Design of Scenario-based Application-optimized Data Replication Strategies through Genetic Programming

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der

Carl von Ossietzky Universität Oldenburg

zur Erlangung des Grades und Titels eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation von

Herrn Syed Mohtashim Abbas BOKHARI

geboren am 22.08.1990 in Islamabad

Gutachter: Prof. Dr. Oliver Theel

Weiterer Gutachter: Prof. Dr. Oliver Kramer

Tag der Disputation: 18.08.2021

Abschließende Erklärung

Hiermit erkläre ich, dass mir die Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg bekannt sind und von mir befolgt wurden. Im Zusammenhang mit dem Promotionsvorhaben wurden keine kommerziellen Vermittlungs- oder Beratungsdienste (Promotionsberatung) in Anspruch genommen.

Syed Mohtashim Abbas Bokhari

Unterschrift:

Ort, Datum: Odenburg, 18.08.2021

"Transience substantiates the Necessary Existence."

Syed Mohtashim Abbas Bokhari

Kurzzusammenfassung

Ein verteiltes System ist ein Paradigma, welches für die moderne, technologische Welt unverzichtbar ist, in der jede Sekunde unzählige Anfragen verarbeitet werden. Dafür braucht es in verteilten Systemen eine hohe Verfügbarkeit. In einer sich verändernden Umgebung sind durch die Komplexität und Skalierbarkeit der Ressourcen und Komponenten die Systeme häufig Fehlern ausgesetzt, da Millionen von Geräten miteinander verbunden sind und kommunizieren. Verteilte Systeme erlauben einer Vielzahl von Nutzern auf gemeinsame Ressourcen zuzugreifen, wodurch Fehler unausweichlich werden. Außerdem neigt bereits ein einzelnes Replikat zum Ausfall, was katastrophal für die Verfügbarkeit einer Zugriffsoperation ist. Da Fehler im verteilten Kontext unvermeidbar sind, nimmt dieser großen Einfluss auf die Verfügbarkeit von Diensten. Fehler verhindern die Verfügbarkeit von Daten, wodurch sie den Ausfall des Systems verursachen. Replikation spielt eine Rolle bei der Milderung solcher Fehler. Sie maskiert diese, um eine fehlertolerante Umgebung zu schaffen, wodurch jede mögliche Verminderung der Verfügbarkeit der Daten ausgeschlossen wird. Sie maskiert die Fehler zur Laufzeit gegenüber dem Benutzer und das System funktioniert weiterhin entsprechend den Erwartungen.

Hohe Verfügbarkeit und niedrige Kosten der Zugriffsoperation bei Einhaltung der Datenkonsistenz sind die hervorstehenden Herausforderungen bei der Konzeptionierung zuverlässiger Dienste. Replikation ist ein passendes Werkzeug, welches hochverfügbare Zugriffsoperationen zu relativ niedrigen Operationskosten ermöglicht. Obwohl gegenwärtig verschiedene Datenreplikationsstrategien verwendet werden, um das Replikationsverhalten der Daten zu steuern, ist die Frage, welche Strategie für ein gegebenes Szenario oder eine Applikationsklasse die beste ist, unbeantwortet. Dieses beinhaltet einen bestimmten Datendurchsatz, eine Verteilung über ein Netzwerk, die Verfügbarkeit der einzelnen Replikate und die Kosten einer Zugriffsoperation. Die Entscheidung für eine passende Strategie in einer bestimmten Umgebung und in einem spezifizierten Szenario ist eine Herausforderung, bei der Kompromisse eingegangen werden müssen.

Es existieren viele Szenarios, welche die verschiedenen Kompromisse zwischen den Qualitätsmetriken widerspiegeln. Eine relevante Strategie für ein spezifiziertes Szenario zu finden ist mühselig, da unendliche viele Szenarien existieren können, während die Anzahl der Lösungen begrenzt ist. Diese Szenarien können nicht vollständig von den gegenwärtigen Strategien abgedeckt werden. Es ist erforderlich, neue Datenreplikationsstrategien zu entwerfen, welche auf die gegebenen Szenarien spezialisiert sind und dabei jene Anforderungen erfüllen, die mit bekannten Strategien unerfüllt bleiben. Eben diese Anforderungen sind in einer Weise konfliktär zueinander, sodass die Erfüllung eines Ziels die Verwerfung eines anderen bedeutet. Dies impliziert, das keine optimale Lösung existiert. In dieser Hinsicht fokussiert sich diese Forschung auf zielgerichtete Modellierung, Analyse und Ansätze maschinellen Lernens, um automatisch solche Datenreplikationsstrategien zu identifizieren und entwerfen, welche für das

gegebene Szenario optimal sind, auf vordefinierten Anforderungen und Eigenschaften basieren und eine sogenannte „Voting Structure“ nutzen.

Diese Dissertation stellt einen Ansatz zur Abwägung und Optimierung der Ziele vor, welcher auf genetischer Programmierung basiert und nicht nur geeignete Replikationsstrategien identifiziert, sondern neue Strategien entwirft und konfliktäre Ziele gegen eine Metrik optimiert. Dieser Mechanismus entwirft automatisch bisher unbekannte Datenreplikationsstrategien, welche gegen ein gegebenes Szenario optimiert sind. Er untersucht unbekannte Strategien und entwickelt eine Population von Strategien, welche jeweils ein Programm repräsentieren, schrittweise weiter. Über die Generationen werden die Strategien mit einer evolutionären Methode gegen die Entwurfskriterien optimiert, während die Datenkonsistenz und die Korrektheit der Lösungen gewahrt wird. Weiterhin werden die Operatoren „multi-crossover“ und „multimutation“ vorgestellt, welche unser Programmiergerüst des maschinellen Lernens erweitern, während die Konsistenz der Lösungen garantiert wird, um innovative hybride Replikationsstrategien zu erzeugen. Dieser Mechanismus nutzt sogenannte „Voting Structures“ gerichteter, azyklischer Graphen (bekannt aus Literatur) als eine einheitliche Repräsentation von Replikationsstrategien. Diese nutzt die Heterogenität der Strategien, was die Notation hybrider Strategien vereinfacht. Die Strukturen werden durch den überliegenden Algorithmus zur Laufzeit interpretiert, um entsprechende Quoren abzuleiten. Die Quoren werden verwendet, um die replizierten Objekte zu verwalten.

Diese Forschung bietet einen intelligenten und automatischen Mechanismus, um neue Datenreplikationsstrategien zu erzeugen und den Entscheidungsprozess zwischen den relevanten Strategien zu erleichtern. Mehrere Strategien, die unterschiedliche Kompromisse hinsichtlich der definierten Anforderungen eingehen, werden aus den erzeugten Strategien präsentiert und zur Auswahl angeboten. Die Forschung demonstriert den Nutzen des auf genetischer Programmierung basierenden Ansatzes durch die signifikante Reduktion der Schreib- und Lesekosten, bei nur wenig geringerer Zugiffsverfügbarkeit. Der Mechanismus erzeugt innovative Replikationsstrategien, die in ihrer Kombination bisher unbekannt waren. Der gezeigte Ansatz ist sehr effektiv und extrem flexibel darin Strategien anzubieten, die mit gegenwärtigen Strategien vergleichbar sind. Er erzeugt neue Strategien unter Verwendung der relevanten genetischen Operatoren.

Abstract

A distributed system is a paradigm, which is indispensable to the current world due to countless requests with every passing second. Therefore, in distributed computing, high availability is very important. In a dynamic environment due to the scalability and complexity of the resources and components, systems are fault-prone because millions of computing devices are connected via communication links. Distributed systems allow many users to access shared computing resources, which makes faults inevitable. Also, a single replica is prone to failure, which is, too, devastating for the availability of the access operations. Since failures are often inevitable in a distributed paradigm, it greatly affects the availability of services. Faults hinder the availability of the data, thereby causing systems to fail. Replication plays a role in mitigating such failures by masking them to achieve a fault-tolerant distributed environment, thereby eliminating any such possible hindrances in the availability of the data. It masks the faults at run-time while users are unaware of it and the system continues to work as expected.

High availability and low cost of the access operations as well as maintaining data consistency are major challenges for reliable services. For this, replication is an appropriate means to provide highly available data access operations at relatively low operation costs. It is the concept by which highly available data access operations can be realized while the cost should be not too high either. Although several contemporary data replication strategies are being used to control the replicated behavior once the data is replicated, the question still stands which strategy is the best for a given scenario or application class, assuming a certain workload, its distribution across a network, availability of the individual replicas, and cost of the access operations. The decision to choose an appropriate strategy for a certain environment and a specific scenario is a challenge and full of compromises.

There exist numerous scenarios reflecting different trade-offs between several quality metrics, and identifying a relevant strategy for a specific scenario is quite cumbersome. Since there could exist potentially infinite scenarios and solutions are limited. These scenarios cannot be covered entirely by contemporary strategies. It requires designing new data replication strategies optimized for the given scenarios, satisfying the constraints of such scenarios, which may be left unaddressed otherwise. The constraints of such scenarios are often conflicting in the sense that an increase in one objective could be sacrificial to the others, which implies there is no best solution to the problem, but what serves the specific purpose. In this regard, this research focuses on sophisticated modeling, analysis, and machine learning approaches to automatically identify and design such replication strategies that are optimized for a given application scenario based on predefined constraints and properties exploiting a so-called voting structure.

This dissertation proposes a genetic programming-based multi-objective optimization approach that endeavors to not only identify but also design new data replication

strategies and optimize their conflicting objectives as a single-valued metric. This mechanism automatically designs new replication strategies (up-to-now unknown) optimized for given scenarios. It explores unknown replication strategies and evolves the population of replication strategies (representing each a computer program) gradually, but consistently over several generations of evolution in order to make them optimized to eventually meet the desired criteria while maintaining the consistency (correctness) of the solutions, too. Furthermore, it introduces strong multi-crossover and multi-mutation operators to replication, which strengthen our machine learning framework, at the same time guaranteeing consistency of the solutions, to generate innovative hybrid replication strategies. This mechanism uses a so-called voting structure of directed acyclic graphs known from literature as a unified representation of replication strategies. This unified representation exploits the heterogeneity between the strategies, thereby making the notion of hybrid strategies easier to accomplish, which otherwise would have been very cumbersome to achieve, therefore, to optimize. These voting structures are interpreted by the general algorithm at run-time to derive respective quorums. These quorums are eventually used to manage replicated objects.

The research provides an intelligent, automatic mechanism to generate new replication strategies as well as easing up the decision-making so that relevant strategies with satisfactory trade-offs of constraints can easily be picked and used from the generated solutions at run-time. The research demonstrates the usefulness of this genetic programming-based automatic mechanism by reducing the cost significantly while not comprising too much on the availabilities of the access operations. It generates replication strategies that are innovative and such combinations have not been explored yet. The proposed approach is very effective and extremely flexible to offer competitive results w.r.t. the contemporary strategies as well as generating novel strategies even with a slight use of relevant genetic operators.

Contents

Abschließende Erklärung i

Kurzzusammenfassung iii

Abstract v

1 Introduction 1

1.1 Background 1
1.2 Motivation 3
1.3 Problem statement 4
1.4 Dissertation outline 6
1.5 Summary 7

2 Fault tolerance and replication 8

2.1 Basic concept 8
2.2 Fault, error, and failure 8
2.3 Fault models in the literature 9
2.3.1 Functional failure model 9
2.3.2 Structural failure model 11
2.4 Replication strategies and related work 13
2.4.1 Voting Structures 15
2.4.2 Read-One Write-All Protocol 17
2.4.3 Majority Consensus Strategy 18
2.4.4 Weighted Voting 18
2.4.5 Tree Quorum Protocol 19
2.4.6 Grid Protocol 19

2.4.7 Triangular Lattice Protocol 20
2.5 Summary 21

3 Machine learning and genetic programming 23
3.1 Basic concepts of machine learning 23
3.2 Genetic programming 24
3.2.1 Why genetic programming? 25
3.2.2 GP related work 25
3.2.3 General object-oriented GP 34
3.3 Genetic programming in the context of replication 36
3.4 Summary 39

4 Novel framework to design replication strategies 40
4.1 Adopted fault model 40
4.2 System architecture 40
4.3 Specification of a constraints-based scenario 41
4.3.1 Consistency of operations 42
4.3.2 Number of replicas 42
4.3.3 Availabilities of the access operations 42
4.3.4 Costs of the access operations 43
4.3.5 Fitness weightage 43
4.3.6 Probability of the individual replicas 44
4.4 Manual designs of voting structures by modeling the state-of-the-art strategies 44
4.5 Customized genetic programming algorithm 49
4.6 Fitness function for the strategies' evaluations 50
4.7 Crossover operators for strategies 52
4.7.1 Type 1 operator 53
4.7.2 Type 2 operator 53
4.7.3 Type 3 operator 54
4.7.4 Type 4 operator 54

4.8 Mutation operators for the strategies 59
4.8.1 Type 1 operator 60
4.8.2 Type 2 operator 60
4.9 System parameters 62
4.9.1 μ and λ 63
4.9.2 Initial population probability 63
4.9.3 Intra-crossover probability distribution 63
4.9.4 Mutation probability 63
4.9.5 Intra-mutation probability distribution 63
4.10 Summary 64

5 Experiments and results **65**
5.1 Scenarios 65
5.1.1 Scenario 1 65
5.1.2 Scenario 2 66
5.1.3 Scenario 3 66
5.1.4 Scenario 4 66
5.2 Results and discussions 67
5.2.1 System parameter settings for scenario 1 70
5.2.2 Results for scenario 1 71
5.2.3 System parameter settings for scenario 2 76
5.2.4 Results for scenario 2 77
5.2.5 System parameter settings for scenario 3 82
5.2.6 Results for scenario 3 83
5.2.7 System parameter settings for scenario 4 88
5.2.8 Results for scenario 4 88
5.3 Summary 98

6 Conclusions and future work **100**
6.1 Summarization and contributions 100

6.2 Future work 101
6.3 Dissertation publications 102

Bibliography **103**

Acronyms **111**

Symbols **113**

List of Figures

1.1 Single replica dependency 2
1.2 Single replica failure 2
1.3 Three functionally identical replicas 2
1.4 Decision by majority consensus 3
1.5 Intersection of read and write quorums 3
1.6 Trade-off scenarios 5

2.1 Dependency among fault, error, and failure 9
2.2 Hierarchical ordering of functional failure classes 9
2.3 Ring topology 13
2.4 Contemporary replication strategies 14
2.5 Example of a voting structure 16
2.6 DRSs being represented as voting structures 17
2.7 Generalized tree quorum protocol 19
2.8 Optimized Grid protocol of 3×3 20
2.9 Triangular lattice protocol 3×3 21

3.1 Tree-based GP instance 26
3.2 Sushil Louis's evolved 2-bit adder 31
3.3 An example of subgraph active–active node (SAAN) crossover 32
3.4 Conceptual Basic OOGP representation 34
3.5 Genotype representation of strategies 35
3.6 Phenotype modeling of a TLP-like strategy as a voting structure 36
3.7 Phenotype - DRSs as JSON documents 36
3.8 Genetic programming 38

4.1 Methodology 41
4.2 Hybrid DRSs of MCS & TLP 44
4.3 Voting Structure: MCS on top of TLP 45
4.4 Voting Structure: TLP on the top of MCS 45
4.5 Availability of hybrid DRS 46
4.6 Cost of Hybrid DRS 46
4.7 Hybrid DRSs of GP and TLP 47
4.8 Voting Structure: GP on the top of TLP 47
4.9 Voting Structure: TLP on the top of GP 48
4.10 Availability: MCS vs. hybrid DRS 48
4.11 Cost: MCS vs. hybrid DRS 49
4.12 General crossover 52
4.13 Strategy 1 54
4.14 Strategy 2 54
4.15 Crossover type 1 operator 55
4.16 Part of strategy 1 55
4.17 Part of the strategy 2 56
4.18 Crossover type 2 operator 56
4.19 Crossover type 3 operator 57
4.20 Strategy 1 58
4.21 Strategy 2 58
4.22 Crossover Type 4 operator (a) - offspring DRS 1 59
4.23 Crossover Type 4 operator (a) - offspring DRS 2 59
4.24 Crossover Type 4 operator (b) - offspring DRS 60
4.25 Strategy 1 60
4.26 Crossover Type 4 operator (d) - offspring DRS 61
4.27 Strategy 2 61
4.28 Crossover Type 4 operator (e) - offspring DRS 62
4.29 Chosen points for mutation 62

4.30 Mutated DRS 63

5.1 Hybrid Strategy 1 67
5.2 Hybrid DRS 1, availability of the access operations 68
5.3 Hybrid DRS 1, cost of the access operations 68
5.4 Hybrid strategy 2 69
5.5 Hybrid DRS 2, availability of the access operations 70
5.6 Hybrid DRS 2, Zoom-in availability graph 70
5.7 Hybrid DRS 2, cost of the access operations 71
5.8 2D representation of the generated DRSs 72
5.9 2D representation of the generated DRSs 73
5.10 Fitness graph 73
5.11 Populations' evolution 74
5.12 Optimized hybrid DRS for the given scenario 74
5.13 Availability graph of the read and write operations 75
5.14 Availability, MCS vs. hybrid DRS (16 replicas) 76
5.15 Cost, MCS vs. hybrid DRS (16 replicas) 76
5.16 DRS generated via genetic programming 78
5.17 Pareto front 79
5.18 Scenario, fitness availability analysis 79
5.19 Scenario, populations' analysis 80
5.20 Scenario, optimized hybrid DRS 80
5.21 Availability of the chosen optimized DRS 81
5.22 Availability, MCS vs. hybrid DRS (16 replicas) 81
5.23 Zoom-in view of the respective availabilities 82
5.24 Cost, MCS vs. hybrid DRS (16 replicas) 83
5.25 Pareto front view for scenario 3 84
5.26 An optimized DRS for scenario 3 84
5.27 Availability of the mentioned DRS 85
5.28 Cost of the mentioned DRS 85

5.29 Another optimized DRS with a slightly different crossover point 86
5.30 Availability comparison of the two Pareto front solutions 87
5.31 Cost comparison of the two Pareto front solutions 87
5.32 Fitness availability analysis of the generated DRSs 89
5.33 Populations' analysis 89
5.34 Costs analysis of the generated DRSs 90
5.35 Discovered optimized hybrid DRS 91
5.36 Availability, MCS vs. hybrid DRS (16 replicas) 91
5.37 Zoom-in view of the respective availabilities 92
5.38 Cost, MCS vs. hybrid DRS (16 replicas) 92
5.39 Another optimized hybrid DRS 93
5.40 Availability comparison between MCS and other hybrid DRSs 93
5.41 Availability comparison between MCS and other hybrid DRSs 94
5.42 Availability comparison between MCS and other hybrid DRSs 94
5.43 Cost comparison between MCS and other hybrid DRSs 95
5.44 A mutated hybrid DRS 95
5.45 Availabilities of MCS, hybrid, and mutated DRSs 96
5.46 Availabilities of MCS, hybrid, and mutated DRSs 97
5.47 Costs of MCS, hybrid, and mutated DRSs 97
5.48 Auto-generated availability graphs via genetic programming 98

List of Tables

3.1 LGP phenotype instance 27

3.2 Example BNF form 28

3.3 An example genotype in GE. Here, the eight-bit binary codons have all been packed as integers for convenience 28

3.4 Decoding the example genotype given in Table 3.3 29

5.1 Results of the GOOGP on repeated runs 77

Chapter 1

Introduction

1.1 Background

In the past decades, computer systems such as banking cash machine networks, networked business systems, or internet services have become prevalent and gained significant importance in our private as well as business lives daily. The flip side of the coin is that computer systems have not only caused economic damage but also failures leading to the loss of lives [1]. Thus, a key concern is the trustworthiness of such systems w.r.t. safety, security, privacy, performance, correctness, and availability. In particular, availability as the probability that a system is operable at a given time and provides the intended service, is of utmost importance.

Providing highly available data access operations is a prevalent problem in computer science. Relying on a single replica significantly confines the availability of the data. Therefore, the increase in the number of replicas to store the data objects is inevitable, which, when smartly applied, increases the availability of the data object and makes it more fault-tolerant. Because now, the data can be accessed by approaching other replicas too; thus, the system continues to operate despite the failure of certain nodes hosting replicas. Data replication is, hence, a means by which some failures in a distributed paradigm can be masked, thereby attaining better availability and fault tolerance in the system. However, in the case of replicated data, one could easily succumb to incorrect values when one replica is updated and other replicas do not reflect the change. Hence, the challenge comes up in managing those replicas and avoiding inconsistencies so that replicas always yield correct values. Inconsistency means discrepancy in the data among created replicas. Moreover, conflicting operations, too, need to be managed to prevent them from affecting correctness. These problems are known as consistency issues, and falling into these issues is not the intended behavior of the system.

All this suggests that in anticipation of accomplishing high availability by merely replicating the copies of the same data over several nodes is not a straightforward task. The goal of the data access operations is also to behave in a replicated system the same as they would do in a non-replicated system. This is known as one-copy serializability (1SR) [2], which is achieved by the strategies that enforce 1SR to maintain the correctness of the data. It is extremely important to maintain the correctness of the data, particularly, for mission-critical measurements. The data must also be exclusively locked for the write operation so that no concurrent access operations (except read-read) can be performed on the replicas, simultaneously, to again adhere to the correctness

notion. The data should always be consistent to meet the 1SR property. For this, there are strategies known as data replication strategies (DRSs), i.e., [3], to ensure such property and control the replicated behavior to make distributed systems highly available, thus, more reliable.

Let us consider a simple example of a replica, i.e., R1, a single replica, given in Figure 1.1 that maintains crucial data.

FIGURE 1.1: single replica dependency

In the case of such crucial data or critical measurements, any failure in the replica, i.e., in Figure 1.2, can easily lead to a catastrophic outcome that may cost lives.

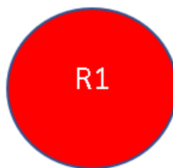

FIGURE 1.2: Single replica failure

For this, instead of relying on one replica, a system can rather be comprised of three functionally identical components, i.e, R1, R2, and R3 as shown in Figure 1.3.

FIGURE 1.3: Three functionally identical replicas

This way, the failure of one component, i.e., R2 as shown in Figure 1.4, can easily be compensated by the other two components, i.e, R1 and R3, if the decision is supposed to be made by the majority for it to be conclusive. The presence of this redundancy among the components along with this majority protocol can easily compensate for the failure of a single component as shown in Figure 1.4. In Figure 1.4, R2 is down, but data can be accessed as R1 and R3 are still up and running; hence, we achieve fault tolerance in the system and, thereby, higher availability.

Many DRSs enforce a quorum mechanism (a threshold of a minimal number of replicas) comprised of a read quorum (rq) and a write quorum (wq) to perform the preferred access operations. A quorum is a subset of the set of all the replicas mandatory to execute an operation. The access operations are either a read or a write operation. A

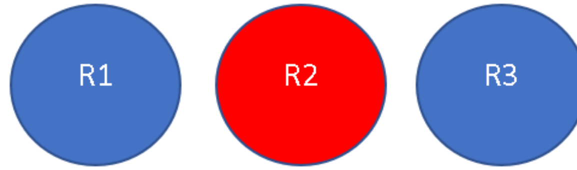

FIGURE 1.4: Decision by majority consensus

read operation reads values from the set of replicas of a read quorum (RQs) and a write operation (WQs) writes values to the set of replicas of a write quorum as shown in Figure 1.5. A read operation reads the data of all replicas of a RQ, i.e., R1 and R2, and spots the up-to-date replica through its latest version number. While a write operation uses an atomic commit protocol, like the Two-Phase commit Protocol [2] to write the updated value to all replicas of a WQ, i.e., R2 and R3. Also, through such quorums, the data is exclusively locked for the write operations since concurrent read-write and write-write operations may also lead to incorrect values.

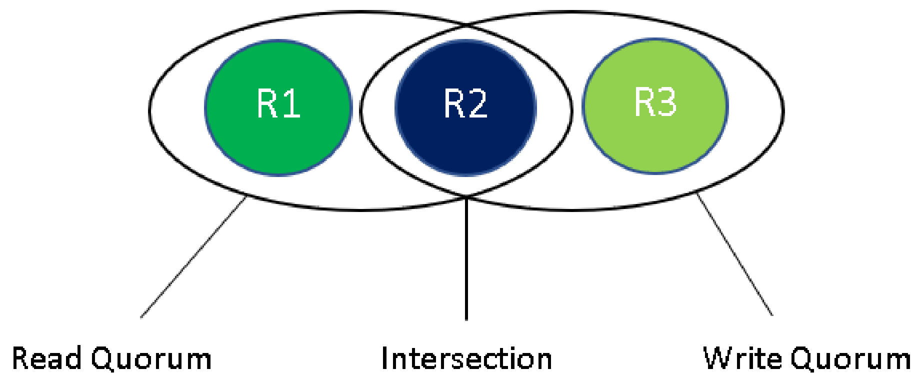

FIGURE 1.5: Intersection of read and write quorums

Hence, these strategies work in such a way that the intersection between the read and write quorums and the locking mechanism happen to meet the 1SR property as shown in Figure 1.5. The DRSs may enforce certain topologies and patterns to access replicas, which thereby indicate their diversity. Some strategies, known as structured replication strategies, i.e., [4], adopt certain patterns and logical structures to access replicas while others, known as unstructured replication strategies, i.e., [5], adhere to no such logical structure, hence, offer more freedom in accessing the replicas though at a higher cost of the access operations. The decision for a suitable DRS to be chosen for an environment surely is a trade-off between different quality metrics, i.e., load, capacity, availability [6], scalability, and cost [7].

