tool for automated test data generation and execution based on combinatorial approach," International Journal of Software Engineering and Its Applications, vol. 1, no. 1, pp. 19-36, 2007.

[235] J. Miller, M. Reformat, and H. Zhang, "Automatic test data generation using genetic algorithm and program dependence graphs," Information and Software Technology, vol. 48, no. 7, pp. 586-605, 2006.

[236] C. Mingsong, Q. Xiaokang, and L. Xuandong, "Automatic test case generation for uml activity diagrams," in Proceedings of the 2006 international workshop on Automation of software test, AST '06, (New York, NY, USA), pp. 2-8, ACM, 2006.

[237] W. G. J. Halfond and A. Orso, "Improving test case generation for web applications using automated interface discovery," 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, (New York, NY, USA), pp. 145-154, ACM, 2007.

[238] Impetus, "Designing a successful test automation strategy: Connecting the dots." Impetus Whitepaper. Available at www.impetus.com [Last Accesed: December 2011]

[239] Manindra Agrawal, S. Akshay, Blaise Genest, P. S. Thiagarajan, "Approximate Verification of the Symbolic Dynamics of Markov Chains". J. ACM 62(1): 2:1-2:34 (2015).

[240] Poonam Gera, Kumkum Garg, Manoj Misra, "Trust-based Multi-Path Routing for Enhancing Data Security in MANETs" I. J. Network Security 16(2): 102-111 (2014).

[241] Chen X, Gu Q, Qi J X, Chen D X. Applying Particle Swarm Optimization to Pairwise Testing. In: Proceedings of 34th Annual IEEE Computer Software and Applications Conference, COMPSAC'10. 2010, 107-116.

[242] Wappler S, Wegener J. Evolutionary Unit Testing Of Object-Oriented Software Using A Hybrid Evolutionary Algorithm. In: Proceedings of IEEE Congress on Evolutionary Computation, CEC'06. 2006, 851-858.

[243] Alba E, Chicano F. Observations in using parallel and sequential evolutionary algorithms for automatic software testing. Computers and Operation Research, 2008, 35(10): 3161-3183.

[244] Li K, Zhang Z, Kou J. Breeding software test data with Genetic-Particle swarm mixed Algorithms. Journal of Computers, 2010, 5(2): 258-265.

[245] Singla S, Kumar D, Rai H M, Singla P. A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts. International Journal of Advanced Science and Technology, 2011, 37: 15-26.

[246] Wu X, Wang Y, Zhang T. An improved GAPSO hybrid programming algorithm. In: Proceedings of International Conference on Information Engineering and Computer Science, ICIECS'09. 2009, 1-4.

[247] Zhang S, Ying Z, Hong Z, Qingquan H. Automatic path test data generation based on GA-PSO. In: Proceedings of IEEE International Conference onIntelligent Computing and Intelligent Systems, ICIS'10. 2010, 142-146.

[248] Chen S. Particle swarm optimization with pbest crossover. In: Proceedings of IEEE Congress on Evolutionary Computation, CEC'12. 2012, 1-6.

[249] Kaur A, Bhatt D. Hybrid particle swarm optimization for regression testing. International Journal on Computer Science and Engineering, 2011, 3 (5): 1815-1824.

[250] Goldberg D E, Holland J H. Genetic Algorithms and Machine Learning. Machine learning, 1988, 3(2): 95-99.

[251] Eberhart R C, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the 6th International Symposium on Micromachine Human Science, pages. 1995, 39-43.

[252] Kennedy J, Eberhart R C. Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. 1995, 4, pp. 1942-1948.

[253] Rabanal P, Rodriguez I, Rubio F. A functional approach to parallelize particle swarm optimization. In: Proceedings of Metaheuristicas, Algoritmos Evolutivos y Bioinspirados, MAEB'12. 2012

[254] Jones B F, Sthamer H, Eyres D E. Automatic test data generation using genetic algorithms.Software Engineering Journal, 1996, 11(5): 299-306.

[255] Hansen N, Finck S, Ros R, Auger A. Real-parameter black-box optimization benchmarking 2009: noiseless functions definitions, INRIA Technical Report RR-6829, 2009.

[256] Younes M, Benhamida F. Genetic Algorithm-Particle Swarm Optimization (GA-PSO) for Economic Load Dispatch. PRZEGLAD ELEKTROTECHNICZNY (Electrical Review), 2011, 87(10): 369-372.

[257] Juang C F. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2004, 34(2): 997-1006.

[258] Saini D K, Sharma Y. Soft Computing Particle Swarm Optimization based Approach for Class Responsibility Assignment Problem. International Journal of Computer Applications, 2012, 40(12): 19-24.

[259] Wappler S, Schieferdecker I. Improving evolutionary class testing in the presence of non-public methods. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ASE'07. 2007, 381-384.

[260] Wilson GC, McIntyre A, Heywood and M I. Resource review: Three open source systems for evolving programs: Lilgp, ecj and grammatical evolution. Genetic Programming and Evolvable Machines, 2004, 5(1): 103-105.

[261] 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.

[262] Wang H C, Jeng B. Structural testing using memetic algorithm. In: Proceedings of the Second Taiwan Conference on Software Engineering. 2006.

[263] Arcuri A, Yao X. Search based software testing of object-oriented containers. Information Sciences, 2008, 178(15): 3075-3095.

[264] Nayak N, Mohapatra D P. Automatic Test Data Generation for data flow testing using Particle Swarm Optimization. Communications in Computer and Information Science, 2010, 95(1): 1-12.

[265] Ahmed B S, Zamli K Z, Lim C P. Constructing a T-Way Interaction Test Suite Using Particle Swarm Optimization Approach. International Journal of Innov. Comput. Inf. Control., 2011, 7(11): 1741-1758.

[266] Li A, Zhang Y. Automatic generating all-path test data of a program based on PSO. In: Proceedings of World Congress on Software Engineering, WCSE'09. 2009, 4, 189-193.

[267] Fraser G, Arcuri A. Whole test suite generation. IEEE Transactions on Software Engineering, 2013, 39(2): 276-291.

[268] Gong D W, Zhang Y. Generating test data for both path coverage and fault detection using genetic algorithms. Frontiers of Computer Science, 2013, 7(6): 822-837.

[269] McMinn P, Holcombe M. Evolutionary testing of statebased programs. In: Proceedings conference on Genetic and evolutionary computation, GECCO'05. 2005, 1013-1020.

[270] Rothermel G, Untch R, Chengyun C, Harrold M J. Prioritizing Test Cases for Regression Testing. IEEE Transaction of Software Engineering, 2001, 27(10): 929-948.

[271] Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of 33rd International Conference on Software Engineering ICSE'11. 2011, 1-10.

[272] Ali S, Briand L C, Hemmati H, Panesar-Walawege R K. 2010. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. IEEE Transactions of Software Engineering, 2010, 36(6): 742-762.

[273] Vargha A, Delaney H D. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics, 2000, 25(2): 101-132.

[274] Grissom R, Kim J. Effect sizes for research: A broad practical approach. Lawrence Erlbaum, 2005.

[275] M. I. Younis and K. Z. Zamli, "MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems," ETRI Journal, vol. 32, no. 0, pp. 73-82, 2010.

[276] B. S. Ahmed and K. Z. Zamli, "T-way Test Data Generation Strategy Based on Particle Swarm Optimization," in Proceedings of the 2010 Second International Conference on Computer Research and Development, ICCRD'10, (Washington, DC, USA), pp. 93-97, IEEE Computer Society, 2010.

[277] P. Joshi, H. S. Gunawi, and K. Sen, "PreFail: a Programmable Tool for Multiple-Failure Injection," in Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications, 2011, pp. 171-188.

[278] H. S. Gunawi, T. Do, P. Joshi, P. Alvaro, J. Yun, J.-s. Oh, J. M.Hellerstein, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, K. Sen, and D. Borthakur, "FATE and DESTINI: A Framework for Cloud Recovery Testing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-127, Sep 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-127.html.

[279] IBM, "CloudBurst Cloud Testing Tool." Available at http://www-304.ibm.com/ [Last Accesed: September 2011].

[280] Zephyr, "Zephyr Cloud Testing Tool," September 2011. Available at at http://Zephyr.com/ [Last Accesed: September 2011].