1.2 Motivation

These metrics are often conflicting with each other in a way that one cannot be optimized without deteriorating the others. The availability of a read operation is point symmetrical to the availability of write operation [8] in optimized DRSs which means,

both cannot be optimized independently. An increase in the read availability often, therefore, results in sacrificing the write availability and vice versa, so is the case with the cost of the access operations. This could easily fall into the realm of a multi-objective optimization problem [9]. This includes mathematical optimization problems involving more than one objective function to be optimized simultaneously. Multi-objective optimization has its applicability in many domains of science where optimal decisions have to be taken between the trade-offs of two or more conflicting objectives. Since the best solution for one scenario could be the worst for another one, therefore, the goal is to find optimal solutions and quantify the trade-offs in satisfying the specified scenario.

So, the conflicting goals imply no best solution and compromises must be made depending on the scenario of suitable choices w.r.t. concerned application. Furthermore, there can be infinitely many such scenarios, but DRSs lack to cover them entirely. The questions arise as to what are those compromises, to what extent particular values can be compromised, and at the expense of what? These compromises could be highly application-specific, thereby resulting in many different scenarios of suitable choices, which will be discussed briefly in the next section and, in detail, in Chapter 4. This requires designing of new optimized DRS [10]. As for this, the research intends to provide application-optimized DRSs to fulfill such specified scenarios. In this regard, this dissertation is an interesting overlap between the concepts of replication in distributed systems and machine learning. It aims to automatically application-optimize DRSs to satisfy such constraints-based scenarios through a strong machine learning framework. Simplistically, this dissertation solves a multi-objective problem of DRSs through genetic programming, which results in new replication strategies optimized for the specified constraints. In this regard, the research questions are explained in detail here.

1.3 Problem statement

The research problem of constraints is illustrated by a triangle of trade-offs given in Figure 1.6 where, in our case, the consistency part is static because 1SR is maintained all the time to keep the data highly consistent. 1SR is a notion that allows a replicated system to behave as a non-replicated system. The problem predicates upon application-specific scenarios, which will be discussed in detail later in Chapter 4. However, to make it rather simplistic at the moment, the quality metrics are availability and cost of the access operations, which also depend upon the number of replicas and their individual replica availability given that the 1SR holds. The availability of the access operations is the probability by which a user can successfully perform an operation from anywhere within the system. The operation cost is the average minimal number of replicas required (to be accessed) to get the correct value, which is the average of the minimum number of replicas mandatory to form a quorum. Choosing such metrics is for abstraction and also to simplify the problem to carry out the analysis, which otherwise would be very complicated to perform.

These metrics are important for making the services of the system available as well as reducing the messaging overheads and delays. Therefore, the aim, in general, is to increase availability and reduce costs. However, as mentioned, the availabilities of read and write operations are optimally point-symmetric to each other [8], which implies both cannot be optimized independently. Also, an increase in the cost of a read

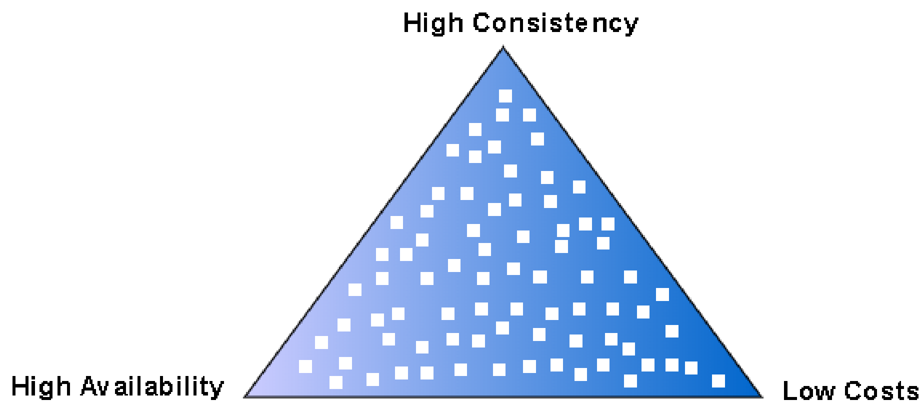

FIGURE 1.6: Trade-off scenarios

operation often compromises the write operation's cost. Moreover, the relationship between the cost and availability of the access operations is not linear either. In a distributed paradigm, there exist numerous cases of trade-offs between these quality metrics represented by white marks shown in Figure 1.6. These white marks represent potentially infinitely many scenarios between the cost and availabilities of the access operations (including the total number of replicas and their individual availabilities too) despite securing the consistency notion to be 1SR. These scenarios cannot be addressed entirely by contemporary strategies. Considering the fact that not every strategy fulfills each scenario, leaves many scenarios unaddressed, for which no optimal strategy exists. Hence, there is no best solution (in terms of a global optimum) but solutions that serve a particular purpose (i.e., local optima).

It demands the designing of new replication strategies that are optimized [10]. One way of it is to use existing strategies to design new ones, but the question arises, how to construct unknown DRSs out of the existing ones, and what are the hindrances in doing so? For this, this dissertation focuses on the automatic identification and design of optimized data replication strategies. It endeavors to exploit heterogeneity among the existing solutions to develop new hybrid replication strategies (i.e., heterogeneous DRSs combined together). In this regard, the challenges are:

1) resolve the multi-objective problem of conflicting nature by possibly determining the fitness as a single quality-metric,

2) develop means to identify the application-optimized DRSs,

3) if there are no such replication strategies optimized of the scenario, come up with new strategies to solve the problem, and

4) finally, but most importantly, develop a machine learning approach to automatically but intelligently design as well as optimizing DRSs for specified scenarios.

This work not only combines the concepts of data replication in distributed systems with genetic programming but also introduces new genetic operators (multi-crossover as well as multi-mutation) to explore more possibilities as well as combinations, by gaining fine-grained control over the mechanism. It aims to develop and strengthen

a machine learning mechanism to automatically identify as well as designing such optimized data replication strategies, satisfying the scenarios. The strategies are diverse and use different topologies and patterns to access the replicas. In this regard, this approach makes use of the heterogeneity of the solutions through a unified representation (based on directed acyclic graphs) known as voting structures [11], [12], [13] to create new innovative solutions.

1.4 Dissertation outline

This dissertation has resulted in six scientific publications but has the potential to produce many more (as mentioned in the future work of this dissertation in Section 6.2), which cannot be fulfilled at the moment due to time restrictions. My dissertation work is initiated by a basic idea "A Flexible Hybrid Approach to Data Replication in Distributed Systems", which is published as a Springer Book Chapter in Advances in Intelligent Systems and Computing (AISC) [10], and presented at the Computing Conference (SAI), London in 2020. This paper manually combines, models cutting-edge replication strategies as unified voting structures, evaluates their performances, and compares the results with contemporary strategies.

Initially, I demonstrated newly designed DRSs using voting structures, particularly focusing on Majority Consensus Strategy, Grid Protocol, and Triangular Lattice Protocol, which later, act as building blocks in the automatic designing of such strategies. I subsequently implemented the idea using genetic programming and presented preliminary results in a publication titled "Design of Scenario-based Application-optimized Data Replication Strategies through Genetic Programming". This was presented in Valletta, Malta in 2020, and published in the proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART), 2020 [14], an extended version of which, titled "Designing New Data Replication Strategies Automatically" is also published as a Springer Book Chapter in Lecture Notes in Artificial Intelligence (LNAI) in 2021 [15].

Also, in this regard, a paper titled "A Genetic Programming-based Multi-objective Optimization Approach to Data Replication Strategies for Distributed Systems" focusing on multi-objective optimization has been presented and published in the proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC, WCCI), Glasgow, Scotland in 2020 [16]. Moreover, an extension of this work by introducing new genetic operators (i.e., multi-crossover and multi-mutation) has been presented and published in the proceedings of the conferences, the 26th IEEE International Conference on Parallel and Distributed Systems (ICPADS) [17] held in Hong Kong in 2020 and the 25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), 2020 [18] held in Australia in 2021.

The dissertation is structured as follows. Chapter 2 explains the basic concept of fault tolerance and replication, the state-of-the-art DRSs, and other contemporary approaches to address the problem, as well as their limitations, followed by the concept of voting structures. Chapter 3 discusses the basic concepts of Machine Learning to Genetic Programming, and subsequently introduces a concept of General Object-orientated genetic programming in the context of fault tolerance and replication. Chapter 4 specifies the fault model, describes the proposed methodology and algorithms to address the research problem. Chapter 5 presents the results and their comparisons, followed by a conclusion and future work, in Chapter 6.

1.5 Summary

Data replication is an important means to provide availability and fault tolerance in distributed systems since data replication to some extent masks the failures of replicas and the system continues to work as expected. However, we need protocols, i.e., data replication strategies to control this replicated behavior of the system, once the data is replicated. Data replication strategies exhibit different behavior and properties. In this regard, this work is about designing new replication strategies to manage replicas, which is comprised of six publications in popular domain conferences, ICPADS, PRDC, CEC, ICAART, SAI, etc. The problem is predicated upon the quality metrics, i.e., availability and cost of the access operations, the total number of replicas, individual replica availability, consistency of access operation, etc., forming different application-specific scenarios. Every replication strategy has different properties w.r.t. these metrics. These quality metrics are conflicting with each other. Simplistically, the aim, in general, is to increase availability and reduce costs. In this regard, this research is about designing (automatically) new replication strategies to fulfill such quality metrics (scenarios) by resolving their multi-objective nature of properties. Furthermore, the basic outline and the structure of this dissertation are also described in this chapter.

Chapter 2

Fault tolerance and replication

2.1 Basic concept

Data replication is a well-known and prevalent concept to enhance the availability of systems prone to failures. It improves availability and helps to construct dependable and trustworthy systems. Fault tolerance is a concept to achieve an increased availability of the access operations through replication, which is attained by 1) the redundancy within components, and 2) a protocol to manage this redundant behavior so that correctness is ensured. Data replication in distributed systems is used to improve the read and the write operation availability on critical data objects as well as improving the read and write operation efficiency, and balancing the workload in the system. A distributed system comprises multiple autonomously operating nodes. Generally, each node maintains a local copy of the data object called a replica. Each replica is associated with a version number and operation-specific locks. The nodes are interconnected by a communication network and interact with one another by sending and receiving messages through the communication network. In spite of the presence of certain faults in distributed systems caused by node or communication link failures, the access operations on the replicated data objects remain available because of the redundancy in replicas. Also, the presence of multiple replicas provides a choice to execute an operation on certain replicas, i.e., geographically nearby located replicas or even the local replica.

2.2 Fault, error, and failure

According to [19], a system failure occurs when the system begins to behave incorrectly in relation to its specification. An error is a part of the system state that subsequently may result in a system failure. This implies a sequence of state transitions causing a system to fail eventually, however, the presence of an error does not necessarily cause a system failure. Errors are potentially observable and detectable since they are part of the system state as the system's data while the cause of an error is a fault. The presence of a fault in the system does not imply the manifestation of error because faults may or may not cause errors. However, the inverse of it holds; the existence of an error in the system state implies the existence of a fault.

Figure 2.1 illustrates an example of the dependencies among fault, error, and failure. It initiates with the failure of one of the distributed system's components, which

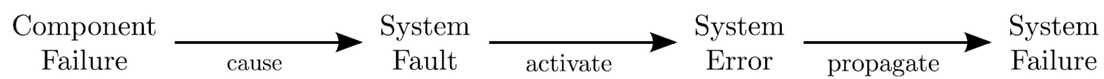

FIGURE 2.1: Dependency among fault, error, and failure [20]

causes a fault in the system. This fault may lead to an error in the system, which may subsequently cause a system failure.

2.3 Fault models in the literature

No reliable system can be constructed from imperfect components to tolerate arbitrary numbers or severe component failures. In this regard, a decision must be made about the component failures and the system tolerance level w.r.t. those failures. For this, a failure model specifies the failures a system is supposed to tolerate. The formalization of failure assumptions is categorized into functional and structural failure models [21].

2.3.1 Functional failure model

A functional fault model defines the semantics of failures that are supposed to be tolerated by the system. This includes the specification of the behavior a failed component may exhibit. The failure classes of crash, omission, timing, and byzantine failures, identified by [22] are illustrated in Figure 2.2 as an ordered hierarchical inclusion concerning their severity. A failure class includes another if the behavior of a failed component in the latter is also possible in the former failure class. For instance, the timing failure class is included in the Byzantine failure class, but not in the omission failure class.

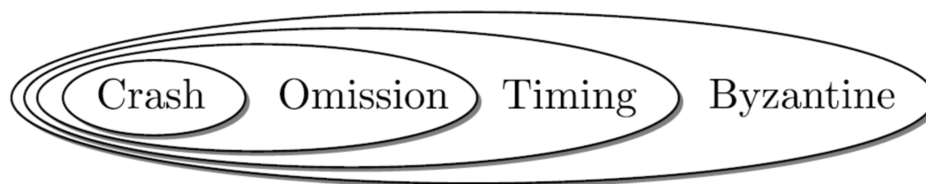

FIGURE 2.2: Hierarchical ordering of functional fault classes [23]

Crash failure

Such a failure occurs when a component fails w.r.t. the failure semantics of the crash failure class, which means either it behaves correctly (adhering to the specification) or has crashed (halted or stopped to proceed). Furthermore, [24] differentiates the crash failure class based upon the state of the failed component, having recovered it. Having recovered the failed component, if it is restarted from its initial state - that is not dependent on the progress the component has made while being non-failed – an amnesia-crash of the component has happened. The recovered component is unaware

of its failure since it is unable to differentiate between failing and subsequently being repaired or starting from its initial state. An amnesia-crash failure may render a reliable channel unreliable because the message buffer of the channel is reset to the initial (empty) state. Thus, possibly enqueued messages are lost and never delivered, hence, violating the reliability.

A partial-amnesia-crash failure occurs when after recovery, some parts of the component's state reset to the initial (empty) state whereas others adhere to the state prior to failure.

A failed component suffers the pause-crash failure if it restarts in the state it was prior to its failure. Also, pause-crash failures and partial-amnesia-crash failures need the state of the component to be preserved upon failure in order to restore it for the recovery, for instance, by utilizing a stable storage [25], [26]. In addition, the recovering component is aware of the fact of failure in restoring the state from stable storage. A halting-crash failure occurs when a crash-failed component fails to recover and, hence, remains failed for an indefinite time.

Henceforward, the term crash failure is identified with pause-crash failure in case not specified explicitly otherwise. Pause-crash failure manifests a fail-silent behavior as the system fails silently; therefore, the difference of it with the fail-stop failure is that the fail-stop can be detected by the alarm.

Omission failure

If a non-failed process or channel fails to execute the instructions of its local program, and respectively, fails to deliver one or more messages, then this is called an omission failure. An omission failure renders a reliable channel an unreliable channel as it loses one or more messages. This deletes them from the message buffer and, hence, violates the reliable delivery requirement.

If an omission-failed process always fails to execute instructions, which means making no progress, then this is halting-crash-failed. Also, if a channel does neither receive nor deliver messages, then this is a halting-crash failure.

Timing failure

A timing failure occurs when a component functions correctly but does not adhere to the timing restrictions w.r.t. the job, so it does not meet the expected time frame. Components that fail according to their failure semantics if the timing failure class provides, a very delayed response, too early, or even no response. However, the latter resembles an omission failure.

Byzantine failure

The failure falls in the category of the Byzantine failure class, when it may exhibit an arbitrary behavior while failing. This may include correct, but unusual or malicious behavior. Thus, Byzantine-failed components are also known as fail-uncontrolled

components [27]. A Byzantine failure may cause arbitrary errors in the time, as well as value domain of data [28], thereby allowing the byzantine-failed component to intentionally destroy the distributed computation. Most importantly, a Byzantine-failed component may also emulate the failure behaviors of the crash, omission, and timing failure classes.

Besides, there are incorrect computation failures, resulting in incorrect values. However, our problem lies within the realm of crash-failure (fail-silent) or fail-stop behaviors. The specification of mandatory component failures such as quantitative failures, referring to the frequency of the replica failures, is defined by a structural failure model as described next.

2.3.2 Structural failure model

A structural failure model explains which components may fail, the number of possible failed components in a run of the distributed system, and also their types. If merely nodes are subject to failure whereas channels are not, the structural failure model is called a process failure model. On contrary, if nodes are perfect while channels are subject to failures, the structural failure model is said to be a channel failure model. If both, the nodes and the interacting channels are subject to failures in a run, the structural failure model is said to be a combined failure model.

The set of failed components in a run is termed the failure scenario for that particular run.

Threshold-based structural failure model

A basic example of a structural failure model is the so-called threshold model, which [29], [30] specify that max. t out of n components such that $t \leq n$ may fail in a run. For instance, assume a system comprising three nodes (replicas) *p1*, *p2*, *p3* that are connected with each other through perfect channels and a structural failure model specifying that max. one node may fail in a run. This failure scenario is described by a threshold model with a threshold of $t = 1$. Presumably, because of the knowledge about the concrete application scenario, the actual failure specification is improved to furthermore incorporate the case that *p1* and *p2* may fail in a run. This improved failure specification cannot be accurately described by a threshold model. It is not possible to define the additional failure scenario for *p1* and *p2* being failed without increasing the threshold to $t = 2$, thereby also defining the failure scenarios of *p1*, *p3* and *p2*, *p3* failing in a run. Threshold models can easily underspecify or overspecify the failures a system may face: Either a failure scenario that may happen is not defined or a failure scenario that may not happen is defined and unnecessarily respected in the structural failure model. Thus, a threshold model defines the possible failure of arbitrary subsets of the set of components whose cardinality is less than or equal to t.

Set-based structural failure models

Such failure models are, for instance, adversary structures [31] or DiDep [32], allowing a more fine-grained and comprehensive structural failure specification through the sets

of the subsets of components subject to failure in a run. Such type of failure models strictly generalizes threshold models because they can be represented by sets of subsets of components. Reconsidering the above example, the set-based structural failure model $\{\emptyset, \{p1\}, \{p2\}, \{p3\}, \{p1, p2\}\}$ specifying that either no nodes, exactly one node, or only *p1* and *2* are subject to failure in a run, which precisely describe the refined failure specification.

Furthermore, set-based structural failure models are able to describe dependencies within component failures that threshold models are unable. For instance, in the set-based structural failure model, the failure of *p1* and *p2* avoids the failure of *p3* in the same run. Assuming replica *p1* cannot fail on its own, that is, the structural failure model becomes $\{\emptyset, \{p2\}, \{p3\}, \{p1, p2\}\}$, then its failure also prevents the failure of *p3*, however, enforces the failure of *p2* in the same run. Informally, an independent component failure does neither prevent nor enforce the failure of other components. The structural failure model where the three replicas *p1*, *p2*, *p3* may fail independent of each other is $\{\emptyset, \{p1\}, \{p2\}, \{p3\}, \{p1, p2\}, \{p1, p3\}, \{p2, p3\}, \{p1, p2, p3\}\}$.

Assumption coverage

Even in a careful structural failure model, the risk of underspecification of the intensity of failures remains. Such risks are expressed by the notion of assumption coverage [28], the probability that no more failures will occur than the ones specified in the failure scenarios. The assumption coverage is a probability; therefore, must be less than 1 as some failure scenarios may be left unspecified, either deliberately or unintentionally. If all the prospective failure scenarios were specified in the structural failure model, the model would perhaps be of no use because it would become too complex, in addition to describing too severe failure scenarios for a system to tolerate. For instance, the failure of all components of the system can be specified in the structural failure model, but such a failure scenario cannot be tolerated by any system comprising imperfect components – at least not if the components are failed for a longer period of time.

Network Partitioning

The failure of channels alters the topology (which can also be due to the failure of nodes) of the distributed system, thereby impacting also the communication between the replicas. This can also be due to the failure of nodes. This as a result affects the progress of the distributed computation. For instance, if the two channels, i.e., in the ring topology shown in Figure 2.3, connecting *p1* and *p2* and *p5* and *p6* fail, the network breaks into two partitions. However, when the inter-partition communication is ruptured, it does not affect the intra-partition communication among replicas. If the two channels suffer a halting-crash and hence fail permanently, then the system permanently fails, too, provided that the replicas of both the partitions cooperate for the distributed computation to progress. If, on the contrary, the failed channels finally recover, then the progress of the distributed computation is not rendered impossible but only delayed till at least one of the failed channels gets back – and no intermittent failures occur – such that the partitions rejoin.

As for the fault model for this research, it falls in the realm of the functional failure model, the replicas are supposed to manifest a fail-silent behavior. All failures are

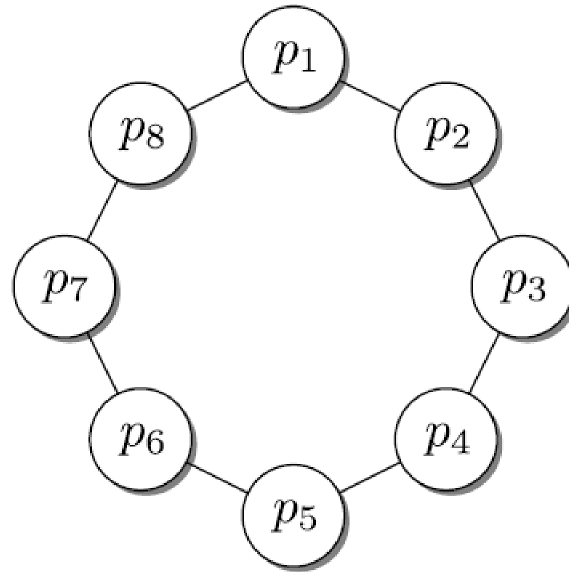

FIGURE 2.3: Ring topology [20]

assumed to be independent of each other. The network is supposed to be fully connected without communication failures. A network partitioning may occur; however, channel failures are not part of the chosen fault model in order to keep the analysis less complicated. As for the network partitioning, it does not harm the proposed mechanism in this research as the mechanism works on the quorum of replicas independent of the network partitioning. Only nodes (machines) hosting replicas can fail and the probability that a node has failed at any particular point in time is (1-p). p gives the probability that a node is available at an arbitrary point in time. So, this fully connected behavior with no communication failures is not necessary for correctness purposes but rather for analysis purposes, which means with such assumptions it is easier to carry out the experiments and analysis.

2.4 Replication strategies and related work

DRSs in general are categorized into two major classes: unstructured and structured DRSs. Unstructured DRSs, for instance, the Majority Consensus Strategy (MCS) [5] uses combinatorics and minimum quorum cardinalities to specify a quorum system. The MCS requires $\lceil n/2 \rceil$ replicas for the read quorum and $\lceil (n+1)/2 \rceil$ for the write quorum to execute any operation in a system comprising n replicas. This threshold-based quorum system allows all the replicas an equal opportunity to be in a read or a write quorum. However, it succumbs to high operational cost and scalability issues because of linearly increasing quorum cardinalities. This is not the case in structured replication strategies, where structural properties and patterns are used to specify a quorum system. For instance, the Grid Protocol [4] imposes a logical rectangular i * j grid structure where i indicates columns and j rows for a system comprised of i * j = n replicas. A read quorum consists of replicas from at least each column while a write quorum constitutes all the replicas at least from a column along with one replica from each column to satisfy the quorum system intersection property. There are various

other replication strategies known from the literature such as Read-One Write-All (ROWA) [33], the MCS [5], the Tree Quorum Protocol (TQP) [3], the Weighted Voting Strategy (WVS) [34], Hierarchical Quorum Consensus (HQC) [35], the Grid Protocol [4], and the Triangular Lattice Protocol (TLP) [36]. Details and the working logic of these strategies will be discussed later in this chapter. These structured, as well as unstructured strategies, are also known as static replication strategies. Besides, there exist also dynamic replication [37], [38], [20] based upon static replication strategies, enabling the adaptation of the quorum, thereby allowing switching among quorum systems at run-time. This dissertation will act as a building block to dynamic replication, too, as conceptually, a dynamic DRS is a set of static replication strategies where each represents the quorum system for a particular subset of the set of replicas among which the system may switch at run-time. These strategies constitute different semantics, patterns, and properties in terms of accessing replicas, thereby resulting in different thresholds of objectives, i.e., availabilities, costs, total replicas, etc.

Moreover, static DRSs are not able to tolerate replica failures beyond a strategy-specific threshold, however, dynamic DRSs can switch among quorum systems to adapt to failures. In this regard, the proposed mechanism will provide new replication strategies whenever a replica failure is detected, thereby making a quorum system switching possible. So there exist these DRSs to control the replication behavior in distributed systems, but as mentioned before, there are trade-offs, too, of several quality metrics, thereby making numerous scenarios between them, which leads to the need of designing new DRSs since there is no single best solution, and existing DRSs are insufficient to fulfill the scenarios entirely. This research uses a so-called hybrid approach where different strategies are "glued" together to form a new DRS. The state-of-the-art has not much focused on a hybrid approach to explore new strategies. So, there is not much work done on such a hybrid approach, yet, there exist only a few attempts in the literature such as [39] and [40], which merely combine TQPs and GPs, but they do not impose any unified structure on the nodes that greatly limits the operability of the approach. Figure 2.4 shows different topologies, semantics, and patterns of replication strategies for accessing replicas, thereby indicating their diverse nature. This diversity makes the task of developing a hybrid approach very cumbersome to pursue. It leaves less room for a hybrid approach to work effectively as it cannot incorporate the varied strategies freely. As a consequence, many scenarios could be left unaddressed. Whereas, to address this issue, if a hybrid approach is applied to such a diverse nature of topologies and varied patterns for accessing replicas, the problem easily goes out of hand.