[281] Singh, V. Kumar, and V. Sharma. "Elitist Genetic Algorithm Based Energy Balanced Routing Strategy to Prolong Lifetime of Wireless Sensor Networks." Chinese Journal of Engineering 2014.

[282] Azad, P. and Sharma, V. (2015). Pareto-optimal clustering scheme using data aggregation for wireless sensor networks. International Journal of Electronics, 102(7), 1165-1176.

A

Appendix

A.1 Introduction to Apache Hadoop MapReduce

MapReduce is the child product given by Google researchers Dean and Ghemawat [131] which as a platform points at better and faster development, deployment of analytical programs moulding massive datasets of google large distributed clusters for computational activities. Their concept was to "map" operation followed by a "reduce" operation. MapReduce as a refined and flexible approach aids in developing large-scale distributed applications. Employing user friendly operability realizes large scale data-processing simply by implementing corresponding interfaces, classes and other operations. As technology it is at core for cloud computing where by utilizing the interfaces high end complex business processing can be distributed for the spontaneous execution.

MapReduce as combination of two distinct functions, Map and Reduce which implements divide and conquer strategy. Map function handles parallelization whereas Reduce collate the results. Specifically, master nodes divide the initial input into several splits each having its own unique key and distributes them to several slave nodes, who either in parallel or independently iterates over their respective received splits. These nodes are referred as Mappers whose output is aggregated via reducer function.

Reducer processes and collates the set of intermediate key value pairs with the same key values generated by Mappers a list of output results. Reducer then iterates over the resulting group causing further reduction from each group. Hadoop store the input and output of the task in

Hadoop Distributed File System (HDFS). Master node is responsible to perform the tasks like monitoring of nodes and scheduling of tasks on the available nodes. The same set of nodes can be designated for the MapReduce framework and HDFS which implies coexistence of computational storage nodes. This configuration effectively deploys a framework for scheduling task on data nodes and hence improves network bandwidth utilization of the entire cluster.
MapReduce Framework has been implemented in several unique ways. Among them, AppEngine MapReduce and Hadoop MapReduce 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.
Companies like IBM and Yahoo have already adopted it. 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.

A.2 Pareto Optimality

Multi-objective optimization helps in achieving optimal solution that fulfills all the constraints raised by the component objectives. It caters the need of software companies who are adopting agile methodology for software development where cost efficient bug free software is of great importance. Computational cost of test cases can often be measured by the time taken for its execution. The solution that detects maximum number of faults with coverage of all the feasible methods and branches and has minimum number of test cases is considered as optimal solution or best solution. Multi-objective optimization problems can be solved by the following methods:

i. **Sequential Approach**: In this methodology multi-objective problem is broken into its singular objectives and solution is found out in series fashion where solution of first objective is given as input to another objective solution and so on to find out the cumulative result of the problem.

ii. **Weighted Approach**: In this classical approach weighted sum of 'n' different objectives are combined to form the single objective as follows:

$$f' = \sum_{i=1}^{n} (w_i . f_i) \tag{A.1}$$

$$\sum_{i=1}^{n} (w_i) = 1 \tag{A.2}$$

However it is easy to compute the weighted sum of the constituent objectives but it is often cumbersome to assign the correct weights to the different objectives.

iii. **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:

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

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. [136]. 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.

A.3 Crowding Distance

Crowding Distance is used to get an estimate of the density of solutions surrounding a particular point in the population. It is calculated by taking the average distance of the two points on either side of this point along each of the objectives. This quantity $i_{distance}$ serves as an estimate of the size of the largest cuboid enclosing the point i without including any other point in the population and it is termed as crowding distance. In Fig. A.1, the crowding distance of the *ith*

solution in its front (marked with solid circles) is the average side-length of the cuboid (shown with a dashed box). The algorithm used to calculate the crowding distance is shown in the Algorithm 7.