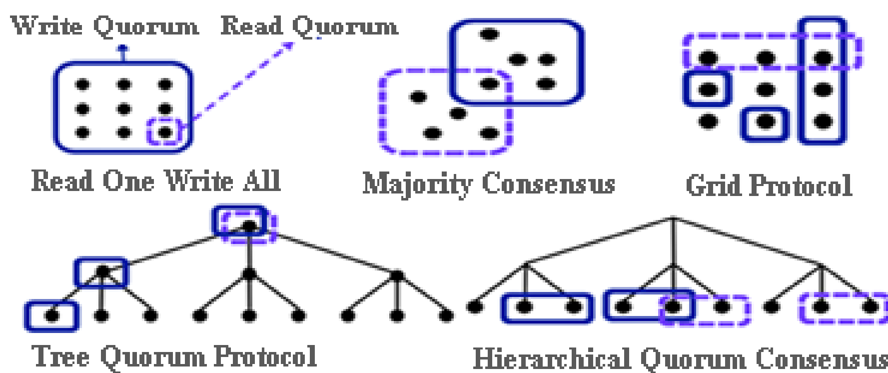

FIGURE 2.4: Contemporary replication strategies [41]

To solve such problems, in the context of this dissertation, lies in the realm of an optimization problem that uses so-called voting structures to achieve a holistic hybrid approach in DRSs, thereby ignoring all their logical imposition in anticipation of making the task easier to accomplish. Though expert-based manual designs of optimized DRSs using the concept of voting structures have been presented in [42], [12], [13], but lack automation, which limits the efficiency of the approach since the search space is huge. The state-of-the-art approaches in optimization and machine learning include multi-objective evolutionary algorithms, stochastic optimization techniques [43] for quantification performance measures, etc. As mentioned earlier, this dissertation is focused on multi-objective optimization and stochastic techniques, particularly, genetic programming [44], [45] to solve the problem. Besides multi-objective optimization, there exist other approaches, i.e., constraints optimization, multimodal optimization, combinatorial optimization, etc. Also, some other recent works on quorum optimization can be found here [46], [47].

Initially, in Section 4.4, newly designed DRSs are demonstrated using voting structures, particularly focusing on MCS, GP, and TLP, following the same line of research as in [42], [12], [13]. And this, later on, acts as a building block in the automatic design of such strategies so that any quorum-based strategy can freely be merged with any other quorum-based strategy. It endeavors to automatically design new solutions and optimize them through machine learning to satisfy the specified scenario, hence, assisting multi-criteria decision making. It uses a genetic programming-based approach that enables the system to design holistic hybrid DRSs at run-time and optimize them as computer programs over several generations of evolution. An optimized strategy from the generated strategies can be picked at run-time depending upon the scenario and preferences of certain objectives. This genetic programming-based mechanism is also endowed with strong multi-crossover as well as multi-mutation operators to easily and also, flexibly design innovative replication strategies.

2.4.1 Voting Structures

To address the described topological and diversity issues between DRSs, a unified representation of these strategies by a concept like General Structured Voting [11] is required for the simulation and machine learning approaches to be applied over it. Expert-based manual designs of optimized DRSs using the concept of voting structures have been presented in [42], [12], and [13]. Figure 2.5 represents a quorum system by a directed acyclic graph (DAG) named a voting structure. Every individual voting structure, in our case, is a computer program that is interpreted by a general algorithm given in [11], at run-time to derive read and write quorum sets. These quorum sets are used to manage replicated objects. A voting structure is traversed recursively by the algorithm to derive the quorums for respective access operations at run-time independent of the varied topologies of the strategies. The nodes of a voting structure are either physical nodes representing actual replicas or virtual nodes that constitute the groupings of physical and virtual nodes. The virtual nodes are labeled V_i, where i = 1, 2,... while the physical nodes are labeled p_j where $1 \leq j \leq n$ and n represents the total number of replicas of a system. Irrespective of being a physical or virtual node, every node is endowed with votes comprised of a natural number on the right) which could also be comprehended as the weightage of that node in the collection of the quorum. Furthermore, each node is equipped with a pair of minimal quorums rq (wq) for the read and write operations to collect from its child nodes. The minimal quorums for

each node to gather per operation have to be less than or equal to the sum of the votes of its children. Some replication strategies, i.e., the Tree Quorum Protocol imposes a partial order on the quorums by which to use quorums for operation execution. The specification of such an ordering allows certain quorums to be used prior to others. In such cases, the directed edges of voting structures can be marked with operation-specific priorities imposing such orderings with 1 being the highest and ∞ being the lowest). This ordering is for accessing the replicas accordingly to reduce the cost. This voting structure is traversed by the recursive algorithm to derive respective quorums. It starts from the root node and queries as many of its child nodes as specified in the minimal quorums to orchestrate the quorums of physical replicas for the respective access operations. On each level, in the case of a tree structure, the quorums have to abide by the conditions (2.1) and (2.2) for a total number of votes V to meet the consistency criterion. Here, rq (wq) is a number representing the minimal read (write) quorum.

$$rq + wq > V \text{ (to avoid read-write conflict)} \tag{2.1}$$

$$wq > V/2 \text{ (to avoid write-write conflict)} \tag{2.2}$$

For example, the voting structure given in Figure 2.5 constructs the following read (RQ) and write quorum sets (WQ) to perform the data access operations:

$RQ = \{\{p1\}, \{p2, p3\}, \{p2, p4\}, \{p3, p4)\}$
$WQ = \{\{p1, p2, p3\}, \{p1, p2, p4\}, \{p1, p3, p4\}\}$

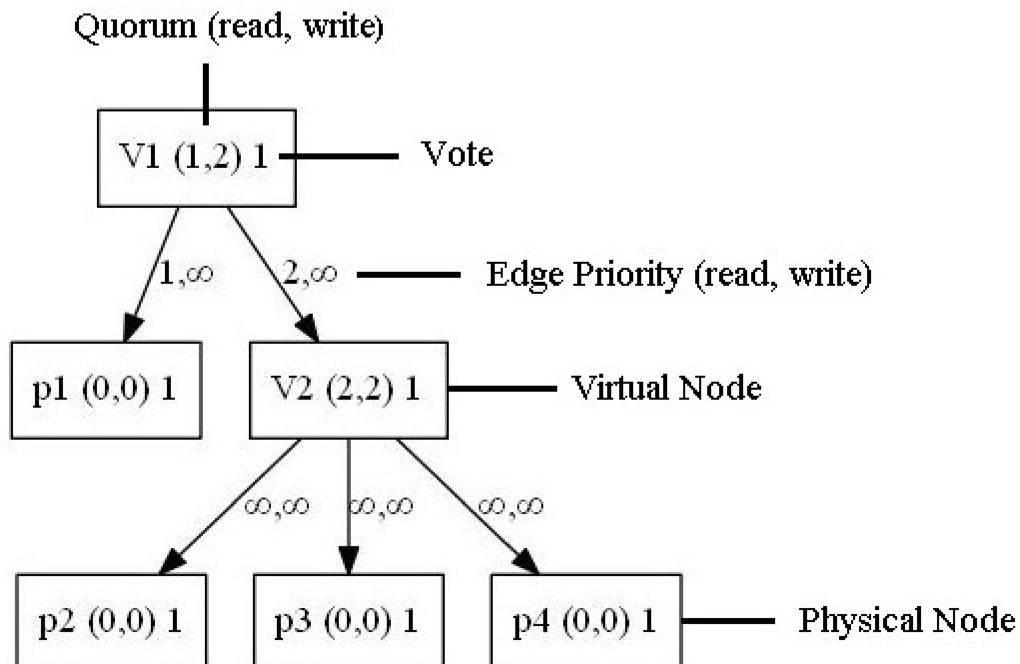

FIGURE 2.5: Example of a voting structure

Figure 2.6 demonstrates MCS, TQP, ROWA, GP, and TLP (from left-top to right bottom) comprising four replicas as a modeled unified representation in the form of voting structures, respectively. The respective quorums and votes are set in the instances; however, for simplicity, edge priorities are not represented in the figure and the interpreting algorithm takes care of the order of replicas inherently. These voting structures forming quorum systems eliminate the diversity between the replication strategies since the same quorums would be derived recursively here, as it would be in an orthodox representation; therefore, this representation is immensely powerful and the key to the proposed automatization approach. Next, the details of some of the common data replication strategies are explained.

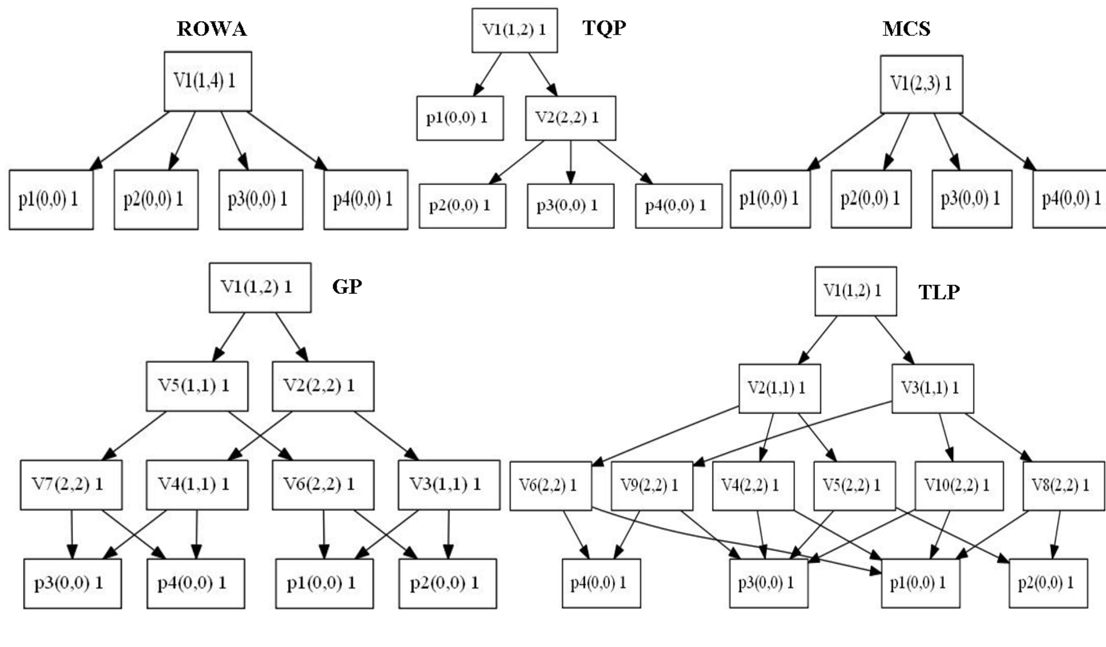

FIGURE 2.6: DRSs being represented as voting structures

2.4.2 Read-One Write-All Protocol

This perhaps is the most intuitive and simplest data replication strategy known as Read-One Write-All Protocol (ROWA) [2]. ROWA is an unstructured replication strategy where the write quorum set consists of a single quorum that contains all n replicas in the system while the read quorum set consists of n quorums, each comprising a distinct single replica. Such a construction satisfies the quorum system intersection property. On one hand, the write operation needs all replicas of the system to participate in the operation execution, thereby not tolerating even a single replica or channel failure. This may result in some replicas being unavailable or unreachable by other replicas via the network. On the other hand, with such a read quorum cardinality, read operations can easily be executed locally without involving any other replicas. Certainly, ROWA is strictly biased towards the read operation in every aspect. It is better for application scenarios where the read operations are by far more frequently performed than write operations, and where the read operation availability is of extreme importance because of the frequency of the read operations. The Read-One Write All Available Protocol is proposed to improve the write operation availability [2]. In this strategy, write operations are executed on all the available replicas of the single write quorum, which

means, on all the non-failed replicas contrary to ROWA where all the replicas are required. As a result, a formerly failed replica may return a stale replica value if a (local) read operation is executed on it upon its recovery, and a write operation has been executed while this replica was failed. Thus, upon recovery, the replica must be aware of the failure and must carry out a scheme-specific recovery protocol to update its replica before executing regular access operations. [7, 48] consider the Read-One Write All Available Protocol to be the best option for a range of cluster computing applications with its specific communication network infrastructure and network topologies.

2.4.3 Majority Consensus Strategy

Majority Consensus Strategy (MCS) [5] uniformly allocates each replica a vote of 1. The read quorum threshold of votes is $\lceil n/2 \rceil$ and the write quorum threshold of votes is $\lceil (n+1)/2 \rceil$ for a total of n votes and replicas in the system. As $\lceil n/2 \rceil + \lceil (n+1)/2 \rceil > n$ and $2 \cdot \lceil (n+1)/2 \rceil > n$, the quorum system intersection property is met for the read and the write quorum set. It does not impose any specific order on the quorum sets to be probed accordingly. Therefore, the computational work induced on the replicas by the quorum probing strategy can be balanced among them. Due to the quorum system specification of vote thresholds, any replica is equally appropriate for being in a particular quorum as not some specific replicas but several replicas make a quorum. Therefore, MCS is highly resilient to replica failures since as far as the number of non-failed replicas meets at least the minimal votes necessary for a read or a write quorum, an arbitrary set of replica failures can be tolerated. As a result, MCS offers very high operation availability for both, the read and the write operation, provided that the individual replicas' availability exceeds 0.5. However, quorum cardinalities grow linearly in the number of replicas, thereby resulting in only linear scalability, for instance, in relation to message complexity.

2.4.4 Weighted Voting

Weighted Voting [34] generalizes MCS by allowing the allocation of a specific number of votes to each replica, thereby rendering MCS with its uniform vote assignment a special case of Weighted Voting. It is hence dedicated to be used where the replica availabilities are not uniform but divergent since replicas with higher availabilities can be prioritized over replicas with lower availabilities by assigning them more votes to potentially increase their operation availability. Besides, the quorum probing strategy can access computationally more powerful replicas by assigning them more votes than less powerful replicas. For instance, in terms of message complexity, it is suitable for the replication strategy to favor quorums with small cardinalities, resulting in those replicas with more assigned votes in order to be selected more often than other replicas (with fewer votes) to participate in the operation execution. However, finding a suitable vote assignment is not a trivial matter [49], [50], [51]. Depending upon the particular vote assignment, quorum cardinalities may increase linearly in the number of replicas, which results in scalability issues the same as in the case of MCS.

Weighted Voting Protocols are optimal for the access operations availability in a fully connected network topology with perfect channels while assuming a replica availability of at least 0.5 and replica failures to be independent [52]. Next, structured static

replication strategies are discussed with the first being the Tree Quorum Protocol [3], followed by the Grid Protocol [4], and the Triangular Lattice Protocol [36].

2.4.5 Tree Quorum Protocol

The structured Tree Quorum Protocol (TQP) [3] logically organizes the replicas in a tree structure as shown in Figure 2.7 by an example with 13 nodes. A read quorum is derived from the tree structure by probing it in a certain manner. Initially, the read quorum comprises the root node on level zero of the tree. If this node has failed, then as a substitute, a preferably non-failed majority of its child node on level one of the tree is used. For each failed node in the selected majority, this failed node is again substituted by a (preferably) non-failed majority of its child nodes on level two of the tree, and so forth. The traversal ceases if 1) a set of non-failed nodes are assembled constituting the read quorum or 2) a failed node cannot be replaced by a majority of its child node, either because the failed node is a leaf node or too many of its child nodes have failed too. In the latter case, the read quorum becomes unavailable.

For instance, in Figure 2.7, if the nodes *p1* and *p3* fail, the set {*p2*, *p4*} is a valid read quorum. If the node *p4* fails too, the set {*p2*, *p11*, *p12*} becomes a valid read quorum. The TQP imposes an ordering on the read quorums to probe them accordingly. A write quorum comprises the root node plus a majority of its child nodes plus a majority of their respective child nodes, and so forth. For instance, the set {*p1*, *p2*, *p3*, *p5*, *p6*, *p8*, *p9*} is a valid write quorum of the TQP with 13 nodes as shown in Figure 2.7.

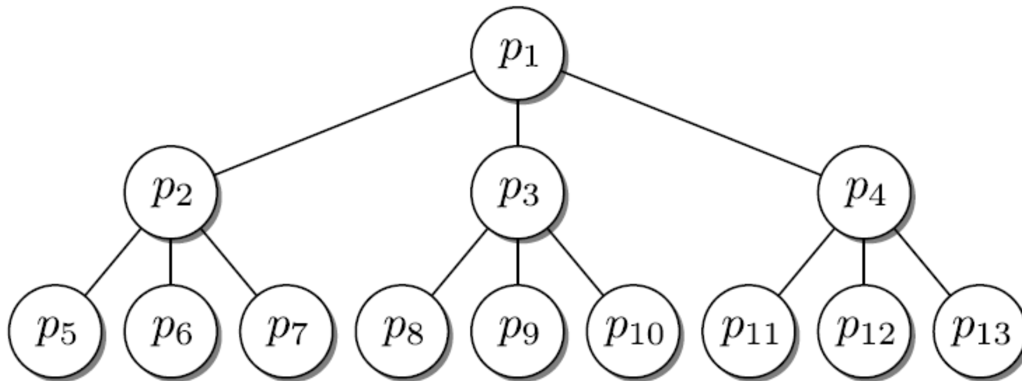

FIGURE 2.7: Tree quorum protocol [20]

Note that quorum systems following the TQP cannot be represented by vote-assignable quorum sets [3].

2.4.6 Grid Protocol

The Grid Protocol [4] arranges the nodes in a logical rectangular $k \times j$ grid with $k \cdot j = n$ for a system consisting of n nodes where k is the number of columns and j is the number of rows in the grid. Figure 2.8 illustrates the logical 3×3 grid for a system comprising nine nodes (replicas). A horizontal crossing of the grid from the leftmost column to the rightmost column is called a column cover (C-Cover). A vertical crossing

of the grid from the topmost row to the bottom-most row exclusively following vertical edges in such way that it includes all nodes of a single column, which is called a complete column cover (CC-Cover). For instance, the set of nodes {*p3*, *p6*, *p9*} represent a CC-Cover and the set of nodes {*p1*, *p5*, *p6*} form a C-Cover in Figure 2.8.

In the original Grid Protocol [4], a read quorum comprises one node from each column (C-Cover) while a write quorum necessitates both, a C-Cover and an additional CC-Cover to satisfy the quorum system intersection property. The read and write operation availability graphs of the Grid Protocol are not symmetric and, therefore, are not optimal [8]. The optimized version of the Grid Protocol [53], [54], [55] further allows a CC-Cover to be a read quorum. This seemingly trivial improvement increases the probability of finding a read quorum, thereby increasing the read operation availability while not compromising the write operation availability. If a complete column of nodes fails, the original Grid Protocol cannot form a C-Cover, which consists of a read quorum, as a consequence, the read operation becomes unavailable.

The optimized version, however, can use another complete column (CC-Cover) comprising non-failed nodes as read quorum, which keeps the read operation available. The optimized Grid Protocol holds point-symmetric read and write operation availabilities and, therefore, is optimal [8].

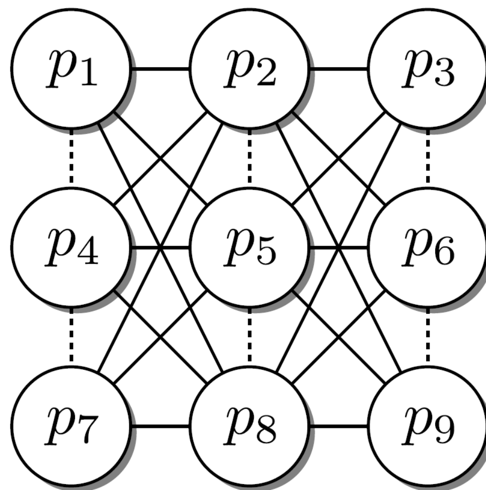

FIGURE 2.8: Optimized Grid protocol of 3×3 [55]

2.4.7 Triangular Lattice Protocol

The Triangular Lattice Protocol (TLP) [36] is another structured grid-based protocol as shown in Figure 2.9 comprising nine replicas, in this case. Following the Grid Protocol nomenclature, a horizontal crossing of the lattice structure is called an H-Cover and a vertical crossing is known as a V-Cover. On contrary to the Grid Protocol that imposes no restriction on which replicas form a horizontal path as long as one replica from each column is included, the TLP limits the replicas being allowed in a horizontal path. For each replica and its successive replica in a horizontal path, the successive replica is either its left-adjacent, right-adjacent, upwards-adjacent, downwards-adjacent, right downwards diagonal-adjacent, or left upwards diagonal-adjacent replica in the lattice structure. Besides, contrary to the Grid Protocol, a vertical crossing of the TLP from

the topmost row to the bottom-most row must not only follow vertical edges, but the same rules of horizontal paths also apply to each replica and its successive replicas in a vertical path.

A read quorum either compromises replicas on a horizontal path from the leftmost column to the rightmost column of the lattice (H-Cover) or replicas on a vertical path from the topmost row to the bottom-most row of the lattice (V-Cover). A write quorum needs both, the replicas of a horizontal as well as a vertical crossing of the lattice. For instance, in the logical lattice structure in Figure 2.9, the replica set {*p4*, *p5*, *p9*} is an H-Cover while the replica set {*p1*, *p5*, *p8*} is a V-Cover. If existent, the diagonal path in a lattice – the replica set {*p1*, *p5*, *p9*} in the figure, is a V-Cover as well as an H-Cover. Therefore, it can be used to execute read as well as write operations.

It means in this diagonal case, only three replicas instead of five are required to perform the write operation if this replica set is used as a write quorum. The TLP construction is complete [55] in the sense that if an edge is added to the structure, the quorum system does not adhere to the intersection property anymore. For instance, adding an edge connecting the *p2* and *p4* in Figure 2.9 results in a possible vertical and horizontal crossing with not any replicas being common in both, as the vertical path, in this case consists of the replica set *p2*, *p4*, *p7* whereas the horizontal path comprises *p1*, *p5*, *p6*, hence, having no replica in common, consequently impacting the correctness behaviour.

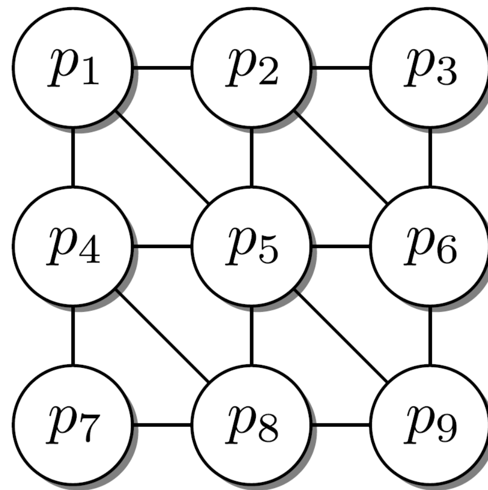

FIGURE 2.9: Triangular lattice protocol of 3×3 [20]

2.5 Summary

The concept of replication is to increase fault tolerance, in particular operation availability, which inheres to two facets, namely redundancy in components and a coordination protocol managing the multiple redundant components. Applied to distributed computer systems, replication is used to improve the read and write operation availability on critical data objects at a relatively lower cost. In this regard, different fault models and state-of-the-art replication strategies are explained in this chapter. Also, the novelty of the proposed approach w.r.t. the existing methods is explained that this work

uses so-called voting structures to automatically achieve a holistic hybrid approach in DRSs, which ignores all their logical imposition in anticipation of making the task easier to achieve. It endeavors to automatically design new solutions and optimize them through machine learning to satisfy the specified scenarios, hence, assisting multi-criteria decision making. Having discussed the replication strategies and related work, now we move on to discussing the machine learning aspect of the proposed approach. It uses a genetic programming-based approach that enables the system to design holistic hybrid DRSs at run-time and optimize them as computer programs over several generations of evolution.

Chapter 3

Machine learning and genetic programming

The chapter briefly explains the most common machine learning techniques, subsequently narrowing it down to genetic programming and its common variants, and the way genetic programming can be used in the context of data replication and fault tolerance in distributed systems.

3.1 Basic concepts of machine learning

Machine learning combines statistics and learning paradigms of artificial intelligence. It is the learning from data and observations, and it has evolved into a very successful area of research in the last few decades. Depending upon the problem, nature of the data, and its goals, machine-learning methods are applied. The most commonly known techniques are supervised learning [56], unsupervised learning [57], semi-supervised learning [58], reinforcement learning [59], and evolutionary algorithms [60] (also known as evolutionary computation).

Supervised learning means learning from data with class labels. For this, normally, data is split into two parts, training and testing data, to train the model (on the training data) and test it (on the testing data), respectively. Labels are, hence, additional information available for training the data. Such type of learning is for task-oriented problems, and the task is generally to predict the unknown class labels. In this regard, if the labels are binary or discrete, the learning task is a classification problem, i.e., classification of cancers [61]. On contrary, if labels are continuous, the task is called a regression problem, i.e., traveling time estimations [62].

Unsupervised learning is the learning solely from the structure of the data itself, without any labels. Hence, in the case of unsupervised learning, the data is not labeled. Such type of learning is for data-oriented problems, i.e., customer segmentation. Clustering and dimensionality reduction are the two variants of unsupervised learning. Hence, in unsupervised learning, there are no labels present for all the observations in the dataset unlike supervised learning, whereas a semi-supervised learning [58] falls in between these two. In many realistic scenarios, the labeling cost is very high since it demands skilled human experts, i.e., financial fraud detection, etc., where there can be millions of unlabeled records, too, alongside [63]. So, in the absence of labels in the majority of

the observations, and their presence in a few, semi-supervised algorithms are the best option.