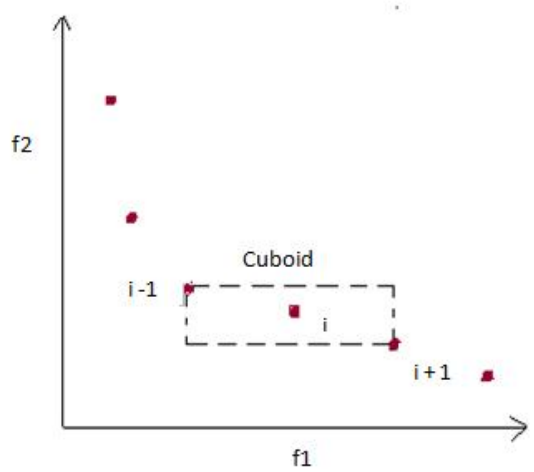

Figure A.1: Crowding Distance Computation

The next section throws light on the Amazon Cloud Services.

A.4 Amazon Cloud Services

This thesis deals with software testing in an elastic cloud environment. The research involves launching a distributed application within a cloud environment as a case study for software testing. The cloud platform selected for the research was Amazon Elastic Compute Cloud (EC2) which is part of Amazon Web Services (AWS).

A.4.1 Amazon Web Services(AWS)

Amazon is arguably a pioneer in offering cloud services and AWS has become the de facto standard for cloud infrastructure services. Other IaaS services either complement AWS or are considered competitors to them. The story behind Amazon's rise in the IT sector was that they wanted to find ways to make better use of under-utilized peak computing power of their on-line retail. High proportion of their sales are processed in the weeks before Christmas. Amazon launched AWS in 2002 to turn their IT cost weakness into an opportunity. The initial plan was selling idle capacity to organizations who needed secure and reliable computing infrastructure, at other times than around the Christmas sales. Today Amazon offers a large number of cloud services, with largest attention on Amazon EC2, described in section 4.3.2 and accommodating storage services described in section 4.3.3.

A.4.2 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (Amazon EC2) is the name of the service that provides the re-sizable compute capacity, or what is referred to as a cloud. Included in Amazon EC2 is a set of web services that allows users to launch and manage Linux/UNIX and Windows server instances in Amazon's data centers, and load them with custom application environment. It is possible to commission from one up to thousands of server instances simultaneously. Public information on on how many instances EC2 customer can reserve at a time is not available, but the default starting limit is twenty reserved instances that a customer can purchase each month. The limit can be raised by contacting Amazon9. Amazon provides web user interface for managing cloud resources. Figure 2.2 is a snapshot of the AWS Management console EC2 view. The console provides management for AWS's compute, storage, and other cloud features. For EC2 management the console has options to start and stop EC2 instances and configure networking and security features. Amazon furthermore provides a software development kit (SDK) for cloud service management called Amazon AWS SDK. Currently the SDK's API has been implemented for Java, Microsoft .NET, PHP, Python, and Ruby. Eclipse users can download the AWS Toolkit plug-in which makes Java application deployment to instances possible from within the Eclipse IDE, plus additional instance management features [5]. The AWS SDK for Java is used for this thesis' case study, to manage cloud instances for the test bed.

A.4.3 Amazon Machine Image

A core feature of cloud computing is virtualization. Virtual machines in Amazon EC2 are called Amazon Machine Images (AMI). An AMI consists of a read-only File-system image which includes an operating system and typically some application software (e.g. web application or a database). Amazon provides prepackaged AMIs with Windows and Linux operating systems installed. Various 3rd party vendor also provide prepackaged images, like Ubuntu which provides official Ubuntu Linux images on Amazon EC2. Users can also create their own images from scratch or extend publicly available images and re-package them with new instance IDs. An AMI can be public, paid, or shared. A public AMI can be used by anyone: A paid image requires subscription fees, and a shared AMI is private and can only be used by Amazon EC2 users specified by the owner of the image. Amazon EC2 Hardware Specifications and Pricing The hardware specification for Amazon instances is selected at startup of each instance and can be selected from predefined a set of instance types defined by Amazon (e.g. Small, Large, or Extra Large). The instance type controls what resources the AMI uses; RAM, CPU and disk size. Table 2.1 lists the variations for EC2 instances and the different prices in USD per hour based on computing power. The prices shown in the table are for Linux/UNIX usage within the European region. Windows operating system costs are roughly around 10-25% higher for each instance type. The case study in this thesis uses the standard 'Small' instance type. The cost for

this thesis' research has summed up to be 112.59 USD for 1,186 instance hours. Amazon provides further pricing options for reserved instances on annual basis or for spot-priced instances. Pricing details can behind on Amazon's EC2 website.