Reinforcement learning is a type that allows machines and software agents to determine the ideal behavior automatically in a particular context to optimize performance. It is based upon simple reward feedback called reinforcement signal, being essential for the agent to learn its behavior. Reinforcement Learning is defined by a particular problem, and all its solutions are categorized as Reinforcement Learning algorithms. In a problem, an agent is supposed to decide and choose the best action based on its current state, when repeatedly done, it is known as a Markov Decision Process. To develop intelligent programs known as agents, reinforcement learning follows these steps: 1) input state is observed by the agent, 2) a decision-making function determines to execute an action through the agent, 3) having executed, the action, the agent receives a reward or reinforcement from the environment, 4) the state-action pair information about the reward is stored.

Evolutionary algorithms (EAs) [60] are used to discover solutions to problems humans do not know how to solve directly. Free of human preconceptions or biases, the adaptive nature of EAs can generate solutions that are comparable to, and often better than the best human efforts. It is the study of non-deterministic search algorithms that are based on aspects of Darwin's theory of evolution by natural selection [64]. In this regard, a detailed account of the history of evolutionary computation can be found in the work of [65]. However, it is interesting to note that the idea of artificial evolution was suggested by one of the founders of computer science, Alan Turing, in 1948 [66].

There are many EAs, but most common are Genetic Algorithms (GAs) [67], [68], Genetic Programming (GP) [44], [45], Evolutionary Strategies (ESs) [69], [70], Evolutionary Programming (EP) [71], etc. In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. Genetic algorithms are frequently used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators, i.e., mutation, crossover, and selection. The emphasis is on the role of genetic recombination (often called 'crossover'). In GAs, the solutions are often represented as bit strings. Genetic programming is a variant of GAs in which the solutions being manipulated are computer programs rather than bit strings. So genetic operators, i.e., crossover and mutation are applied to programs rather than bit strings. Genetic programming is demonstrated to learn programs for tasks, i.e., simulated robot control, recognizing objects in visual scenes, etc. Evolutionary Strategies includes a similar mechanism, but using real-valued numbers and mostly relying on mutation. Evolutionary programming was originally designed to evolve deterministic finite automata that accept a set of input strings. Evolutionary programming was later extended for optimization in binary and continuous solution spaces, too, while being equipped with mutation rate adaptation techniques. Next, genetic programming and its different variants are discussed, in detail.

3.2 Genetic programming

Genetic Programming is the automatic evolution of computer programs, the origins of which (and evolutionary computation) go back to the origins of evolutionary algorithms [65]. In 1958, Friedberg designed an algorithm to evaluate the quality of a

computer program, make some random changes to it and then test it again to check for improvements, and so on [72], [73]. Smith used a form of GP in his Ph.D. thesis in 1980 to construct a learning system [74]. In 1981, Forsyth emphasis the utility of GP in artificial intelligence by evolving Boolean expressions for different prediction problems, i.e., prediction of the survival of heart patients, prediction of the British soccer results, and identification of the athletes good at sprinting from those good at longer distances [75]. In 1985, Cramer evolved sequential programs in the computer languages JB and TB [76], the latter has the form of symbolic expression trees. He used a few assembler-like functions, coded as positive integers, with integer arguments. In the same year, being unaware of Cramer's work, Schmidhuber also experimented with GP in LISP, and later reimplemented it in a form of PROLOG [77], [78]. However, GP became more popular after the publication of John Koza's book in 1992 [44]. In general, it is quite challenging to evolve computer programs because of the fact that computer programs are highly constrained and must adhere to a specific grammar for them to be compiled.

3.2.1 Why genetic programming?

Genetic programming is mainly used to optimize computer programs. The reason for choosing it is 1) its ability to evolve and optimize DRSs in the form of DAGs as the encoding scheme used in this research is DAG-based voting structures, 2) to avoid brute force since the search space is huge, which would make the task computationally expensive because there are too many combinations and possibilities by which the DRSs can be combined. Thus, the proposed mechanism allows replication strategies to evolve as computer programs over several generations to attain a constant evolutionary trajectory. Therefore, genetic programming helps in intelligently designing DRSs without attempting to brute force all the possible combinations.

This research, therefore, uses GP to automatically identify and design application-optimized DRSs. As mentioned earlier, GP is a type of EA; however, the major difference between GP and other genetic variants of machine learning is the representation. Furthermore, specifically the difference with GAs, in GAs, an individual is a candidate solution, and individuals are generally "raw data" in some encoding scheme (i.e., a string); however, GP can be considered a special case of GA, in which each individual represents a computer program (i.e., a nested data structure) rather than merely a raw data. Hence, GAs search a solution space whereas GP explores a program space [79]. As GP is used to evolve computer programs, this dissertation as well, therefore, uses it to evolve the generations of replication strategies as computer programs and optimize their constraints to meet the criteria. The criteria are manifested in computable functions known as objective functions (such as explained in the next chapter), which conflict with each other in the real world. The problem is to find a solution that satisfies the given constraints and to optimize a vector function (i.e., a fitness function, described later in the work) whose elements represent objective functions.

3.2.2 GP related work

There are many types of Genetic Programming, i.e, Tree-based GP, Stack-based GP, Linear GP, Grammatical Evolution, Cartesian GP, etc. In tree-based GP, the computer programs are represented in tree structures that are evaluated recursively to produce the

resulting multivariate expressions. In stack-based genetic programming, the computer programs in the evolving population are represented in a stack-based programming language. Linear GP is a subset of genetic programming in which programs in the evolving population are expressed as a sequence of instructions from imperative programming language or machine language. Grammatical Evolution, a novel approach to Genetic Programming that adopts principles from molecular biology coupled with the use of grammars to specify legal structures in a search. Cartesian GP is a very efficient and flexible type of Genetic Programming that encodes a graph representation of a computer program, which is also relevant. Over the years, genetic programming has been shown to handle most (if not all) basic constructs of common programming languages, including functions, iteration, recursion, variables, and arrays. Recently, it has even proven possible to evolve programs in actual programming languages such as Java [80], [81], [82]. Also, GP is nowadays widely applied to image classification [83]. Existing works also show that GP can extract domain-specific features for texture image classification, object classification, scene classification, and even facial expression classification [84], [85], [86], [87]. Next, the common variants of GP are discussed.

Tree-based GP

In tree-based GP, tree structures are used to represent the computer programs, and such tree structures are then evaluated recursively in order to produce the resulting multivariate expressions. Conventional nomenclature states that a tree node (or just node) is an operator [+,-,*,/] while a terminal node (or leaf) is a variable [a,b,c,d]. Figure 3.1 represents a tree-based GP instance comprised of the respective operators and leaf nodes. Also, LISP was the first programming language applied to tree-based GP, as the structure of LIPS matches the structure of the trees.

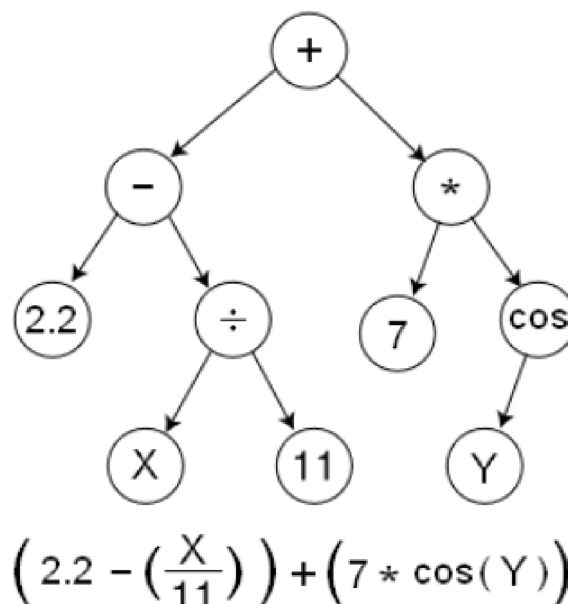

FIGURE 3.1: Tree-based GP instance [88]

LISP was invented by John McCarthy in 1958, and it is one of the oldest high-level computer languages [89]. Even nowadays, it is used widely by researchers in artificial

intelligence. All LISP computer programs can be written in the form of data structures known as trees, which consequently simplifies the task of applying genetic operations to generate valid programs. In 1992, John Koza published a comprehensive work on the evolution of computer programs in the form of LISP expressions [44]. Tree-based GP was the first application of Genetic Programming, but nowadays, many other languages such as C++, Java, Python, etc., are also used in developing tree-based GP applications. As mentioned earlier, there are several other types of GP, i.e., linear, stack-based, cartesian, which are usually more efficient in executing the genetic operators. However, tree-based GP offers a visual means to engage new users of GP and stays viable when implemented through a fast programming language or underlying suite of libraries. For instance, Karoo GP is an example of a scalable, tree-based GP application suite built in Python and the TensorFlow library for multicore and GPU support [66].

EAs typically use recombination (also known as crossover) and mutation operations for generating new prospective solutions. In tree-based GP, crossover means an exchange of subtrees between the parent solutions while mutation substitutes a subtree by a randomly generated one, and details on accomplishing it are well known in literature [44], [45], [90], [91]. Moreover, in tree-based GP, the size of solutions (chromosomes) is variable since crossover and mutation can create offspring solutions of different sizes.

Linear or Machine code GP

Linear genetic programming (LGP) is a subset of genetic programming in which the evolving population is represented as a sequence of instructions from imperative programming language or machine language. In LGP, programs are a constrained linear set of operations and terminals (inputs). These programs are quite similar to the programs written in machine code. In GP, a linear tree is a program that comprises a variable number of unary functions along with a single terminal. Also, linear tree GP differs from the bit string genetic algorithms as a population may comprise programs of variable lengths, and there may also be more than two types of functions or more than two types of terminals [90].

As LGP programs are typically expressed as a linear sequence of instructions, therefore, they are simpler to read and to operate on as compared to the tree-based GP. For instance, Table 3.1 shows a fairly simple program written in the LGP language Slash/A, which is basically a series of instructions being separated by a slash. By converting such code in bytecode format, i.e., as an array of bytes each representing a different instruction, mutation operations can be performed merely by changing an element of such an array.

TABLE 3.1: LGP phenotype instance [88]

```
input/0/save/input/add/output/.

input/    # gets an input from user and saves it to register F
0/        # sets register I = 0
save/     # saves content of F into data vector D[I] (i.e., D[0] := F)
input/    # gets another input, saves to F
add/      # adds to F current data pointed to by I (i.e., D[0] := F)
output/. # outputs result from F
```

Grammar-based approaches

Grammatical evolution (GE) is an evolutionary algorithm, particularly, a GP approach pioneered by Conor Ryan, JJ Collins and Michael O'Neill in 1998 [92]. As compilers use grammar to define the legal expressions of a computer language, therefore, it becomes a natural approach for the GP to explicitly evolve chromosomes adhering to any specific grammar. This implies the evolution of all kinds of constrained structures or languages that can be handled through the same generic approach but with a different grammar. A review of such approaches can be found in [93]. Here, a well-known grammatical approach known as grammatical evolution (GE) [94], [95] is discussed.

In GE, binary-string genomes of variable lengths are grouped into codons of eight bits. The integer value specified by the codon is used through a mapping function in order to choose a suitable production rule from the grammar defined under Backus–Naur form (BNF) [66]. BNF grammars comprise terminals, which are items that can appear in the language (i.e., +, *, x, sin, 3.14, etc.) and non-terminals, which can be further distributed into one or more terminals/ non-terminals. A grammar can be expressed in the form of a tuple (N, T, P, S), where N is the set of non-terminals, T is a set of terminals, P is a set of production rules mapping the elements of N to T, and S is a start symbol that is a member of N. If there exist many productions that could be applied, the option is defined with the OR symbol, "|". An example is shown in Table 3.2.

TABLE 3.2: Sample BNF form [66]

Non-terminals			expr, op, pre-op	
Terminals			sin, +, -, /, *, x, 1.0, (,)	
Start symbol			<expr>	
Production rules	(1)	<expr>::	=<expr><op><expr>	(0)
		|	(<expr><op><expr>)	(1)
		|	<pre-op>(<expr>)	(2)
		|	<var>	(3)
	(2)	<op>::=	+	(0)
		|	-	(1)
		|	/	(2)
		|	*	(3)
	(3)	<pre-op>::=	sin	(0)
	(4)	<var>::=	x	(0)
		|	1.0	(1)

TABLE 3.3: An example genotype in GE. Here, the eight-bit binary codons have all been packed as integers for convenience [66]

220	240	220	203	101	53	202	203	102	55	223	202	243	134	35	202	203	140	39	202	203	102

The mapping of genotype shown in Table 3.3 is carried out as follows. The leftmost non-terminal is selected, and the symbol is noted, i.e., <expr>, <op>, <pre-op>. The codon is denoted by C while the number of production rules for a given expression is denoted by N. Thus, the rule to apply becomes R = C mod N, subsequently, the symbol

is rewritten accordingly. The decoding process continues until no rules can be applied. Also, the genotype is supposed to be circular to enable the first codon to follow the last one. Table 3.4 shows a complete decoding process for the genotype given in Table 3.3.

TABLE 3.4: Decoding the example genotype given in Table 3.3 [66]

Expression	*C*	*N*	*R*	Rule
<expr>	220	4	0	<expr>::=<expr><op><expr>
<expr><op><expr>	240	4	0	<expr>::=<expr><op><expr>
<expr><op><expr><op><expr>	220	4	0	<expr>::=<expr><op><expr>
<expr><op><expr><op><expr><op><expr>	203	4	3	<expr>::=<var>
<var><op><expr><op><expr><op><expr>	101	2	1	<var>::=1.0
1.0<op><expr><op><expr><op><expr>	53	4	1	<op>::= -
1.0-<expr><op><expr><op><expr>	202	4	2	<expr>::=<pre-op>(<expr>)
1.0-<pre-op>(<expr>)<op><expr><op><expr>				<pre-op>::=sin
1.0-sin(<expr>)<op><expr><op><expr>	203	4	3	<expr>::=<var>
1.0-sin(<var>)<op><expr><op><expr>	102	2	0	<var>::=x
1.0-sin(x)<op><expr><op><expr>	55	4	3	<op>::= *
1.0-sin(x)*<expr><op><expr>	223	4	3	<expr>::=<var>
1.0-sin(x)*<var><op><expr>	202	2	0	<var>::=x
1.0-sin(x)*x<op><expr>	243	4	3	<op>::= *
1.0-sin(x)*x*<expr>	134	4	2	<expr>::=<pre-op>(<expr>)
1.0-sin(x)*x*<pre-op>(<expr>)				<pre-op>::=sin
1.0-sin(x)*x*sin(<expr>)	35	4	3	<expr>::=<var>
1.0-sin(x)*x*sin(<var>)	202	2	0	<var>::=x
1.0-sin(x)*x*sin(x)				

Because of the fact that, in GE, genotypes are binary strings, no special mutation or crossover operators are needed. The genotype-to-phenotype mapping process always generates syntactically correct individuals. Besides the standard genetic operators (mutation and crossover), a codon duplication operator is also used. Duplication includes the random selection of a number of codons in order to duplicate them. The duplicated codons are positioned at the end of the chromosome, and the genotype length differs.

PushGP

As for stack-based genetic programming, the evolving population of computer programs is represented in a stack-based programming language. A stack-based computer language known as Push was developed by Lee Spector [96]. His GP mechanism using Push known as PushGP allows several advanced GP features, i.e., multiple data types, the automatic definition of subroutines, and control structures. In the Push family of languages, which were specifically designed for GP, a separate stack is allocated for each data type, and the program code itself can be manipulated on data stacks and executed thereafter.

The Push was also designed by Spector to provide support for the self-adaptive form of evolutionary algorithms known as autoconstructive evolution. The autoconstructive evolution system, as it runs, adapts, and constructs its own mechanisms for reproduction and diversity, which means the methods of mutation and crossover can as well be evolved, unlike others where it is imposed from the start. In such stack-based computer languages, global data stacks are used to pass the arguments to the instructions, which

is different from the argument-passing techniques based on registers. In stack-based argument passing, initially, the arguments are specified or computed to push onto the stack and use those arguments to execute an instruction. For instance, consider adding 3 and 5, which is written in postfix notation as 3 5 +, and this code defines that 3 and then 5 will be pushed onto the stack, and then the + instruction will be executed. The + instruction removes the top two elements from the stack, adds them together, and then pushes the result back onto the stack. Any additional arguments are ignored because every instruction takes only the required arguments from the top of the stack. Also, if a stack instruction contains very few arguments, this will be signaled as a run-time error, and the program may be terminated; however, in Push, such a case will be treated as no operation as instruction with insufficient arguments is simply ignored.

Push handles several data types by offering a stack for every type, i.e., a stack for Boolean values, a stack for float numbers, a stack for integers, a stack for program code (CODE), a stack for data types (called TYPE), etc. Every instruction takes the required inputs from appropriate stacks and pushes outputs onto relevant stacks. The CODE stack enables Push to handle recursion as well as subprocedures. Besides, the CODE stack enables evolved programs to push themselves (whole or parts) onto the CODE stack, it allows programs to specify new genetic operators to generate offspring solutions. A Push language reference can be found in [96].

Depending on the specific language and genetic operators being used, stack-based GP can have several advantages over tree-based GP. These may involve improvements/ simplifications for handling multiple data types, bloat-free (bloat is over-adaptation of chromosomes) crossover and mutation operators, programs equipped with loops providing valid outputs despite being terminated prematurely, the evolution of arbitrary control structures, execution tracing, parallelism, and automatic simplification of evolved programs.

Cartesian Graph-based GP

Cartesian Genetic Programming (CGP) originates from an idea of evolving digital circuits developed by Miller et al. in 1997 [97]. However, the term 'Cartesian genetic programming' first appeared in 1999 [98] and was proposed as a general form of genetic programming in 2000 [99]. It is called 'Cartesian' because it represents a program using a two-dimensional grid of nodes. As mentioned earlier, Genetic Programming works on the automatic evolution (as in Darwinian evolution) of computational structures, i.e., mathematical equations, computer programs, digital circuits, etc. John Koza pioneered a form of GP that uses a tree representation of computer programs, which was inspired by the artificial intelligence computer language, LISP. Following the same line of research, Cartesian Genetic Programming is a highly efficient and flexible form of GP that encodes a graph representation of a computer program.

CGP was invented by Julian Miller in 1999 [66] and was developed from a representation of electronic circuits devised by Julian Miller and Peter Thomson developed a few years earlier. CGP expresses computational structures, i.e., circuits, computer programs, mathematical equations, etc., as a string of integers. These integers, known as genes determine the functions of nodes in the graph, the connections to inputs, the connections between nodes, and the locations output in the graph. As many computational structures can be represented as graphs, therefore, to use a graph representation

is very flexible and effective. A good example of this can be artificial neural networks (ANNs), which can be encoded in CGP.

Graph representations are prevalent and widely used in several areas of engineering and computer science [100], [101], [102]. Indeed, neural networks are also graphs. In literature, the first person to evolve graph-based encodings through Cartesian grids appears to be Sushil Louis in 1990 [103], [104]. In the technical report [103], Louis explained a binary genotype to encode a network of digital logic gates where gates in every column can be connected to the gates in the previous column. Figure 3.2 shows an image extracted from Louis's 1993 PhD thesis [105]. Some works on encoding and evolving ANNs (CGPANNs) using CGP can be found here [106], [107], [108], [109], [110], [111], which are efficient and competitive with respect to other methods of evolving ANNs. Unlike trees where there is always a unique path between any node pairs, a graph allows multiple paths between any node pairs. Assuming that every node carries out some computation, therefore, to represent such functions as graphs are more compact than trees since a graph allows to reuse previously computed subgraphs.

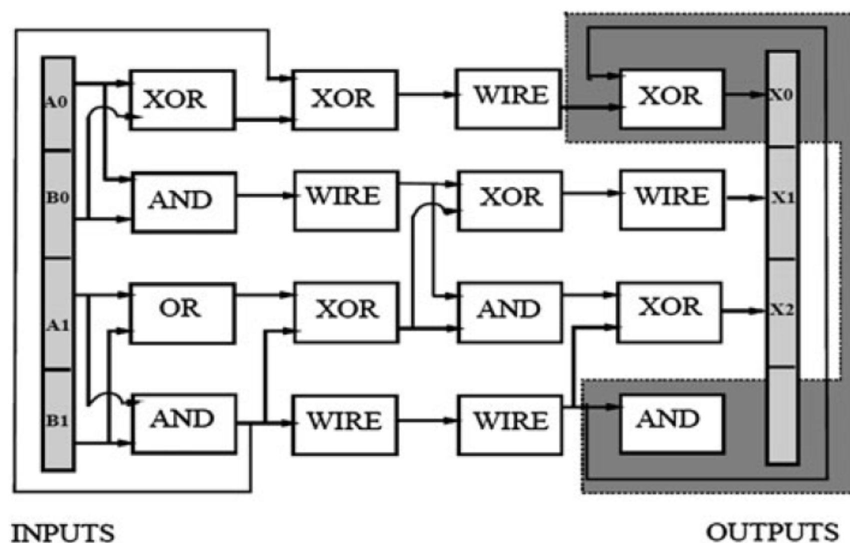

FIGURE 3.2: Sushil Louis's evolved 2-bit adder [105]

Also, Poli proposed a graph-based form of GP (PDGP) inspired by the neural networks [112], [113]. PDGP basically supports the evolution of standard tree-like programs, logic networks, neural networks, recurrent transition networks, and finite state automata. In some of these, it was done by associating labels with the edges in the program graph. Besides, terminal sets and usual functions, PDGP needs the specification of a set of links for determining how nodes are connected. The link labels depend upon what is to be evolved. For instance, in neural networks, labels on the link are numerical constants for the neural network weights.

When PDGP is implemented, the program is represented as an array with a topology of the grid. Each node consists of a function label and the horizontal displacement of the nodes in the previous layer being used as arguments for the function. The horizontal displacement is an offset from the position of the calling node. Terminals or functions are associated with each node in the grid even if they are not referenced in the program path, which are inactive nodes. The basic operator of crossover in PDGP is known as

subgraph active-active node (SAAN) crossover, which is basically a generalization of the crossover used in tree-based GP. Figure 3.3 shows an example of SAAN crossover, which is also is defined as follows, 1) the crossover point is selected as a random active node in each parent, 2) a subgraph in the first parent is extracted, which includes all the active nodes that are used to calculate the output value of the crossover point, 3) the subgraph of the parent solution is inserted into the second parent to create the offspring, and in doing so, if the width of the subgraph exceeds having inserted the node, the subgraph is wrapped around. As for the mutation, Poli used two types of mutation in PDGP. 1) A global mutation inserts a randomly generated subgraph into an existing solution, 2) A link mutation alters a random connection in the graph by initially choosing function node randomly, subsequently choosing a random input link of such a node and, finally, modifying the offset associated with the link.

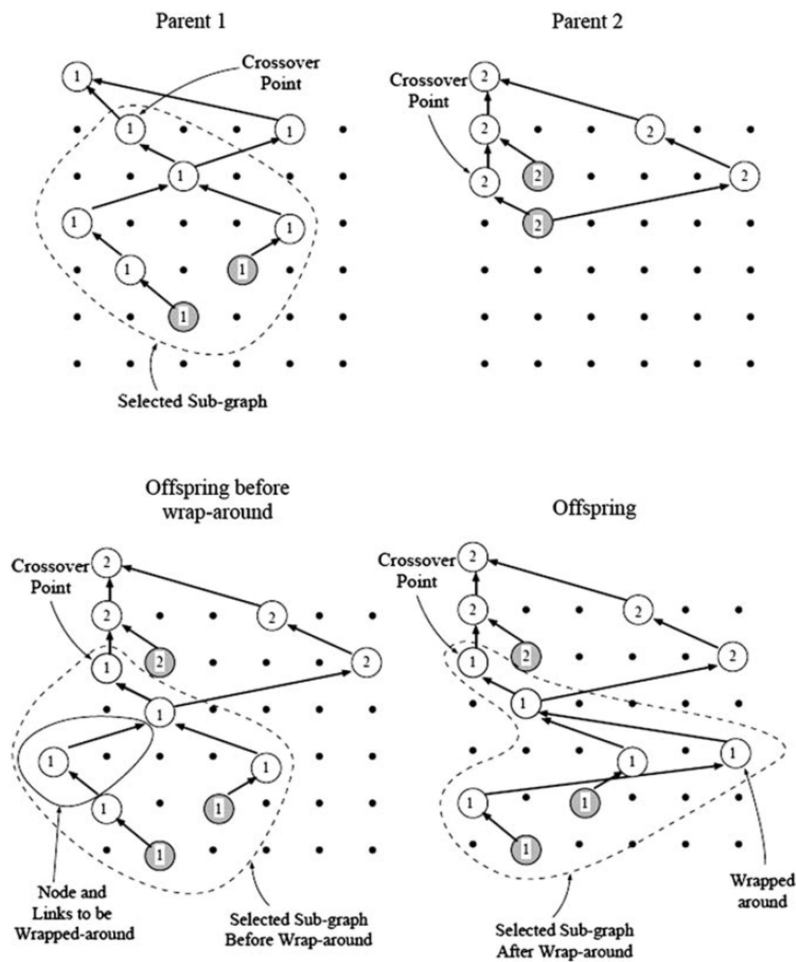

FIGURE 3.3: An example of subgraph active–active node (SAAN) crossover [113]

CGP also encodes directed graphs; however, the genotype is just a one-dimensional string of integers. Besides, CGP genetic operators operate on the chromosome directly while in PDGP they rather operate on the graph directly. Moreover, CGP mostly relies on mutation, particularly, Poli's link mutation as its main search operation; however, there also exist a number of crossover methods for CGP [66]. Instead of offsets, CGP uses absolute addresses for determining where nodes get their input data from. In addition, CGP evolutionary algorithms may also use small populations and elitism as in

ESs. Elitism is keeping the elites in the selection to control the genetic drift. Also, when EAs are applied to different representations of computer programs, a phenomenon known as bloat occurs. This means that the chromosomes become larger and larger as the generations progress, but without any increase in their fitness. Such programs usually have large portions of code comprising inefficient or redundant subexpressions, which cripples the performance since it is time-consuming to process bloat programs. Eventually, an evolved program may also exceed the memory capacity of the system, also, such evolved solutions can be very hard to interpret or understand. Such problems of blot programs, their possible causes, and proposed solutions are discussed here [90], [91], [114]. However, Cartesian GP does not suffer from genotype growth since the genotype is of fixed size. Besides, it does not appear to suffer from phenotypic growth either [115]. Normally, program sizes remain small even when very large genotype lengths are allowed.

Object-oriented GP

Object-oriented software design couples the design of data structures (object classes or types) with methods that operate on those structures, thereby providing better modularity and reuse as compared to non-OO techniques. Object-oriented nature may enable GP to scale up to tackle complex problems that would otherwise be infeasible.

Research in this area is very limited. There has been prior research in the area of object-oriented genetic programming (OOGP) [116], [117], but they are initial-stage research. One of the earliest works in this area can be found here [118], upon which the former two papers were built. However, some of these papers still represent the program (chromosome) as a tree-based data structure in contrast to the linear representation proposed here [119], called Basic OOGP that uses a uniform genetic operator. However, the linear representations are not new either, linear GP has a well-developed, if recent, history [120] within mainstream GP. In the initial implementations of OOGP such as [116], traditional crossover and mutation operators are applied with variables as terminals being the principal addition. This OOGP implementation also uses the Java language reflection feature to automatically find the classes and methods in an existing library. Figure 1 shows an example of the tree representation and the program that it represents used in the initial implementation of OOGP.

The Basic OOGP algorithm [116] used for evolution is a standard genetic algorithm; however, the details of the chromosome used, crossover and mutation operators are significantly different. Specifically, each chromosome is a linear array of genes. Each gene in the array has three object pointers: one points to a type A object, called host; one points to a type B object called action, and the other points to a type A object called passive. When interpreted by the Basic OOGP engine it means that the object host will take action on the object passive. As for the genetic operators, the crossover operator implemented here is a uniform crossover implemented at the gene level, which randomly picks a percentage of genes from one parent and exchanges them with the equivalent positions in the second parent. The mutation operator randomly picks two gene positions and generates new randomly generated genes.

The Basic OOGP evaluation engine can then be specified as Figure 3.4. Furthermore, the OOGP has the potential to converge to the result faster than normal GP, even with smaller population sizes and fewer generations, though the results may vary

depending upon the selection, crossover, and mutation mechanisms. The experiments show that incorporating OO concepts into GP is a promising direction to improve GP performance [119].

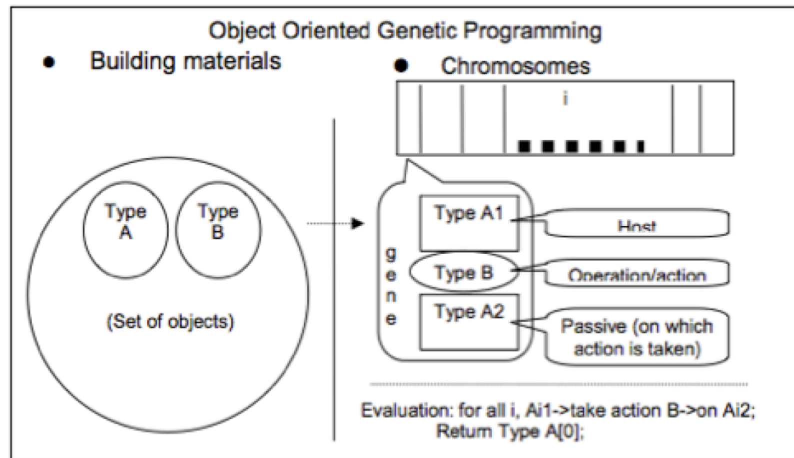

FIGURE 3.4: Conceptual Basic OOGP representation [119]

Hence, this idea of OOGP is quite intriguing, such concepts may also be easier to grasp, implement, and control complex software engineering problems. However, these OOGP implementations are preliminary and have not been directly applied to real-world problems. In this regard, this dissertation introduces a new form of GP that deals with a real-world problem within the domain of fault tolerance in distributed systems. This type of GP operates on DAGs rather than tree-structures (or bit-string formats) and store the chromosomes in the form of a class object, which would be a genotype format for the execution of new kinds of genetic operators on them.

3.2.3 General object-oriented GP

This type of GP operates on DAGs rather than tree-structures (or bit-string formats) and store the chromosomes in the form of a class object. Most of the other object-oriented implementations are Java-based implementations, though this dissertation also uses Java; however, the implementation, in this case, is independent of the programming language and therefore, flexible enough to be implemented in any programming language. Therefore, we call it General object-oriented genetic programming (GOOGP). In the proposed GOOGP, the replication strategies are converted into their appropriate genotype and phenotype representations. As for the genotype representations, the replication strategies as DAGs are stored in the hierarchical form of a class object for manipulations. Conventionally, an object is a member or an "instance" of a class being comprised of the state and the related behavior, state is the data (variables in a programming language) while the behavior is manifested through the methods (functions in a programming language). Methods operate on an object's state and serve as the primary mechanism for object-to-object communication. Hiding the internal state bounds all the interaction to be carried out through an object's methods, which is known as data encapsulation - a fundamental principle of the object-oriented paradigm.

Similarly, in our case, Figure 3.5 depicts the (genotype) class-object representation of a DRS that comprises variables, properties, and behaviors. Interactions are performed

by the methods that work on the internal state of the object. Each node is comprised of a unique node name to identify it uniquely, a number of allocated votes, weightage of each node, priorities w.r.t. the paths, and a list of its children. Each element of the list (carrying children nodes) itself is a node carrying its children nodes, too. Such hierarchical storage is very easy to make alterations with as well as traversing it. Such chromosomes (genotypes) are then traversed recursively to derive the respective quorums.

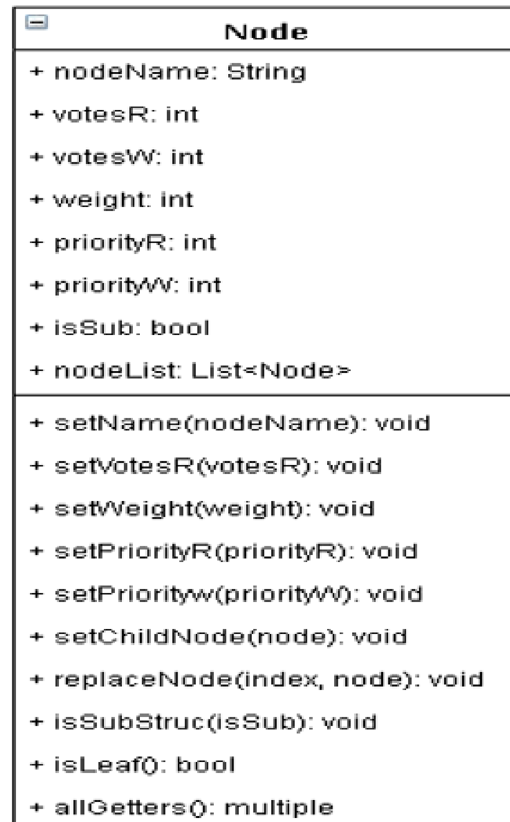

FIGURE 3.5: Genotype representation of strategies

Figure 3.6 shows a complex (computer-generated) example of a voting structure (explained in Section 2.4.1), which is a phenotype that models a TLP-like strategy consisting of six replicas. A manual example of this strategy can be found in [20]. These strategies as an initial population are stored in a scaleable database repository in the form of JSON documents as shown in Figure 3.7, which can easily be queried over any specified criteria. Such a phenotype is easy for visualization purposes, for which different graph visualization libraries are used for better visualization. Any quorum-based replication strategy can be modeled to represent the respective quorums in the form of such genotype and phenotype representations, which are flexible enough with an aim to easily combine them with others through GOOGP afterward. As the genotypes are altered, new strategies are generated as a result, which subsequently are converted into the phenotypes and stored in the database repository. The algorithm for this modeling real-world problems will be discussed in the next chapter, and here the next, this GP is explained in the context of data replication.

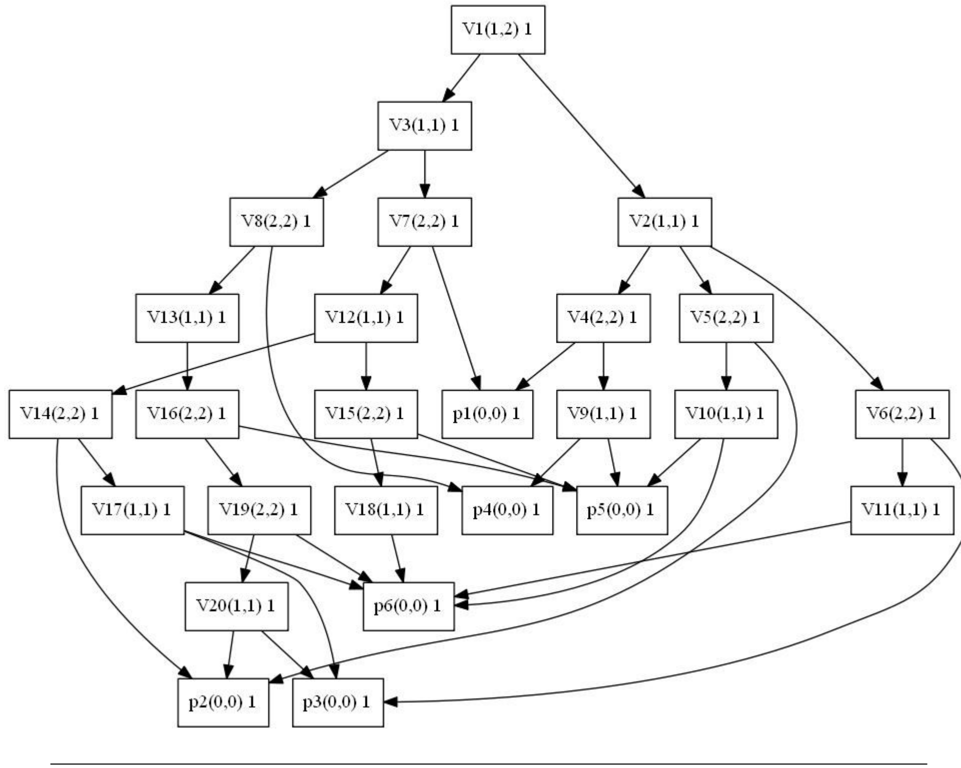

FIGURE 3.6: Phenotype modeling of a TLP-like strategy as a voting structure

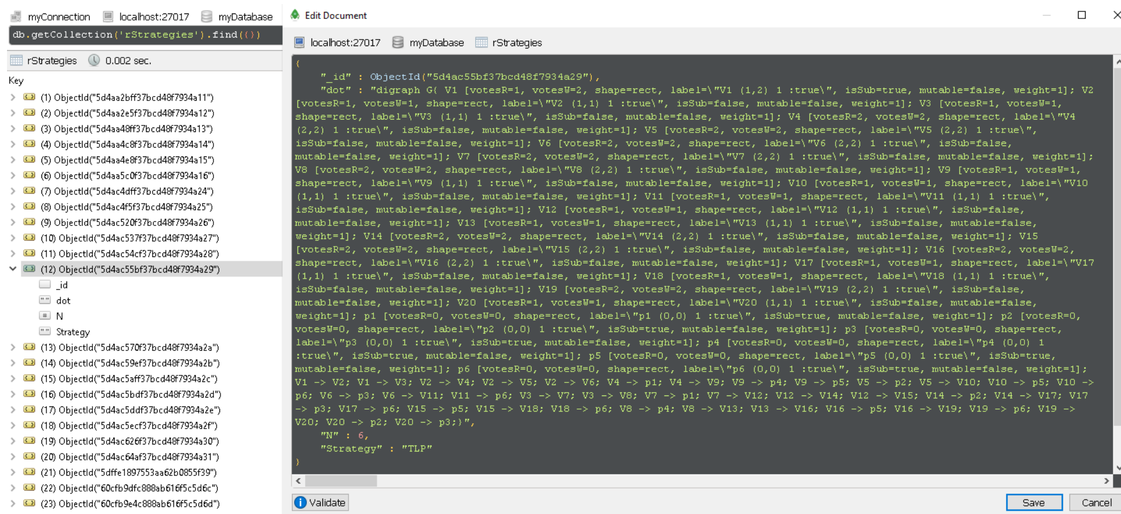

FIGURE 3.7: Phenotype - DRSs as JSON documents

3.3 Genetic programming in the context of replication

EAs such as GP are used to solve optimization problems. In the context of our problem, the term "optimization" means designing such a solution DRSs, which gives the values of all objective functions acceptable to the decision-maker. The most common optimization problems include constrained optimization [121], multi-modal optimization [122], combinatorial optimization [123], multi-objective optimization [124], etc. Constrained

optimization problems consider the problem of optimizing an objective function subject to constraints on the variables. Multi-modal optimization deals with optimization tasks that involve finding all or most of the multiple (at least locally optimal) solutions of a problem, as opposed to a single best solution. Combinatorial optimization is the process of searching for the maxima (or minima) of an objective function whose domain is a discrete but large configuration space. Typical problems are the traveling salesman problem, the minimum spanning tree problem, and the knapsack problem. Multi-objective optimization considers optimization problems involving more than one objective function to be optimized simultaneously. The problem being addressed in this dissertation lies within the realm of multi-objective optimization where the solutions are DAG-based voting structures, as explained earlier.

In the context of this multi-objective problem, there are three possibilities: 1) minimizing all the objectives 2) maximizing all the objectives 3) minimizing some objectives while maximizing others. In our case, it lies with the third option where, for instance, the cost and number of replicas need to be minimized while the availabilities need to be maximized. The availabilities of read and write operations are point-symmetric (for optimized strategies) to each other [8], which means that an increase in one results in a decrease in the other operation's availability. The cost of read and write operations is also conflicting. Likewise, the relation between the total availability (sum of access operations availability) and total cost (sum of access operations cost) is not that straight either. Furthermore, some objectives, i.e., availabilities are values between [0,1] while some objectives could be very large in value, i.e., cost of operations. In certain cases, some objectives are more important than others. Keeping in mind all these aspects, the goal is to increase the total availability of the access operations and decrease the total cost simultaneously, while, at the same time, restricting total replicas to a minimum number.

In this regard, the proposed GOOGP approach works on evolutionary concepts to combine DRSs intelligently and evolve them to eventually meet the specified criteria. Conventionally GP constitutes an encoding scheme, random crossover, mutation, a fitness function, and multiple generations of evolution to solve the specified task on its termination condition. The encoding scheme consists of a genotype (coding space) carrying an underlying set of traits and a phenotype (solution space), which is the behavioral expression of this genotype in a specific environment. Hence, the question arises, which encoding scheme should be used since poor representations may lead to poor results. The encoding scheme in our case is DAG-based voting structures. The crossover [125] is mixing up of genetic material of two existing DRSs to create new offspring solutions. It splits up the genome of two existing solutions at an arbitrary point and swaps them to create the offspring solutions inheriting properties from both of the parent solutions. The mutation operator generally changes the solution randomly but slightly, i.e., by flipping one or more bits from the previous offspring to generate a new altered child solution. The fitness function is to evaluate a DRS w.r.t. all the concerned objectives to meet the desired criteria. The DRSs are designed and optimized over several generations of evolution and presented at run-time, overtly displaying their trade-offs to choose the most suitable non-dominated strategies (that are not dominated by any other solution, which does not necessarily mean they are better in all the objectives) meeting the demands, with acceptable constraints.

In the pursuit of finding a better replication strategy, the questions of crossover and mutation types, as well as points (locations to execute mutation/crossovers), are also

important. Moreover, the population size also matters because a very small size implies few possibilities of executing the crossovers. Therefore, only a fraction of the search space can be explored. Alternatively, a very large size may slow down the genetic approach. Although it is highly problem-specific but very large populations do not solve the problem faster than moderate-sized populations. Figure 3.8 illustrates the problem in the context of genetic programming. It begins with a specified scenario, for which an initial population of solution DRSs is generated. The initial population is analyzed based on its fitness w.r.t. the scenario to check whether the existing solutions are good enough to solve the defined problem. If the criteria are not met, it selects the best solutions among others to perform crossover and mutation (with their defined probabilities) to generate new populations of offspring solutions and evaluate them again. This goes in cycles in anticipation of a constant evolutionary trajectory until an appropriate solution satisfying the specified criteria is identified. The next chapter explains the adopted fault model, the proposed methodology, scenario parameters, fitness function, crossover, and mutation operators, in detail.

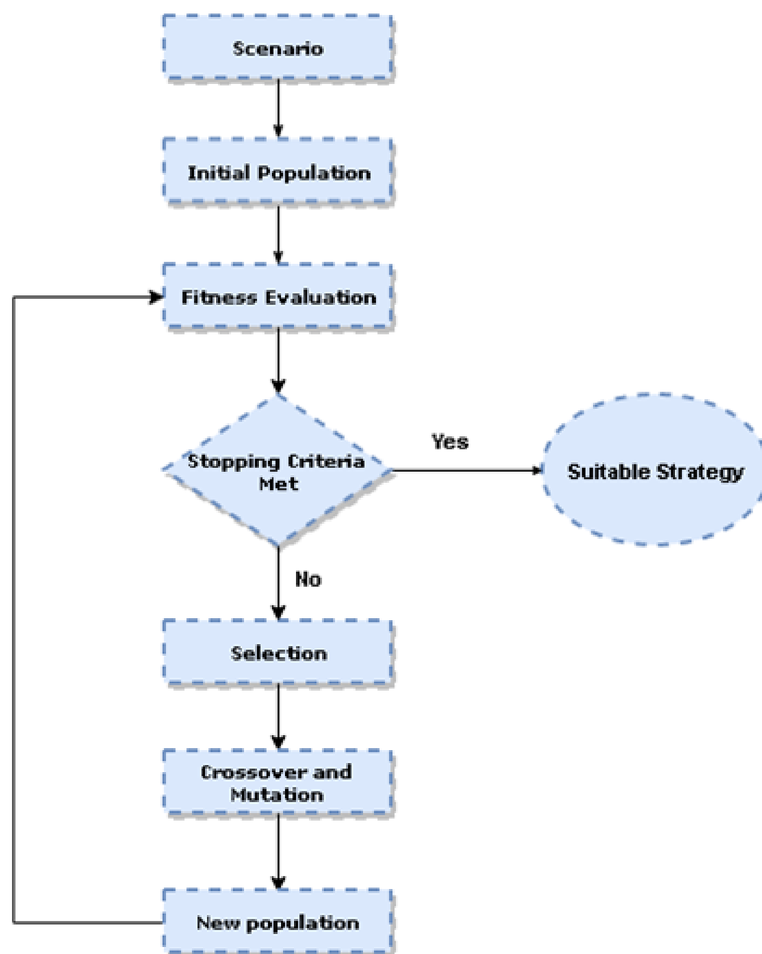

FIGURE 3.8: Genetic programming

3.4 Summary

This chapter briefly discusses machine learning approaches, i.e., supervised, unsupervised, reinforcement, and evolutionary algorithms. Particularly, it focuses on genetic programming and its types, subsequently, discusses it in the context of replication strategies. The genetic approach works on the evolutionary concepts to combine DRSs intelligently and evolve them to eventually meet the specified criteria. It constitutes an encoding scheme, random crossover, mutation, a fitness function, and multiple generations of evolution o solve the specified task on its termination condition. The encoding scheme consists of a genotype (coding space) carrying an underlying set of traits and a phenotype (solution space), which is the behavioral expression of this genotype in a specific environment.

In this regard, the chapter introduces a new form of GP that deals with a real-world problem within the domain of fault tolerance in distributed systems. It proposes General object-oriented genetic programming as a basic approach to resolve the mentioned multi-objective problem. Most of the other object-oriented implementations are Java-based implementations, though this dissertation also uses Java; however, the implementation is independent of the programming language and therefore, flexible enough to be implemented in any programming language. Therefore, we call it General object-oriented genetic programming (GOOGP). This type of GP operates on DAGs rather than tree-structures (or bit-string formats) and store the chromosomes in the form of a class object, which would be a genotype format for the execution of new kinds of genetic operators on them. The encoding scheme in our case is DAG-based voting structures. Encoding schemes are also of utmost importance since poor representations may lead to poor results.

In the proposed GOOGP, the replication strategies are converted into their appropriate genotype and phenotype representations. As for the genotype representations, the replication strategies as DAGs are stored in the hierarchical form of a class object for manipulations. This class-object representation of a DRS comprises variables, properties, and behaviors. Interactions are performed by the methods that work on the internal state of the object. Each node is comprised of a unique node name to identify it uniquely, a number of allocated votes, weightage of each node, priorities w.r.t. the paths, and a list of its children. Each element of the list carrying children nodes itself is a node carrying its children nodes, too. Such hierarchical storage is very easy to make alterations with as well as traversing it. Such chromosomes (genotypes) are then traversed recursively to derive the respective quorums. This idea of GOOGP is quite intriguing, such concepts may also be easier to grasp, implement, and control complex software engineering problems.

Chapter 4

Novel framework to design replication strategies

This chapter combines the concepts of replication with genetic programming. Since DRSs are computer programs, which need to be optimized w.r.t. the problem; therefore, the proposed GOOGP is used to evolve the DRSs as computer programs to eventually control replicated objects. In this regard, the strategies are transformed into the described unified genotype and phenotype representations to subsequently apply genetic programming concepts to them. Through the proposed framework, the concepts of replication can be easily combined with the GOOGP. The framework includes a fault model, constraints of a scenario to be met, GP concepts, and relevant genetic operators, which will be discussed here in this section.

4.1 Adopted fault model

Prior to discussing the basic methodology, the fault model and other assumptions are stated first. The access operations are either read or write and are performed only when the proper quorum is acquired. A quorum is a set of replicas chosen to execute access operations. The replicas are supposed to manifest a fail-silent behavior. All failures are assumed to be independent of each other. The network is supposed to be fully connected without communication failures. Only nodes (machines) hosting replicas can fail and the probability that a node has failed at any particular point in time is (1-p). p gives the probability that a node is available at an arbitrary point in time. Also, this fully connected behavior with no communication failures is not necessary for correctness purposes but rather for analysis purposes, which means with such assumptions, it is easier to carry out the experiments and analysis. The strategies are supposed to be version-based to avoid additional time synchronization issues, i.e., a replica does not only consist of some "payload" data but also a version number. A replica with the highest version number has an up-to-date payload.

4.2 System architecture

Figure 4.1 shows an abstract representation (for understanding) of the proposed methodology utilized to identify and design optimized DRSs. Simplistically, it starts from a scenario to be fulfilled and a set of state-of-the-art replication strategies. The

scenario will be explained in detail in this chapter. The replication strategies are converted into a unified representation of voting structures (representing each a computer program) and stored in a scalable database repository. These voting structures (as unified representations have been explained in the Section 2.4.1) forming quorum systems eliminate the diversity between the replication strategies since the same quorums would be derived recursively here, as it would be in an orthodox representation, therefore, this representation is immensely powerful and the key to the proposed hybrid approach. The experiments and the analysis (shown later here and in the next chapter) are performed on the repository until the desired solution is met which then is inserted back to the repository for future use. As for the selection of appropriate techniques for the identification and the design of optimized data replication strategies, machine learning, mainly GOOGP, is then applied to the repository to search or design appropriate solutions and optimize them accordingly afterwards.