A.4.4 Amazon Data Storage

Because of EC2's elastic nature permanent data persistence is not possible for AMIs. Amazon provides various different cloud services to address data persistence issues in the cloud:

- Amazon Simple Storage Service (S3) provides a web services interface for storing and retrieve data.
- Amazon Elastic Block Storage (EBS) provides block level storage volumes for use with Amazon EC2 instances.
- Amazon SimpleDB is a non-relational data store for data queries via web services.

The difference between S3 and SimpleDB is that SimpleDB is meant to be for smaller amounts of data. SimpleDB was created for performance optimization and in order to minimize costs across AWS services for large objects and files which are stored in S3. Meta-data associated with those files is stored in the SimpleDB. It is possible to run applications in Amazon EC2 and store data objects in Amazon S3. Amazon SimpleDB can then be used to query the object meta-data from within the application in Amazon EC2 and return pointers to the objects stored in Amazon S3. This allows for quicker search and access to objects, while minimizing overall storage costs.

The next section will explain and state the mathematical model of the proposed framework.

A.5 Amazon EC2 Hardware Specifications and Pricing

The hardware specification for Amazon instances is selected at startup of each instance and can be selected from predefined a set of instance types defined by Amazon (e.g. Small, Large, or Extra Large). The instance type controls what resources the AMI uses; RAM, CPU and disk size. Table A.1 lists the variations for EC2 instances and the different prices in USD per hour based on computing power. The prices shown in the table are for Linux/UNIX usage within the US East(North Virginia). Windows operating system costs are roughly around 10-25% higher for each instance type. The case study in this thesis uses the standard 'Small' instance type. The cost for this thesis' research has summed up to be 112.59 USD for 1,186 instance hours. Amazon provides further pricing options for reserved instances on annual basis or for spot-priced instances. Pricing details is shown in the Table A.2 .

Table A.1: Price-list of AWS Instances

Instance Type	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.micro	1	Variable	1	EBS Only	$0.013 per Hour
t2.small	1	Variable	2	EBS Only	$0.026 per Hour
t2.medium	2	Variable	4	EBS Only	$0.052 per Hour
m3.medium	1	3	3.75	1 x 4 SSD	$0.070 per Hour
m3.large	2	6.5	7.5	1 x 32 SSD	$0.140 per Hour
m3.xlarge	4	13	15	2 x 40 SSD	$0.280 per Hour
m3.2xlarge	8	26	30	2 x 80 SSD	$0.560 per Hour
Compute Optimized - Current Generation					
c4.large	2	8	3.75	EBS Only	$0.116 per Hour
c4.xlarge	4	16	7.5	EBS Only	$0.232 per Hour
c4.2xlarge	8	31	15	EBS Only	$0.464 per Hour
c4.4xlarge	16	62	30	EBS Only	$0.928 per Hour
c4.8xlarge	36	132	60	EBS Only	$1.856 per Hour
c3.large	2	7	3.75	2 x 16 SSD	$0.105 per Hour
c3.xlarge	4	14	7.5	2 x 40 SSD	$0.210 per Hour
c3.2xlarge	8	28	15	2 x 80 SSD	$0.420 per Hour
c3.4xlarge	16	55	30	2 x 160 SSD	$0.840 per Hour
c3.8xlarge	32	108	60	2 x 320 SSD	$1.680 per Hour
GPU Instances - Current Generation					
g2.2xlarge	8	26	15	60 SSD	$0.650 per Hour
g2.8xlarge	32	104	60	2 x 120 SSD	$2.600 per Hour
Memory Optimized - Current Generation					
r3.large	2	6.5	15	1 x 32 SSD	$0.175 per Hour
r3.xlarge	4	13	30.5	1 x 80 SSD	$0.350 per Hour
r3.2xlarge	8	26	61	1 x 160 SSD	$0.700 per Hour
r3.4xlarge	16	52	122	1 x 320 SSD	$1.400 per Hour
r3.8xlarge	32	104	244	2 x 320 SSD	$2.800 per Hour
Storage Optimized - Current Generation					
i2.xlarge	4	14	30.5	1 x 800 SSD	$0.853 per Hour
i2.2xlarge	8	27	61	2 x 800 SSD	$1.705 per Hour
i2.4xlarge	16	53	122	4 x 800 SSD	$3.410 per Hour
i2.8xlarge	32	104	244	8 x 800 SSD	$6.820 per Hour
d2.xlarge	4	14	30.5	3 x 2000 HDD	$0.690 per Hour
d2.2xlarge	8	28	61	6 x 2000 HDD	$1.380 per Hour
d2.4xlarge	16	56	122	12 x 2000 HDD	$2.760 per Hour
d2.8xlarge	36	116	244	24 x 2000 HDD	$5.520 per Hour

Table A.2: Price-Table of AWS Spot Instances

General Purpose - Current Generation		
	Linux/UNIX Usage	**Windows Usage**
m3.medium	$0.0081 per Hour	$0.0591 per Hour
m3.large	$0.0162 per Hour	$0.1171 per Hour
m3.xlarge	$0.0323 per Hour	$0.1381 per Hour
m3.2xlarge	$0.0646 per Hour	$0.2751 per Hour
General Purpose - Previous Generation		
m1.small	$0.0071 per Hour	$0.0171 per Hour
m1.medium	$0.0084 per Hour	$0.0331 per Hour
m1.large	$0.0161 per Hour	$0.0662 per Hour
m1.xlarge	$0.0335 per Hour	$0.1326 per Hour
Compute Optimized - Current Generation		
c3.large	$0.016 per Hour	$0.104 per Hour
c3.xlarge	$0.032 per Hour	$0.198 per Hour
c3.2xlarge	$0.064 per Hour	$0.396 per Hour
c3.4xlarge	$0.128 per Hour	$0.792 per Hour
c3.8xlarge	$0.256 per Hour	$1.584 per Hour
c4.large	$0.0203 per Hour	$0.0991 per Hour
c4.xlarge	$0.0398 per Hour	$0.1989 per Hour
c4.2xlarge	$0.1177 per Hour	$0.3988 per Hour
c4.4xlarge	$0.1466 per Hour	$0.807 per Hour
c4.8xlarge	$0.3858 per Hour	$1.5884 per Hour
d2.xlarge	$0.0737 per Hour	$0.2366 per Hour
d2.2xlarge	$0.1454 per Hour	$0.32 per Hour
d2.4xlarge	$0.2863 per Hour	$0.944 per Hour
d2.8xlarge	$0.6117 per Hour	$1.888 per Hour
Compute Optimized - Previous Generation		
c1.medium	$0.0161 per Hour	$0.0501 per Hour
c1.xlarge	$0.0641 per Hour	$0.2001 per Hour
cc2.8xlarge	$0.2594 per Hour	$0.4287 per Hour
GPU Instances - Current Generation		
g2.2xlarge	$0.0679 per Hour	$0.106 per Hour
g2.8xlarge	$0.3157 per Hour	$0.4395 per Hour
GPU Instances - Previous Generation		
cg1.4xlarge	$2.1001 per Hour	N/A*
Memory Optimized - Current Generation		
r3.large	$0.0225 per Hour	$0.1741 per Hour
r3.xlarge	$0.0554 per Hour	$0.2831 per Hour
r3.2xlarge	$0.077 per Hour	$0.4446 per Hour
r3.4xlarge	$0.1758 per Hour	$0.673 per Hour
r3.8xlarge	$0.3754 per Hour	$0.9583 per Hour
Memory Optimized - Previous Generation		
m2.xlarge	$0.0184 per Hour	$0.0714 per Hour
m2.2xlarge	$0.0328 per Hour	$0.1401 per Hour
m2.4xlarge	$0.0718 per Hour	$0.2802 per Hour
cr1.8xlarge	$0.2819 per Hour	N/A*
Storage Optimized - Previous Generation		
hi1.4xlarge	$0.1711 per Hour	$0.4819 per Hour
Micro Instances		
t1.micro	$0.0031 per Hour	$0.0061 per Hour