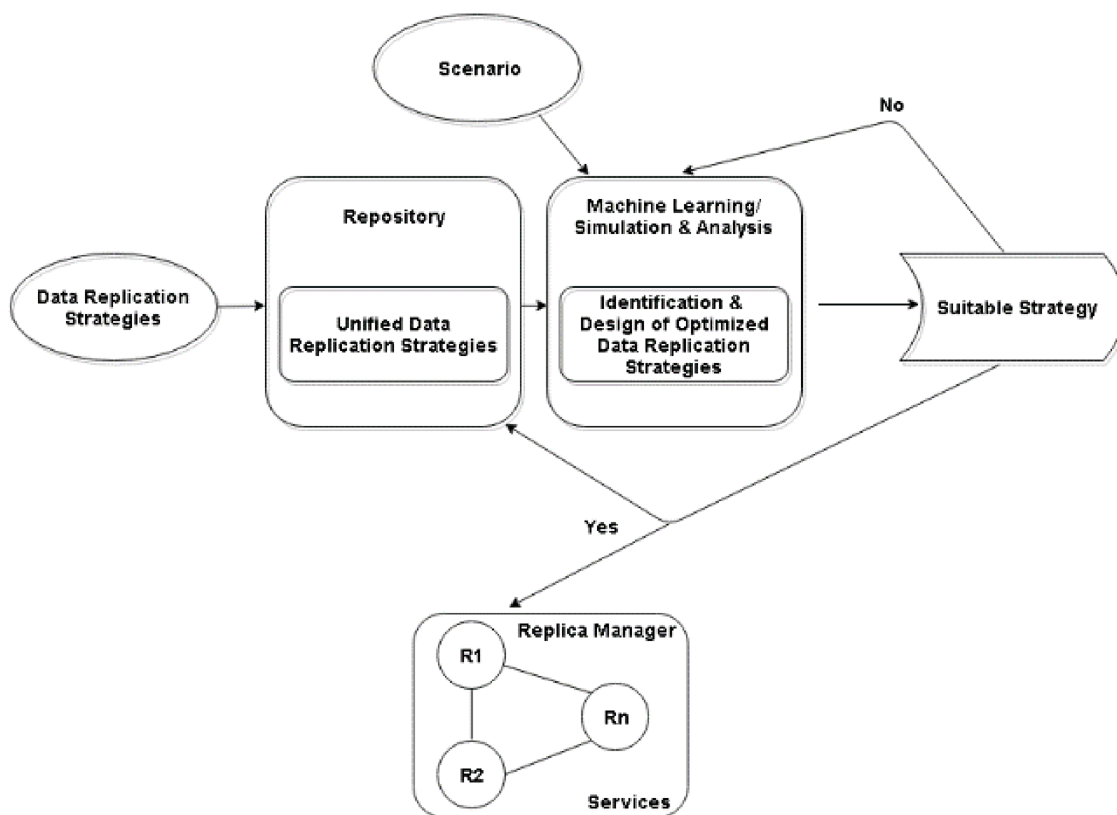

FIGURE 4.1: Methodology

4.3 Specification of a constraints-based scenario

A scenario for DRSs consists of constraints that determine the fitness of a strategy holistically to judge the appropriateness of a solution. These constraints have different thresholds for the replication strategies to adhere to. It could be dependent on the application, its requirements, and resources. Scenarios reflect objectives, which are supposed to be optimized for the input values. The semantics of a scenario is discussed next.

4.3.1 Consistency of operations

DRSs ranging from strict data consistency to relatively weaker notions. In this research, the consistency is 1SR which provides high consistency being maintained by the intersection property between every read (write) and write (write) operation of a DRS. Also, replicas are locked exclusively for the write operations and locked shared for the read operations. This quorum intersection property is used by the DRSs to meet the 1SR property since a single write operation can write all replicas of its WQ, similarly, one or more read operations can read their RQs. 1SR property must be maintained throughout the genetic process, otherwise, the solution becomes invalid and of no use.

4.3.2 Number of replicas

The number of replicas n must be restricted to a threshold of ϵ depending on the resources, but in such a way that availability is not compromised much. In general, an increase in the number of replicas often increases the availability of access operations. This threshold is because it consumes resources to create new replicas nodes for hosting replicas, therefore, a strategy must yield results within this threshold.

$$n, \epsilon \in \mathbb{N}^{+} \wedge n \leq \epsilon \tag{4.1}$$

4.3.3 Availabilities of the access operations

This availability is a probability by which an access operation can be successfully performed by a DRS. The probability that the data access operations are available for a DRS depends on the characteristics of the strategy, the probability of individual replicas p, and the number of replicas n. It is defined by A_r(p, n) and A_w(p, n) respectively, where A_r(p, n), A_w(p, n) $\in$ [0,1]. For some DRSs, there exist closed formulas to calculate the availability as well as the costs. However, generally, the Eqs. (4.2) and (4.3) are used to analyze the data access operations' availability of a DRS. All the RQs and WQs are derived from a DRS to calculate A_r(p, n) and A_w(p, n) for given p and n values. The Equations calculate the read and write operation availabilities respectively.

For the given instance (Figure 2.5), the closed read quorum set (RQS) and closed write quorum set (WQS) are super-sets of all the RQs w.r.t. to the full set of replicas RQs and WQs, respectively.

For instance, the closed read quorum set RQS of RQ is:
RQS = $\{\{p1\}, \{p2, p3\}, \{p2, p4\}, \{p3, p4\}, \{p1, p2\}, \{p1, p3\}, \{p1, p4\}, \{p1, p2, p3\}, \{p1, p2, p4\}, \{p1, p3, p4\}, \{p2, p3, p4\}, \{p1, p2, p3, p4\}\}$

(See [8] for details.) The availability of the access operations for a DRS, generally, is calculated by summing up the probabilities of all the elements existing in RQS (WQS) on a given value of p.

$$A_r(p,n) = \sum_{\forall q \in RQS} p^{|q|}(1-p)^{n-|q|} \tag{4.2}$$

$$A_w(p,n) = \sum_{\forall q \in WQS} p^{|q|}(1-p)^{n-|q|} \tag{4.3}$$

The availabilities of read and write operations must be within a threshold α and β, respectively.

$$A_r, A_w, \alpha, \beta \in [0,1] \quad \wedge A_r \geq \alpha \quad \wedge A_w \geq \beta \tag{4.4}$$

4.3.4 Costs of the access operations

As a cost notion, the average minimal cost is used for a read or a write operation being represented by C_r(p, n) andC_w(p, n) respectively. It is calculated by summing up the minimal operation cost minRQ (minWQ) obligatory to form a read (write) quorum for every replica set present in RQS (WQS), with the probability of the replica set appearing. Finally, the resulting values are divided by the respective operation's availability A_r(p, n) or A_w(p, n). In the context of the given example (Figure 2.5), i.e., $minRQ$ ({p1,p2,p3}) is $|\{p1\}| = 1, minRQ(\{p2, p4\})\ is\ |\{p2, p4\}| = 2, and\ minWQ(\{p1, p2, p3, p4\})\ is\ |\{p1, p2, p3\}| = 3$.

$$C_r(p,n) = \frac{\sum_{\forall q \in RQS} \; p^{|q|}(1-p)^{n-|q|} * minRQ(q)}{A_r(p,n)} \tag{4.5}$$

$$C_w(p,n) = \frac{\sum_{\forall q \in WQS} \; p^{|q|}(1-p)^{n-|q|} * minWQ(q)}{A_w(p,n)} \tag{4.6}$$

The cost of read and write operations has to be within a threshold γ and δ, respectively.

$$C_r, C_w, \gamma, \delta \in \mathbb{R}^+ \quad \wedge C_r \leq \gamma \quad \wedge C_w \leq \delta \tag{4.7}$$

4.3.5 Fitness weightage

It is a so-called fitness weightage (fw) given to any of the concerned objectives to set its importance in the identification, designing, and optimizing the prospective solutions accordingly. It is a value between [0,1] for tilting the fitness value towards certain objectives, which by default would remain neutral. This helps in converting a multi-objective into a single objective problem (through a fitness function), which makes the optimization problem somewhat easier to solve.

$$fw \in [0,1] \tag{4.8}$$

4.3.6 Probability of the individual replicas

There is a subtle difference between the availability of the access operations and the availability of individual replicas p. The user performs the operations with access operations' probability while the probability p is the availability of a node hosting a replica and (1-p) indicates the probability by which a replica may fail at any point in time. In a scenario, p is restricted to be in the interval between $p_{min} \leq p \leq p_{max}$.

$$p_{min}, p, p_{max} \in [0,1] \quad \wedge p_{min} \leq p \leq p_{max} \tag{4.9}$$

4.4 Manual designs of voting structures by modeling the state-of-the-art strategies

In this section, the hybrid approach is practically applied to cutting-edge DRSs by exploiting the concept of voting structures. MCS is the most superior strategy in terms of its operation availabilities where the protocol freely chooses any of the replicas to form the quorum by the majority. TLP fairly competes with MCS in terms of availability of the access operations but gives better cost by slightly compromising its availability. There are many prospective possibilities of combining these DRSs to inherit the qualities from both to some extent.

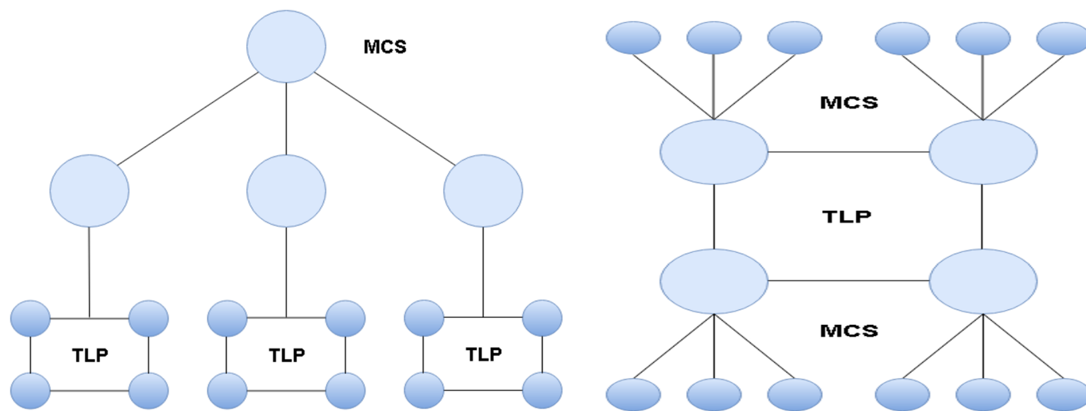

FIGURE 4.2: Hybrid DRSs of MCS & TLP

Simplistically, it could be combined, either way, MCS on top of TLP or vice versa as shown in Figure 4.2. Figure 4.3 presents a hybrid DRS with MCS on top of TLP. This MCS imposes a logical structure over TLP and then, TLP is applied to the physical replicas through MCS. Any two of the child substructures of the root node can be selected to form a read or write quorum. Hence, respective quorums for the access operations can easily be derived by traversing this structure recursively.

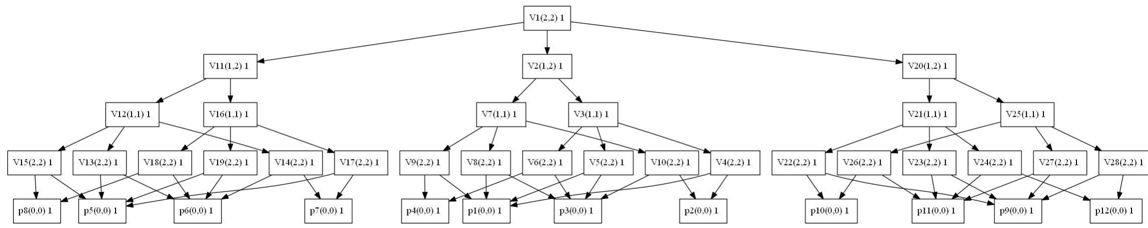

FIGURE 4.3: Voting structure: MCS on top of TLP

Figure 4.4 represents the hybrid DRSs in the form of a voting structure with TLP on top of MCS. MCS with three replicas is attached to every physical node of TLP to obligate the system to consist of a total of 12 replicas. The leaf nodes, hence, are all physical replicas while the rest of them are virtual replicas representing groupings of virtual and actual replicas.

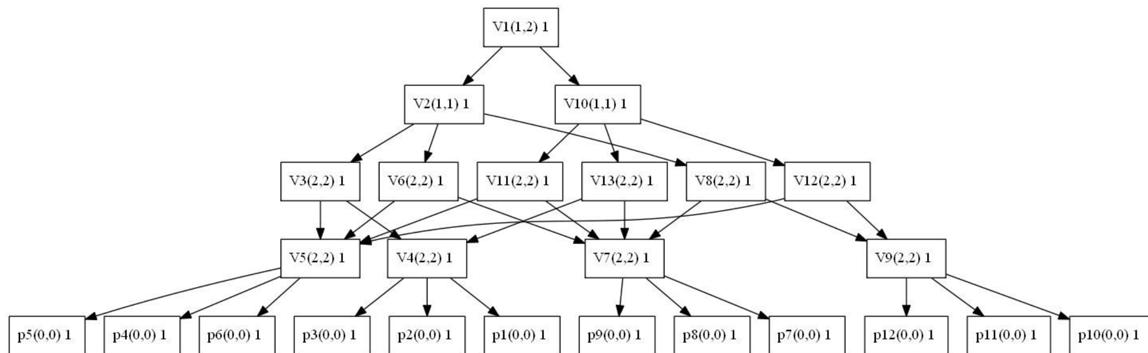

FIGURE 4.4: Voting Structure: TLP on the top of MCS

Figure 4.5 illustrates the availability comparison of the two above-given hybrid strategies in their respective order (Figure 4.3 as Strategy 1 and Figure 4.4 as Strategy 2). These availabilities are being calculated by the Eqs. 4.2, 4.3. The x-axis represents the availability of replicas and the y-axis denotes the availability of access operations. The availability is calculated by adding up probabilities of all the possible cases of the quorums [46]. It can be seen that they exhibit different properties and availabilities. The latter strategy, which has MCS at the bottom, has better write availability than the former one. This improved write availability is at the expense of read availability, but for the later values of p, the second strategy seems to be better for the operations holistically as it is comparatively harder to increase the write availability.

Figure 4.6 displays the cost comparison of the two above-mentioned hybrid strategies in their respective order (Figure 4.3 as Strategy 1 and Figure 4.4 as Strategy 2). The cost is being calculated by adding up the probabilities of the respective cases multiplied by their minimal quorums. Additionally, the resulting value is divided by the respective access operation's availability [46]. The operations have a quite economical cost, the values differ in the middle, and later on, converge onto the same values of four replicas each for the best cases. The formulas for the cost calculations are given in Eqs. 4.5, 4.6.

Similarly, Figure 4.7 is an endeavor to combine TLP and GP (which can also be combined in several ways). The given instance focuses on the possibilities of either GP being on top of TLP or TLP being on top of GP. GP has a better read availability while

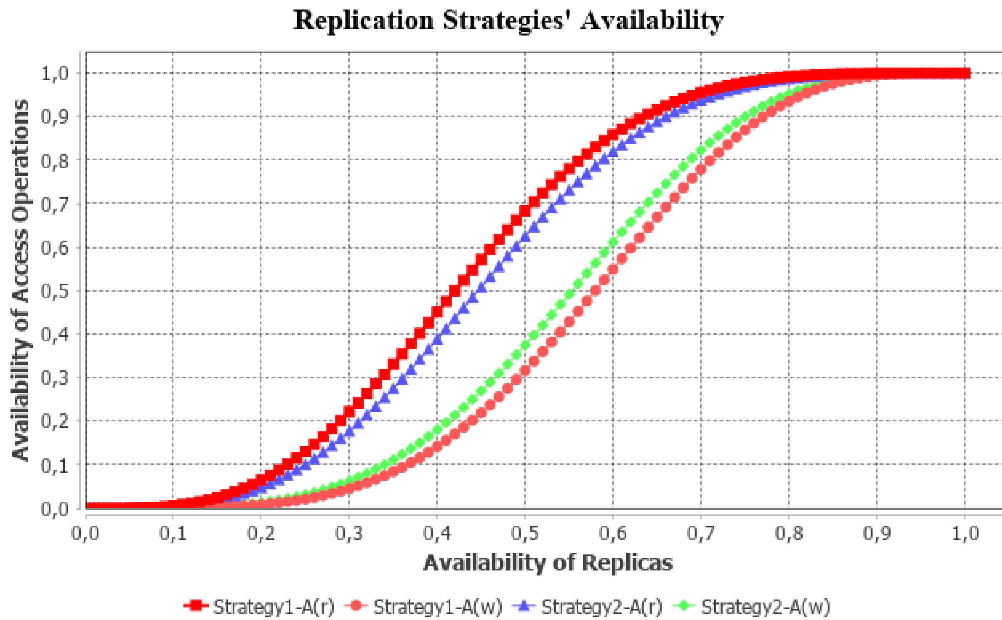

FIGURE 4.5: Availability of hybrid DRS

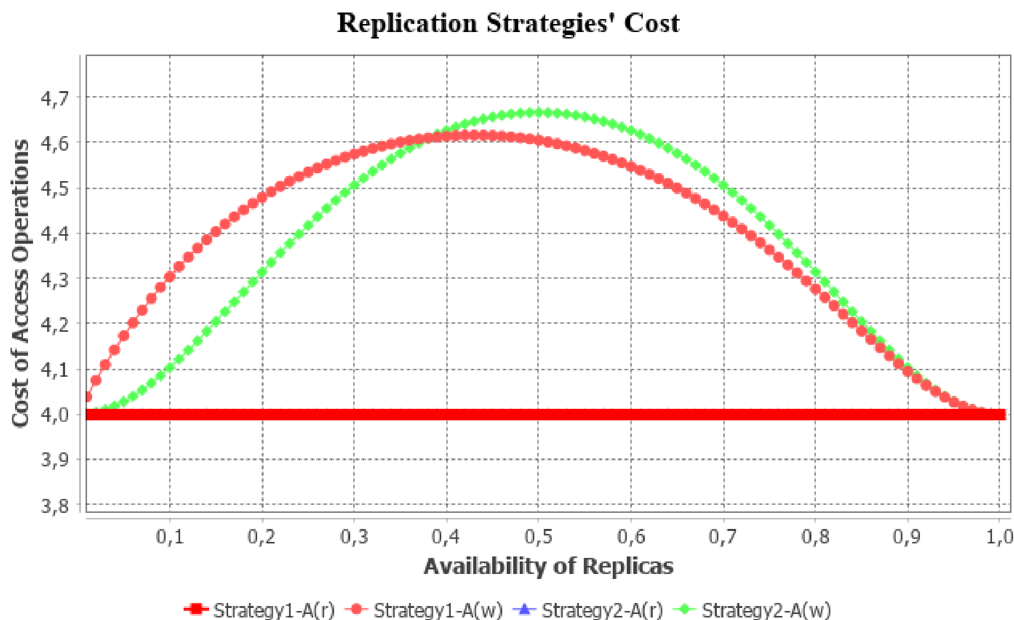

FIGURE 4.6: Cost of hybrid DRS

TLP has an edge over write availability and cost values, particularly for the best cases.

Figure 4.8 depicts the above-mentioned hybrid DRSs in the form of a voting structure with GP on top of TLP. The structure itself is rather complex and, unfortunately, too large to fit in well but it can be noticed that it comprises 16 actual replicas (leaf nodes) and other virtual replicas to support the quorum mapping of the protocols.

Figure 4.9 represents the mentioned hybrid DRS (see Figure 4.8, right DRS) in the form of a voting structure with TLP on top of GP. This type of hybrid approach results in a

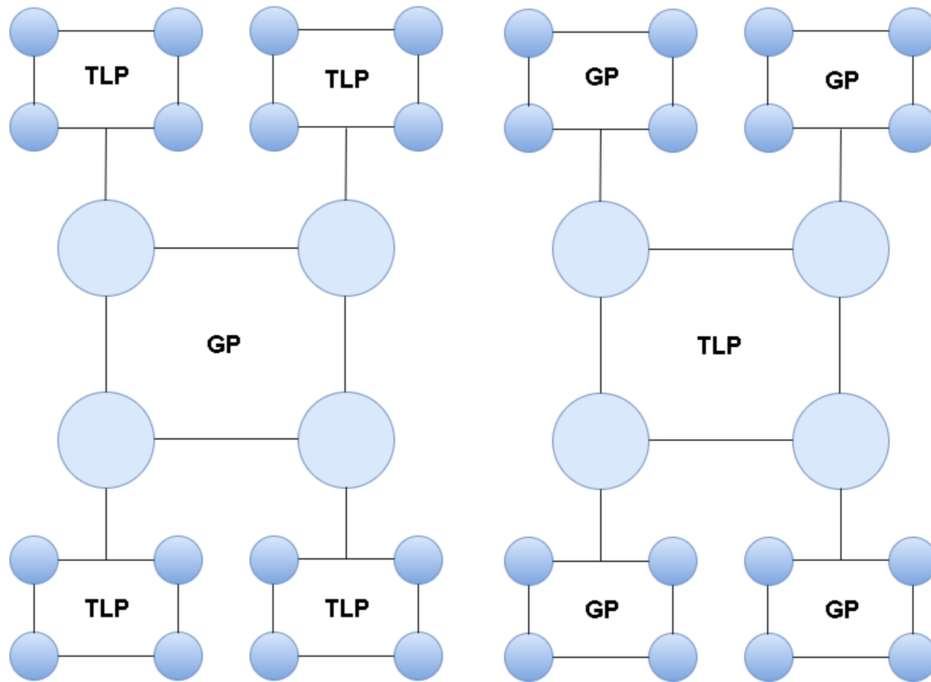

FIGURE 4.7: Hybrid DRSs of GP and TLP

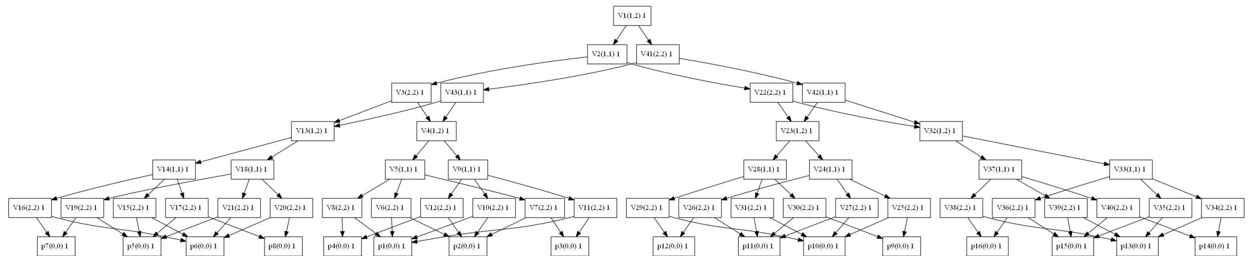

FIGURE 4.8: Voting Structure: GP on the top of TLP

huge increase of read availability and outclasses MCS, both, in read availability and cost of the access operations (as MCS is costly in general) but, unfortunately, compromises the write availability.

As for the other example, Figure 4.10 shows the availability comparison between the hybrid approach (MCS on the bottom of TLP, Figure 4.4) and a flat MCS of 12 replicas. Strategy 1 represents the MCS while Strategy 2 represents the hybrid one. The red and pink lines indicate the read and write availabilities of flat MCS, respectively, whereas the blue and green lines depict the operation availabilities of the hybrid strategy, respectively. It can be seen that the operation availabilities for both these strategies are very close and converge to basically the same values for the later values of p, which is good enough considering quite reliable hardware of today.

As shown in Figure 4.11, in terms of its cost, the hybrid DRS is far cheaper than the flat MCS while it is overt, too, that availability is not much compromised either. The blue and green lines represent the read and write availabilities of the hybrid strategy, respectively. For the best case, it takes merely four replicas to perform a read or a write

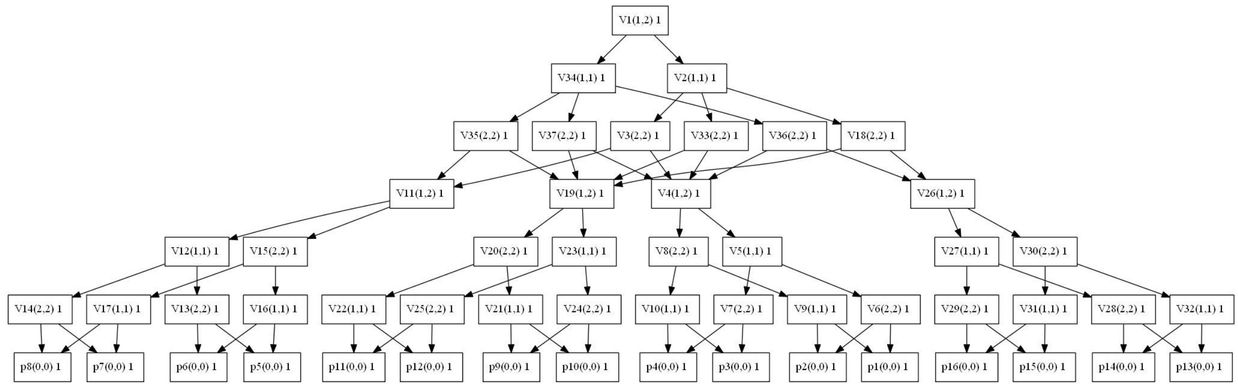

FIGURE 4.9: Voting Structure: TLP on the top of GP

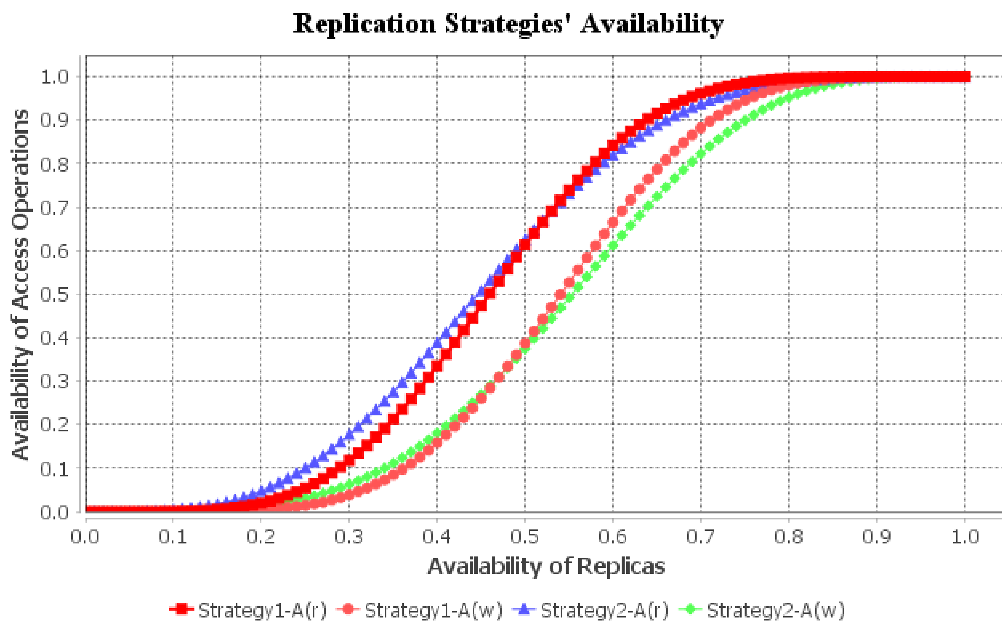

FIGURE 4.10: Availability: MCS vs. hybrid DRS

operation while the flat MCS takes a constant cost of 13 replicas in total to perform both access operations. The goal here is to achieve low operational costs but at the same time not sacrificing too much of the availabilities. Here, the costs of the access operations have been significantly decreased while not much compromising on the availabilities.

As described, the presentation of quorum protocols as voting structures allows replication strategies to be easily merged with the other strategies in many possible ways. The resulting new DRSs can then be used to fulfill the requirements of scenarios that would not have been that easily possible by homogeneous strategies considering the quality metrics of operation cost, operation availabilities, while still guaranteeing 1SR and a threshold of a certain number of replicas. Next, the framework to automatically design such replication strategies is explained with the help of genetic programming.

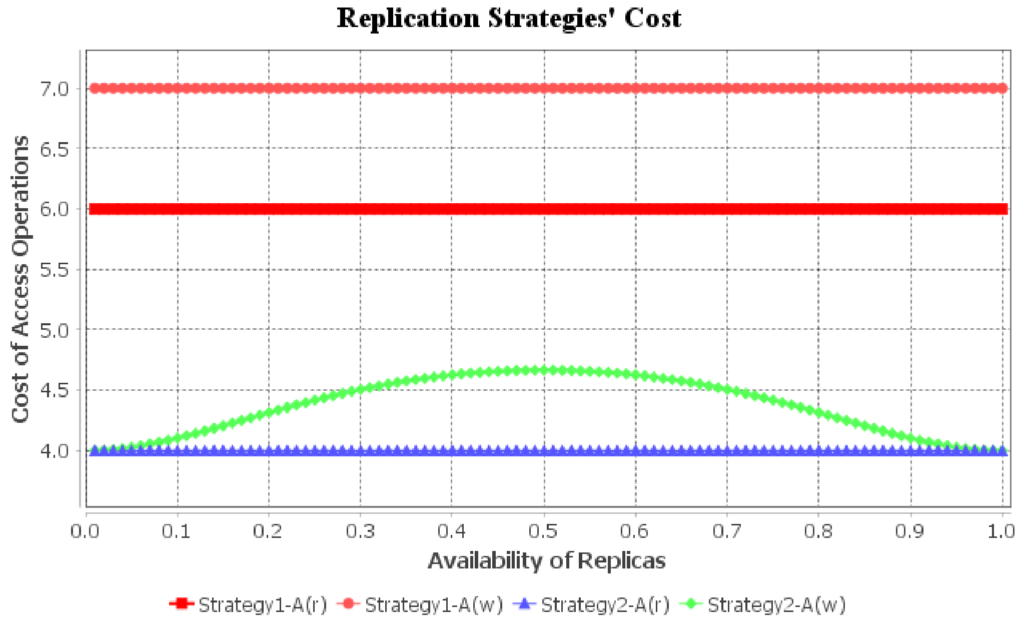

FIGURE 4.11: Cost: MCS vs. hybrid DRS

4.5 Customized genetic programming algorithm

Having discussed the methodology, terminologies, and semantics, this section discusses the implementation aspects of the proposed approach comprising the parameters, functions, the respective crossover (mutation) operators, and the GOOGP algorithm itself, in detail. Once the scenario is specified to find a suitable DRS to fulfill it, the system parameters are set for the algorithm to run. The proposed system in this dissertation is implemented in JAVA, which is feasible for large applications and has better cross-platform support. When a scenario is defined to accordingly find an optimized DRS, the system parameters are set and passed on to the algorithm to run. The basic mechanism is similar to what standard multi-objective optimization algorithms like NSGA-II operate. The selection mechanism is based on the trade-off metric being calculated using fitness values, which is a preference-based multi-objective optimization approach though weighted sum [43]. Regarding the hyper-parameters settings, it is done by experimenting with different values and finding better results. However, usually, a crossover is done with a high probability, and mutation is done with a small probability as in our case, the crossover is performed all the time while mutation with a small probability.

The Algorithm 1 implements the proposed GP approach where initially a scenario is defined based on its objectives. These objectives are evaluated by the fitness function to calculate the expected single-valued fitness for the DRSs to achieve. So, the defined scenario is evaluated by the fitness function, and scenarioFitness to be achieved is calculated as a result. The initial population of voting structures is generated and stored in a database repository. μ and λ are defined, along with crossover and mutation probabilities. The list List contains parent DRSs, the list λList comprises offspring DRSs, whereas the list initPopList consists of an initial population of DRSs. There are, further, intra-crossover and intra-mutation probabilities that are set to use the genetic operators, accordingly. Moreover, initPopList is also being used in every generation

(with a probability of initPopListProb). The Boolean variable isFit determines whether a strategy has achieved the expected level of fitness.

The genetic program loops through all the passed on DRSs, calculates the fitness of every individual strategy, and selects the μ best strategies to the List, in case, there is no satisfactory solution found in the initial population. This μList is then sent to the while loop, where it randomly mates two DRSs either from μList or from the initPopList (with defined probabilities) to form offspring DRSs. Such a use of the initial population is for not letting the existing good solutions vanish away in the next generations. It creates λ number of new offspring strategies through crossovers and mutations. The λList constitutes λ number of newly created strategies, which are evaluated again to check if they satisfy the standard criteria.

As described, different types of crossovers are used, these crossovers are performed on the replication strategies with certain probabilities and also the intra-operators to execute them, accordingly, on the DRSs. These intra-crossover and intra-mutation probability distributions are for using some operators more than the others to accordingly find a better solution. New offspring strategies are produced as a result of these genetic operators. If the criteria are met, then the relevant newly generated optimized strategy is stored in the repository, the while loop terminates and so does the program. If not, it selects the best DRSs to the List from the elements of the (μList + λList) for the next generation to repeat the process of crossovers and mutations on the chosen DRSs of better fitness. This process continues until a suitable strategy is found.

Hence, the DRSs are optimized by every generation and better ones are picked. While replication strategies are being generated, these solutions can easily be plotted revealing their trade-offs overtly. The process becomes cyclic until a solution of desired expectation is found with an acceptable level of trade-offs between the concerned objectives. The chosen solution DRS is saved back to the database repository for future use.

4.6 Fitness function for the strategies' evaluations

As described, the objectives in the scenario are 1) conflicting in nature, 2) imbalanced in a way that values for some objectives are probability ranges while others are very large, 3) some of them must be maximized and some of them must be minimized. This section addresses these problems by developing a fitness function to transform this multi-objective problem into a single-objective problem for determining the quality of a solution through this single-valued metric. The algorithm takes the availabilities, costs, number of replicas, and fitness-weightage specified in the scenario as parameters. These values are calculated by the objective functions (formulas of which are given in Eqs. 4.2, 4.3, 4.5, and 4.6) and then passed on here. The weightage, as mentioned earlier, determines the importance of certain objectives over others in the desired solution. This weightage is multiplied by the respective availability and the cost values, but in the case of cost, it is multiplied by the number of replicas n of the desired strategy divided by its expected cost in order to normalize the imbalance between the availability and cost values as well as resolving the minimization (maximization) problem of these objectives. The calculation of the fitness function is shown in Algorithm 2. At line 4, the sum of both the values is returned as a single-valued fitness to examine the DRSs on this standard criterion. Now, a higher fitness value determines the appropriateness of a solution to the specified constraints.

Algorithm 1:

```
1  Specify a scenario;
2  Specify μ and λ;
3  Specify mutationProb;
4  Specify intraMutationProbList;
5  Specify intraCrossoverProbList;
6  Specify initPopListProb;
7  Define rand;
8  Initialize initPopList;
9  Initialize μList;
10 Initialize λList;
11 Double scenarioFitness = 0.0;
12 Boolean isFit = false;
13 Generate initial population of DRSs to the repository;
14 Retrieve, parse & store the generated DRSs to initPopList;
15 geneticProgrammingFunc () {
16   scenarioFitness = calculateFitness(scenario);
17   Loop through initPopList
18     Calculate fitness;
19     if (fitness ≥ scenarioFitness) {
20       isFit = true;
21       return;
22     }
23   END
24   Choose μ best DRSs to the μList;
25   Do{
26     Empty λList;
27     Loop to λ
28       Select randomly DRS1 from μList;
29       Select randomly DRS2 from (μList || initPopList);
30       rand = rand (0,1);
31       if (rand ≤ intraCrossoverProbList.get(0)) {
32         Perform crossover1 of DRS1, DRS2;
33       }
34       if (rand ≤ intraCrossoverProbList.get(1)) {
35         Perform crossover2 of DRS1, DRS2;
36       }
37       elseif (rand≤intraCrossoverProbList.get(2)) {
38         Perform crossover3 of DRS1, DRS2;
39       }
40       else {
41         Perform crossover4 of DRS1, DRS2;
42       }
43       Generate offspring DRSs;
44       if (rand(0,1) ≤ mutationProb) {
45         if (rand(0,1) ≤ intraMutationProbList.get(0)){
46           Perform mutation1 on the offspring;
47         }
48         else {
49           Perform mutation2 on the offspring;
50         }
51       }
52       Calculate fitness;
53       if (fitness ≥ scenarioFitness){
54         isFit = true;
55         Store offspring DRS into the repository;
56       }
57       Add offspring DRSs to the λList;
58     END
59     Select μ best DRSs to the μList from (μList + λList) for next generation;
60   }
61   While (! isFit);
62 }
```

Algorithm 2:

```
1 fitness (A_r, A_w, C_r, C_r, n, fw){
2     availFitness = (fw) * (A_r + A_w);
3     costFitness = (1.0 - fw) * (n / (C_r + C_w));
4     return (availFitness + costFitness);
5 }
```

4.7 Crossover operators for strategies

The approach uses multi-type crossovers to explore more possibilities of enhancing the fitness of DRSs since it equips the algorithm with more power as compared to single-type crossover. There are many ways in which the DRSs can be combined and the resulting strategy certainly exhibits different properties than its parents. The crossover randomly picks two existing DRSs, as well as their crossover points within the two selected strategies, to subsequently swap their nodes on chosen crossover points and create hybrid offspring DRSs (as shown in Figure 4.12), thereby inheriting mixed properties from both the parent solutions. The crossover point, in our case, must be valid so that it does not affect the 1SR consistency of offspring DRSs. For this, every node has a Boolean variable indicating valid points for crossovers, enabling crossovers to be executed only on those locations, thereby maintaining the DRSs' 1SR property throughout the genetic process. In addition, during the process, the algorithm limits the number of replicas not to grow beyond the specified threshold of ϵ since resources are limited. It also discards solutions not adhering to these properties.

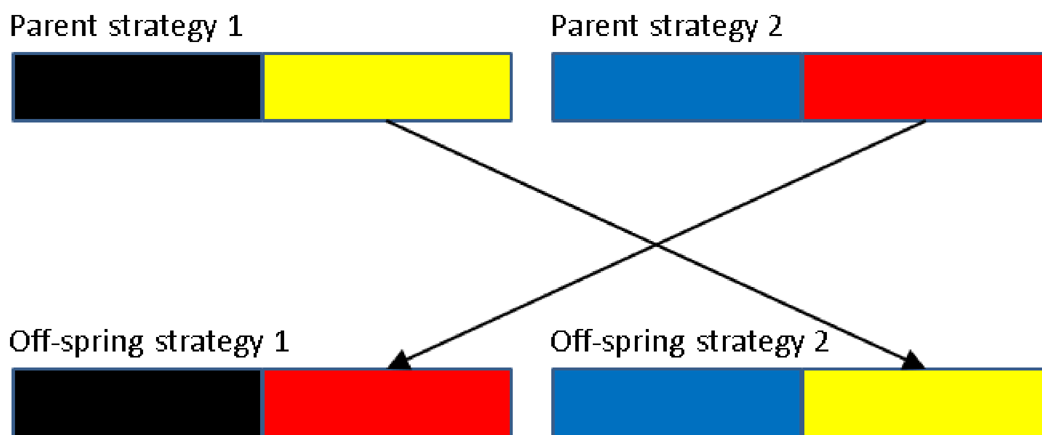

FIGURE 4.12: General crossover

Algorithm 3 represents the general algorithm of crossovers where initially it finds "valid" crossover points randomly, and splits the strategies on those points, swaps them, and returns consistent innovative offspring DRSs. A DeepCopy function makes the copies of the strategies before splitting them so that it does not affect the nature of the original strategies in the list. Although the algorithm constitutes and deals with 8-9 types of crossovers, for simplicity, it is categorized into four main categories:

Algorithm 3:

```
1 crossover (str1, str2) {
2     pointStr1 = findCrossoverPoint(str1);
3     pointStr2 = findCrossoverPoint(str2);
4     part1 = break (deepCopy (str1), pointStr1);
5     part2 = break (deepCopy (str2), pointStr2);
6     newStr1 = replace (str1, pointStr1, part2);
7     newStr2 = replace (str2, pointStr2, part1);
8     return newStr1 & newStr2;
9 }
```

4.7.1 Type 1 operator

The basic Type 1 crossover takes two complete DRSs and combines them horizontally by a new root node without breaking or reducing them to sub-strategies. In Algorithm 4, it creates a new node, sets its relevant properties, and adds the selected two complete strategies as children nodes of the newly created node. For instance, Figure 4.13 and 4.14 show the strategies to be combined by this operator. Having combined these two complete strategies horizontally through a new node, Figure 4.15 shows the resulted hybrid strategy.

Algorithm 4:

```
1  crossoverT1 (str1, str2) {
2      Node rootNodeStr = new Node();
3      rootNodeStr.setNodeName("V1");
4      rootNodeStr.setVotesR(1);
5      rootNodeStr.setVotesW(2);
6      rootNodeStr.setIsSubStrs(true);
7      rootNodeStr.setMutable(true);
8      rootNodeStr.setWeight((int)1);
9      rootNodeStr.setChildNode(str1);
10     rootNodeStr.setChildNode(str2);
11     changeNames(rootNodeStr);
12     return rootNodeStr;
13 }
```

4.7.2 Type 2 operator

Type 2 crossover takes two DRSs and having broken them into smaller sub-strategies, either 1) it could combine two sub-strategies horizontally by a new root Node or 2) go on with combining one sub and one complete strategy horizontally by a new root node. For instance, Figure 4.13 and 4.14 show the strategies to split, having split the DRSs, the chosen sub-strategies are shown in the Figure 4.16 and 4.17, respectively. The chosen sub-strategies being combined horizontally by a new root node are shown in Figure 4.18.

FIGURE 4.13: Strategy 1

FIGURE 4.14: Strategy 2

4.7.3 Type 3 operator

As given in Algorithm 5, Type 3 crossover takes two DRSs as it is and combines one complete strategy with another complete strategy as a child to its root node. It changes the names of all the nodes to avoid the prospective conflict and returns the resulting offspring strategy afterward as shown in Figure 4.19 where it combined the strategies given in Figure 4.13 and 4.14.

4.7.4 Type 4 operator

This type of crossover deals mostly with the vertical combinations rather than horizontal, it takes two DRSs and combines them by replacing: 1) a virtual node of one

FIGURE 4.15: Crossover type 1 operator

FIGURE 4.16: Part of the strategy 1

strategy with the virtual node of another strategy, 2) a virtual node of one strategy with the leaf node of another strategy, 3) a leaf node of one strategy with the virtual node of another strategy, 4) a virtual node of one strategy with a complete other strategy, 5) a

FIGURE 4.17: Part of the strategy 2

FIGURE 4.18: Crossover type 2 operator

leaf node of one strategy with a complete other strategy.

Algorithm 5:

```
1 crossoverT3 (str1, str2) {
2     str1.setChildNode(str2);
3     changeNames(str1);
4     return str1;
5 }
```

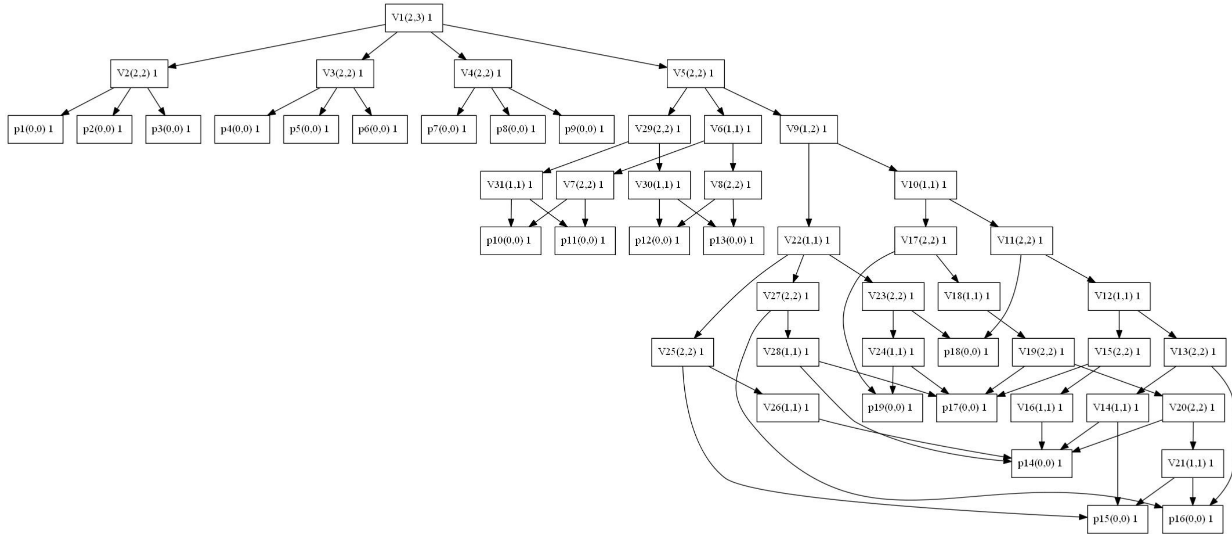

FIGURE 4.19: Crossover type 3 operator

Type 4 operator (a)

For instance, Figure 4.20 and 4.21 represent strategies with their chosen crossover points highlighted in green. These chosen virtual nodes (highlighted in green) are replaced with each other to create two offspring strategies shown in Figure 4.22 and 4.23.

Type 4 operator (b)

For instance, the chosen highlighted virtual node of the strategy shown in 4.22 is replaced with a leaf node as shown in Figure 4.24.

Type 4 operator (c)

For instance, the earlier replaced leaf node (p5) shown in Figure 4.24 can easily be replaced again with any other virtual node such as in Figure 4.22.

Type 4 operator (d)

For instance, the chosen highlighted virtual node of strategy 1 shown in 4.25 is replaced with the complete strategy 2. The resulted offspring strategy is shown in Figure 4.26.

FIGURE 4.20: Strategy 1

FIGURE 4.21: Strategy 2

Type 4 operator (e)

For instance, the chosen highlighted leaf node of strategy 2 shown in 4.27 is replaced with the complete strategy 1. The resulted offspring strategy is shown in Figure 4.28.

Moreover, while performing these multi-type crossovers, the algorithm skips some special case crossovers, i.e., two complete strategies swap or two leaf nodes swap to avoid redundancy, hence, maintains the diversity of the solutions, thereby maintaining the effectiveness of the proposed approach.

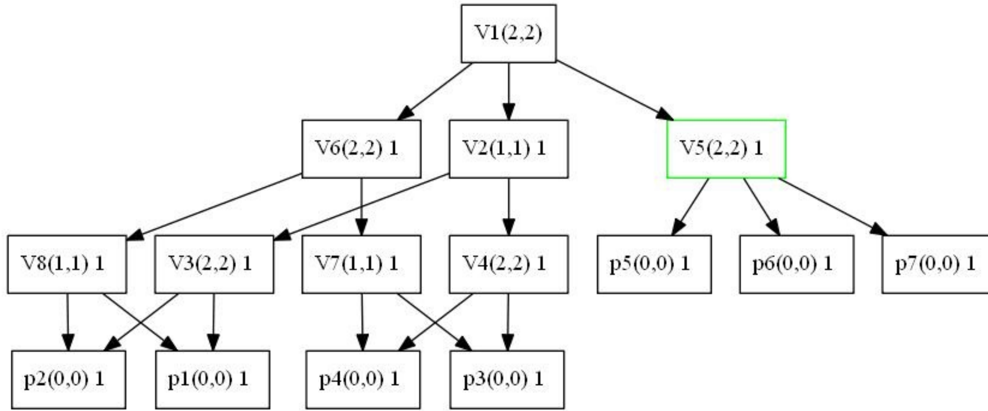

FIGURE 4.22: Crossover Type 4 operator (a) - offspring DRS 1

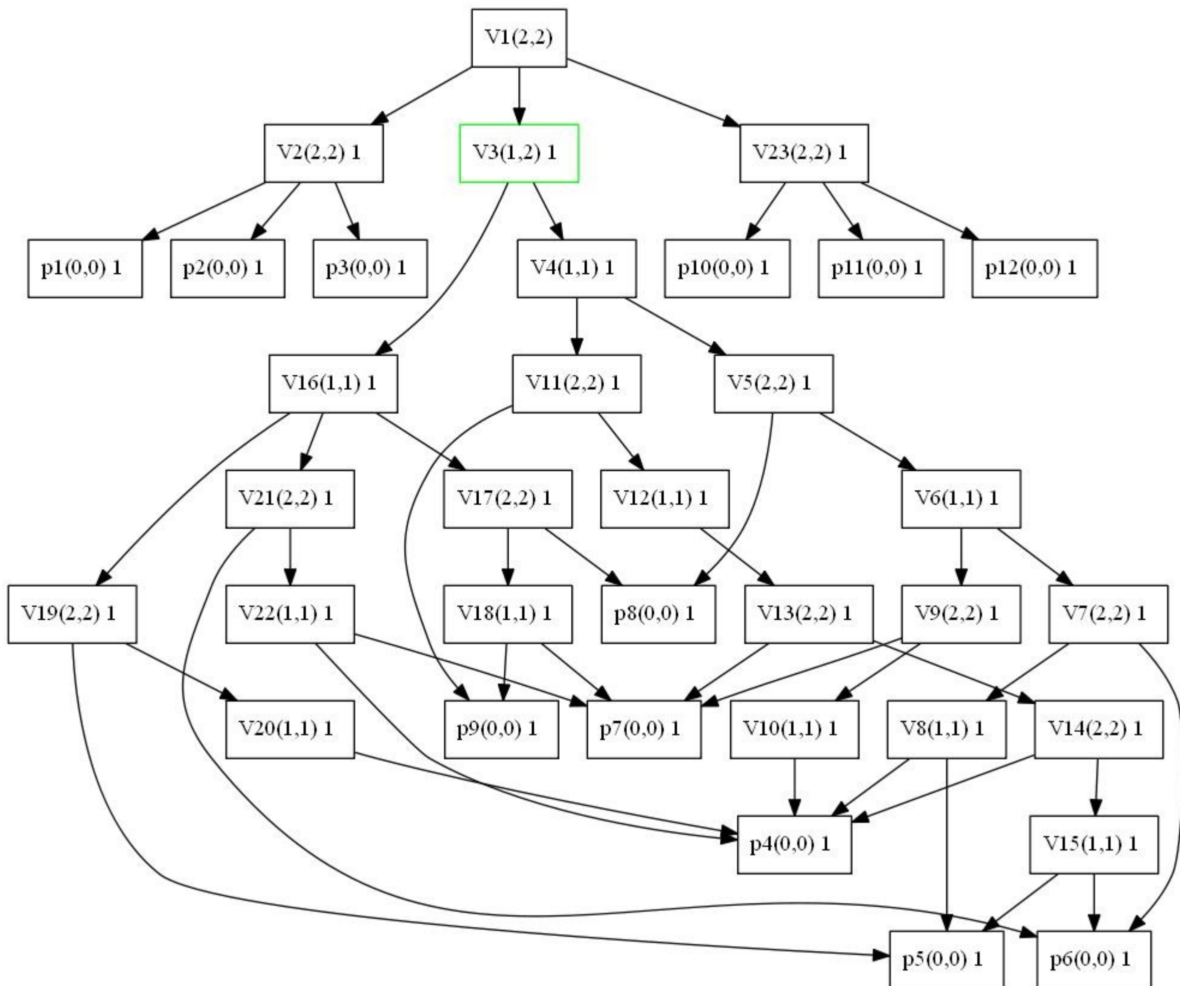

FIGURE 4.23: Crossover Type 4 operator (a) - offspring DRS 2

4.8 Mutation operators for the strategies

The algorithm also performs multi-type mutations on the DRSs with specified probabilities in the system parameters. The mutation slightly changes the properties of a DRS

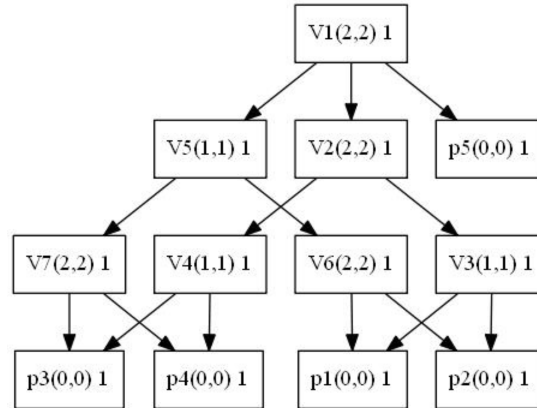

FIGURE 4.24: Crossover Type 4 operator (b) - offspring DRS

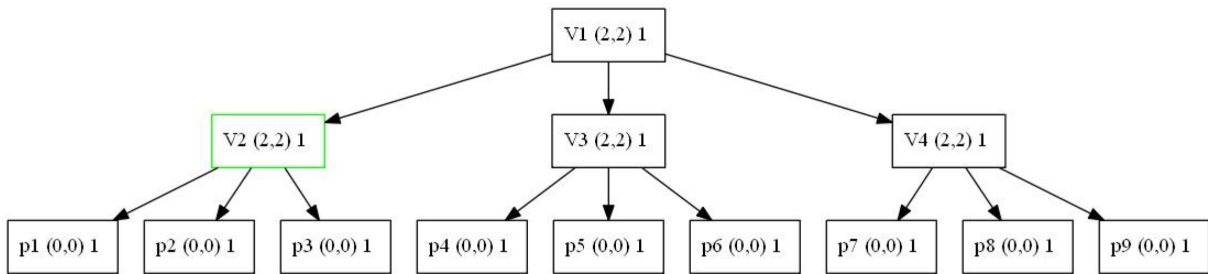

FIGURE 4.25: Strategy 1

so that it becomes different (possibly better) than its primitive form or shape.

4.8.1 Type 1 operator

Crossover is performed every time while mutation is performed only with a certain probability. This type of mutation is given in Algorithm 6, which slightly changes the quorum size and the weightage of nodes (votes), but carefully enough to not destroy the 1SR consistency. The weightage is changed to make certain replicas more important than others. Once the weightage is changed, the quorums must also be altered accordingly, under the Conditions (2.1) and (2.2) to adhere to 1SR. Besides randomness, the mutation points have to be picked carefully by the algorithm in order not to annihilate, again, the 1SR property of a solution and thus, rendering it invalid. This is explained by the strategy given in Figure 4.29 where mutation points are chosen and being highlighted in green. The strategy gets mutated from these chosen points keeping in mind the consistency issues, and the resulting mutated strategy exhibiting different properties is shown in Figure 4.30.

4.8.2 Type 2 operator

This type of mutation reduces the structure of the replication strategy by removing a few replicas from it to confine the structure within the threshold.

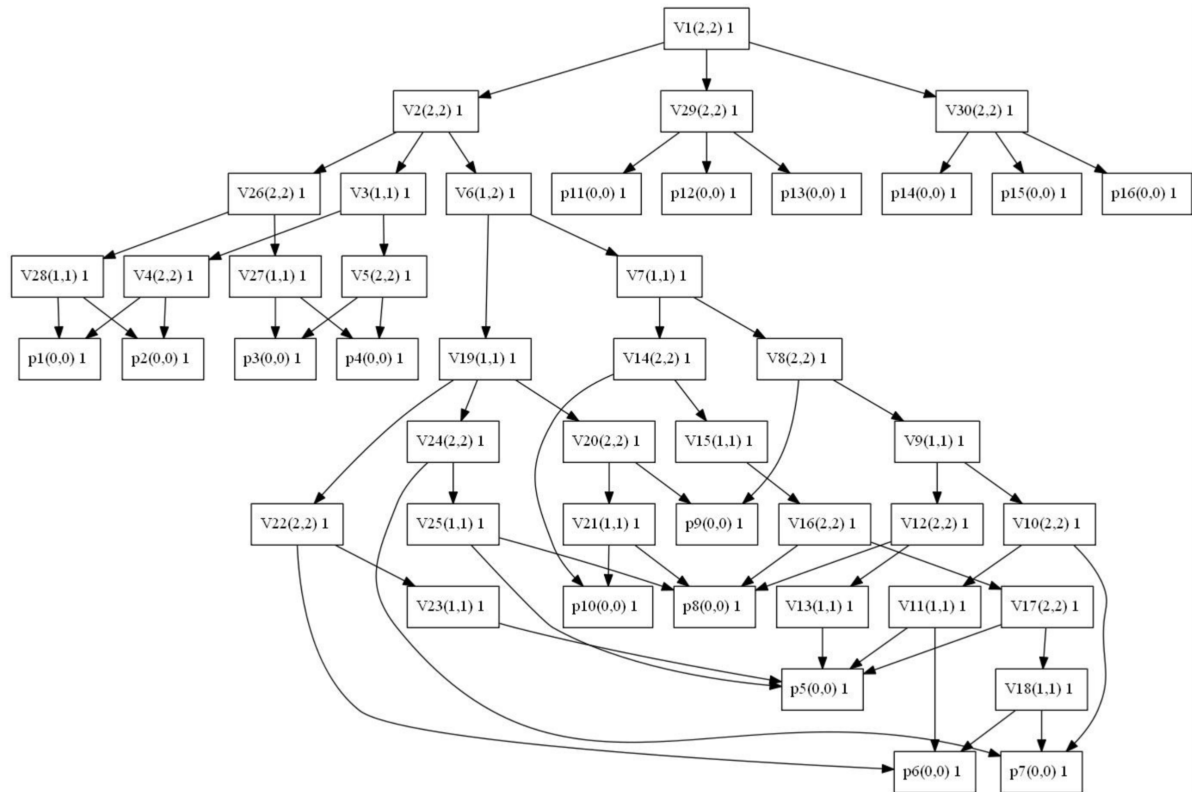

FIGURE 4.26: Crossover Type 4 operator (d) - offspring DRS

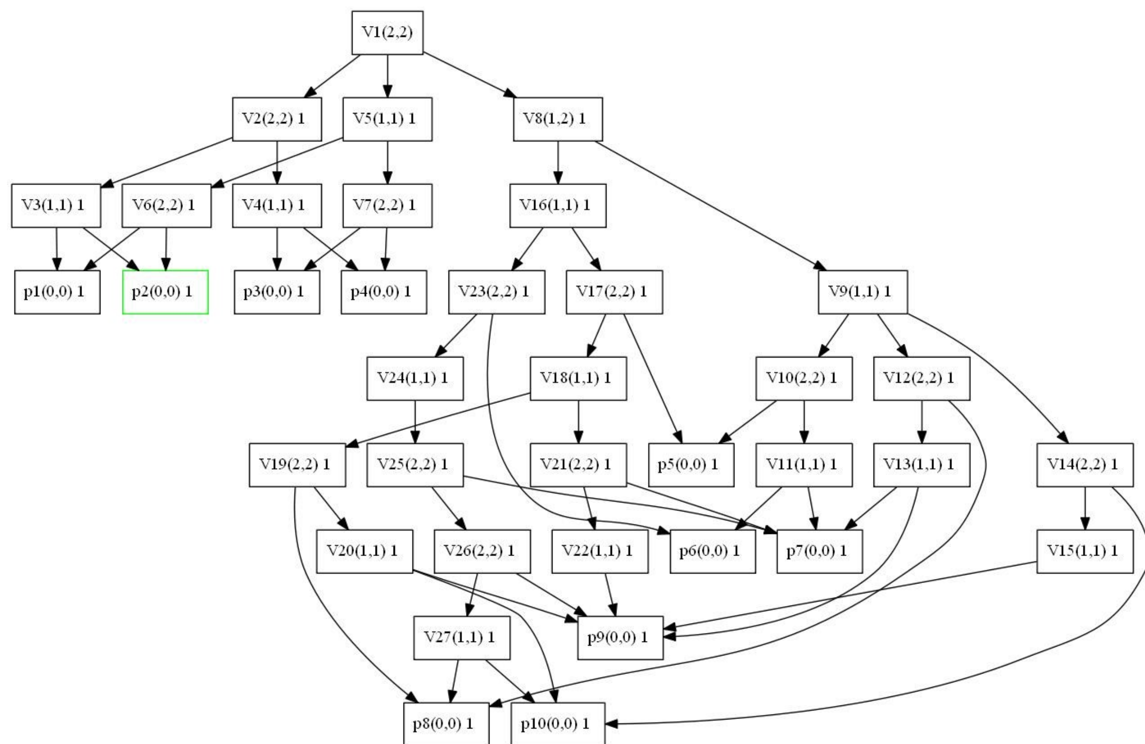

FIGURE 4.27: Strategy 2

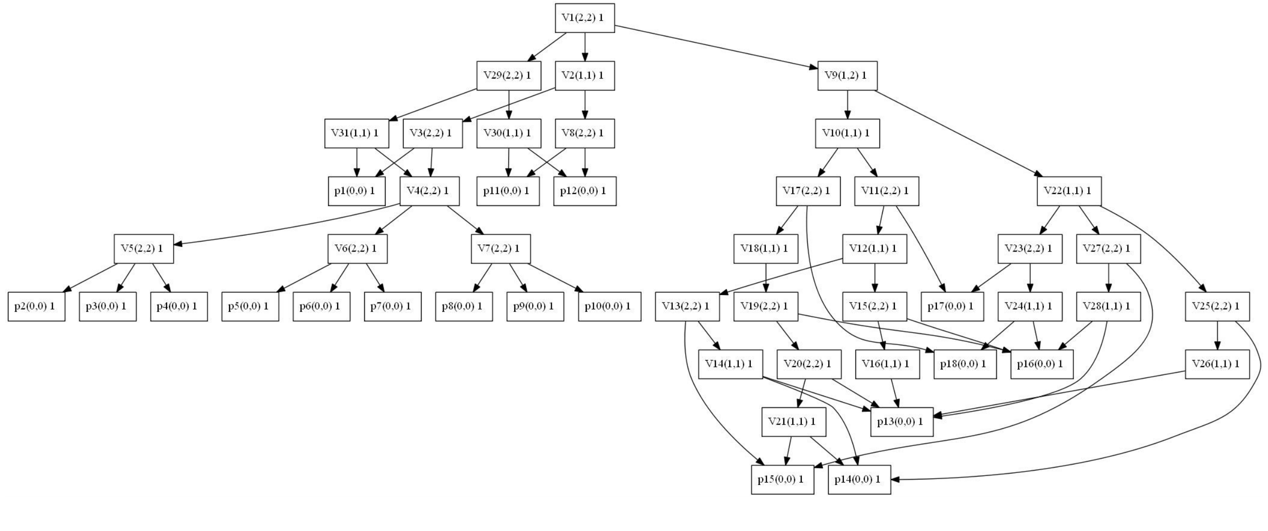

FIGURE 4.28: Crossover Type 4 operator (e) - offspring DRS

Algorithm 6:

```
1 mutation1 (str) {
2     point = findMutationPoint (str);
3     do{
4         alterVotes (str, point);
5         alterQuorums (str, point);
6     }
7     while (checkConsistency (str));
8     return str;
9 }
```

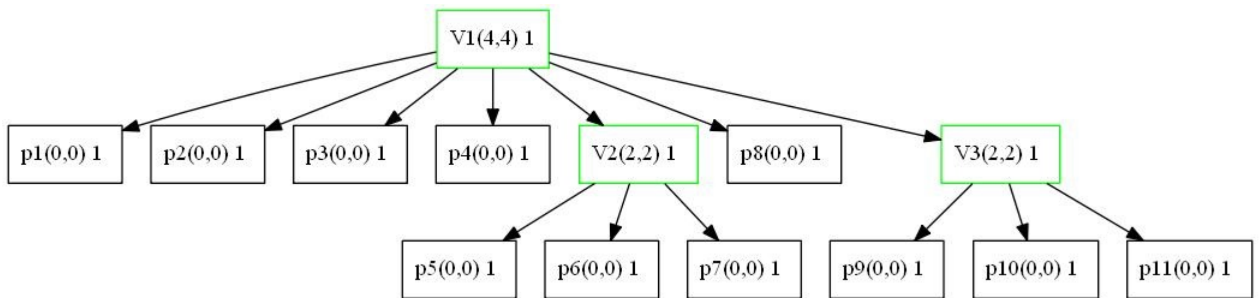

FIGURE 4.29: Chosen points for mutation

4.9 System parameters

Once a scenario is specified to find an appropriate DRS to fulfill it, the system parameters are set for the algorithm to run. The system parameters are as follows.

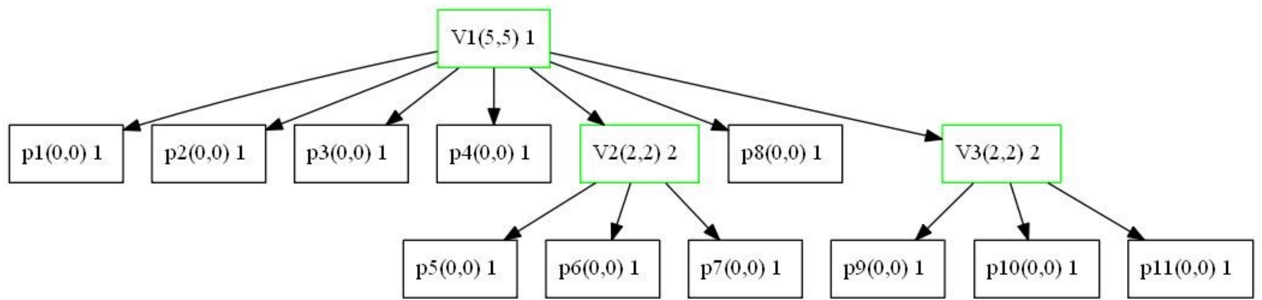

FIGURE 4.30: Mutated DRS

4.9.1 μ and λ

Having provided the repository to select the respective DRSs, the μ and λ values are also set as system parameters for the algorithm to start. μ is the restriction on the number of parents that are used to form the next generation (through genetic operators) and λ is the restraint on the number of off-spring strategies generated as an outcome using μ number of parent DRSs. μ and λ restrict the number of parents and offspring DRSs in the process of GP.

4.9.2 Initial population probability

This is an additional probability to select DRSs from the initial population as well in order not to let the existing strategies vanish away in the next generations.

4.9.3 Intra-crossover probability distribution

The crossover between the DRSs is performed all the time; however, there are multiple types of crossover. Therefore, probabilities among these crossovers are distributed to use some type of operators more frequently than others depending upon the nature of the problem.

4.9.4 Mutation probability

Unlike crossover operators, mutation operators are seldom used; therefore, prospectively a very low mutation probability, i.e., 0.2, 0.3, is defined to execute mutations accordingly.

4.9.5 Intra-mutation probability distribution

There are two types of mutations used in this research, among which the mutation probabilities are distributed in order to determine the frequency of these operators in the pursuit of a solution accordingly.

4.10 Summary

This chapter introduces the basic working framework of this research, the fault model, the scenario parameters, and subsequently, demonstrates some manually combined examples of voting structures leading to the automatic mechanism using General object-oriented genetic programming to design DRSs automatically. It explains the proposed genetic programming approach that includes the algorithm of GP, fitness function, crossover, mutation operators, and their rules, as well as the working mechanism. As for the fitness function, the objectives in the scenario are 1) conflicting in nature, 2) imbalanced in a way that values for some objectives are probability ranges while others are very large, 3) some of them must be maximized and some of them must be minimized. The fitness function transforms this multi-objective problem into a single-objective problem for determining the quality of a solution through this single-valued metric. A higher fitness value determines the appropriateness of a solution to the specified constraints. In this regard, the proposed approach uses multi-type crossovers and mutations to explore more possibilities of enhancing the fitness of DRSs since it equips the algorithm with more power as compared to single-type. These crossovers and mutations are performed on the replication strategies with certain probabilities and also the intra-operators to execute them, accordingly, on the DRSs. These intra-crossover and intra-mutation probability distributions are for using some operators more than the others to accordingly find a better solution. New offspring strategies are produced as a result of these genetic operators. The mechanism has the potential to evolve DRSs as computer programs through crossover and mutation operators, thus, DRSs can be optimized over several generations, and go through several optimization phases to eventually stop at the termination criteria.

Chapter 5

Experiments and results

In this chapter, the results of the proposed approach are presented. As for the assumptions w.r.t. the fault model, it is to simplify the problem, which otherwise easily goes out of hand and it becomes difficult to carry out the analysis. Moreover, such a fault model is also necessary to find out the properties as well as the potential of the strategies that would have been blurred otherwise and could not be detected. For instance, the point-symmetric property [8] in data replication strategies cannot be detected with more complex and realistic parameters. The same is the case with dynamic replication, where it becomes very complex to handle even a few replicas. Also, this fully connected behavior with no communication failures is not necessary for correctness purposes, but rather for analysis purposes, which means it becomes easier to carry out the experiments and analysis. Furthermore, the adopted notions, i.e., average minimal cost (average of the minimum replicas required to execute an operation), as well as the overall fault model being used here is widely used. The newly discovered strategies through our genetic approach are compared on all the discretized values of p to make a fair and, therefore, a realistic comparison. Next, the constraints and properties are specified that the system is expected to meet in the replication strategies.

5.1 Scenarios

Each scenario is comprised of a set of threshold values, which the system is supposed to find in the population of replication strategies. It calculates the values for each replication strategy to check the meeting criteria. Such values are being calculated by the respective objective functions, and the proposed mechanism then offers newer solutions using existing ones and optimizes them (as computer programs) by recombination of those strategies mostly, and comes up with unique solutions adhering to the specified properties. This stochastic nature of the technique makes it much easier to create innovative replication strategies w.r.t. the specified problems every time the genetic process is initiated. Next, the scenarios are stated for which the system is run to find solutions accordingly.

5.1.1 Scenario 1

This scenario consists of the desired read and write availabilities and their respective costs, which must be achieved within the threshold of a maximum of 16 replicas and some availability p of individual replicas. However, the cost is not important in this

case, therefore, full weightage is given to availability. The desired read availability and write availability thresholds are 0.80 and 0.70, respectively, on a node availability of 0.6. The expected read and write costs are set to seven each. The strategy is expected to accomplish these properties inside a threshold of no more than 16 replicas in total. However, the scenario specifies the availability to be fully important, therefore, full (fitness) weight is assigned to the availability of the access operations, in this particular case. Even though p is a scalar value, but the strategy is compared on all the discretized values of p to make a fair and, therefore, a realistic comparison.

p = 0.6, ϵ = 16, α = 0.80, β = 0.70, γ = 7.0, δ = 7.0, fw = 1.0

5.1.2 Scenario 2

This scenario is almost the same as the first one, but with a slight increase in the expected write availability. The write availability of Scenario 1 is slightly changed up to two decimal places. Even this slight change to some decimal places impacts the availability greatly in real-time, [20]. So, these slight changes can make a huge difference, therefore, are hard to achieve. The required read and write availability thresholds hence are 0.80 and 0.72, respectively. This must be achieved within a threshold of no more than 16 replicas on a replica availability of 0.6. Cost is given to be less than or equal to seven for each operation; however, full (fitness) weightage is given to the availability. This specified scenario boils down to fitness of 1.520 to be achieved.

p = 0.6, ϵ = 16, α = 0.80, β = 0.72, γ = 7.0, δ = 7.0, fw = 1.0

5.1.3 Scenario 3

In this example, the availability of the replicas is set to 0.7 while read and write availabilities are set to 0.9 for each operation, inside a total cost of eight for the access operations. The availability is more important than the cost in this scenario, therefore, a weightage of 70% is given to availability and the rest to the cost. These objectives have to be achieved by no more than 16 replicas.

p = 0.7, ϵ = 16, α = 0.90, β = 0.90, γ = 4.0, δ = 4.0, fw = 0.7

5.1.4 Scenario 4

Here, the same scenario parameters are used as given in scenario 2, but with slightly different system parameters, i.e., different mutation and intra-mutation probabilities. Hence, the scenario is defined, again, with a replica availability of 0.6 to achieve an availability of a read 0.80 and a write 0.72 through a total number of replicas no more than 16. The costs of the read and write operations are set to seven each, but availability is given full weightage. The fitness to be achieved depending upon this scenario is 1.520. Even though p is a scalar value, we compare the strategy on all the discretized values

of p to make a realistic comparison. This example will demonstrate the effectiveness of the proposed system in generating a variety of powerful solutions by even slight use of genetic operators.

p = 0.6, ϵ = 16, α = 0.80, β = 0.72, γ = 7.0, δ = 7.0, fw = 1.0

5.2 Results and discussions

Prior to setting the parameters and running the system on the scenario, here, a few examples of hybrid replication strategies generated by the proposed system are discussed. Figure 5.1 gives a relatively simple example of a hybrid DRS generated by the algorithm, which consists of 11 replicas. It can be seen that although the DRS is not very complex and maintains a tree-like structure rather than an acyclic one, yet it is so powerful and optimized in terms of its availability and cost that it is competing the Majority Consensus Strategy (MCS), which is believed to be the best in terms of its availability of write access operations. When compared, the hybrid DRS in terms of its availability is so close to MCS. It is almost the same for higher values of p; however, it is far better when it comes to the cost comparison.

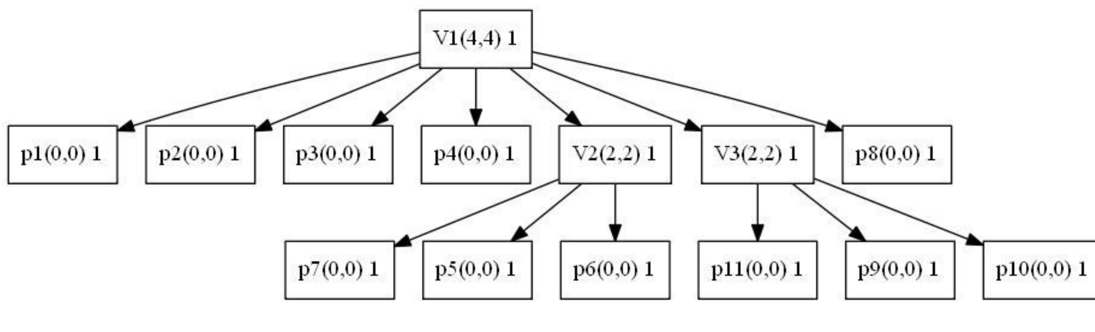

FIGURE 5.1: Hybrid Strategy 1

The availability and cost graphs on the discretized values of p are shown in Figure 5.2 and Figure 5.3, respectively, where Strategy 1 indicates the MCS while Strategy 2 represents a hybrid DRSs. Both strategies consist of 11 replicas each. It can be seen that in terms of operational availability the hybrid strategy is converging onto the same values as MCS for higher values of p. This is a quite good availability but more importantly, it outclasses the MCS in terms of its cost in all the cases. Hence, it covers a scenario, which could have been left unaddressed otherwise.

In the best case, out of 11, it only takes four replicas each to perform a read and a write operation while the total cost for MCS is 12 for all the cases. This is a good example of a relatively less complicated DRS where the availability is not compromised and yet the cost is reduced significantly through the hybrid approach via genetic programming.

Figure 5.4 shows a relatively complex but more economical example of an up-to-now unknown hybrid replication strategy designed via genetic programming, exploiting the voting structures. It is comprised of both the Grid Protocol and the Triangular Lattice Protocol (TLP), where it unprecedentedly combines Grid Protocol comprising four replicas with TLP of six replicas, resulting in a total of ten replicas. It demonstrates

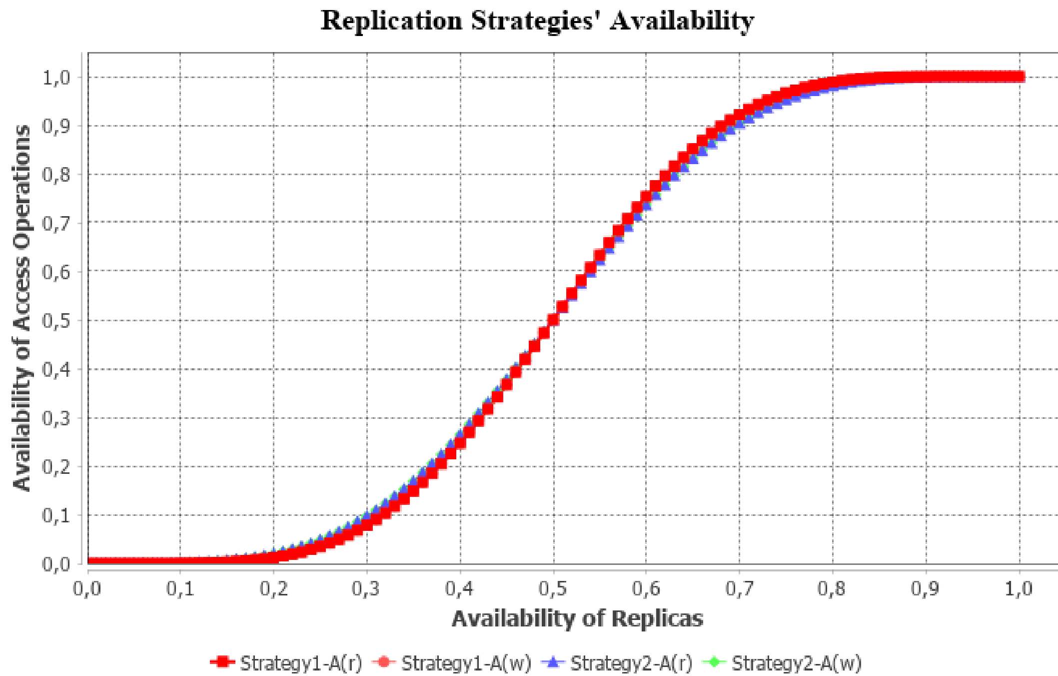

FIGURE 5.2: Hybrid DRS 1, availability of the access operations

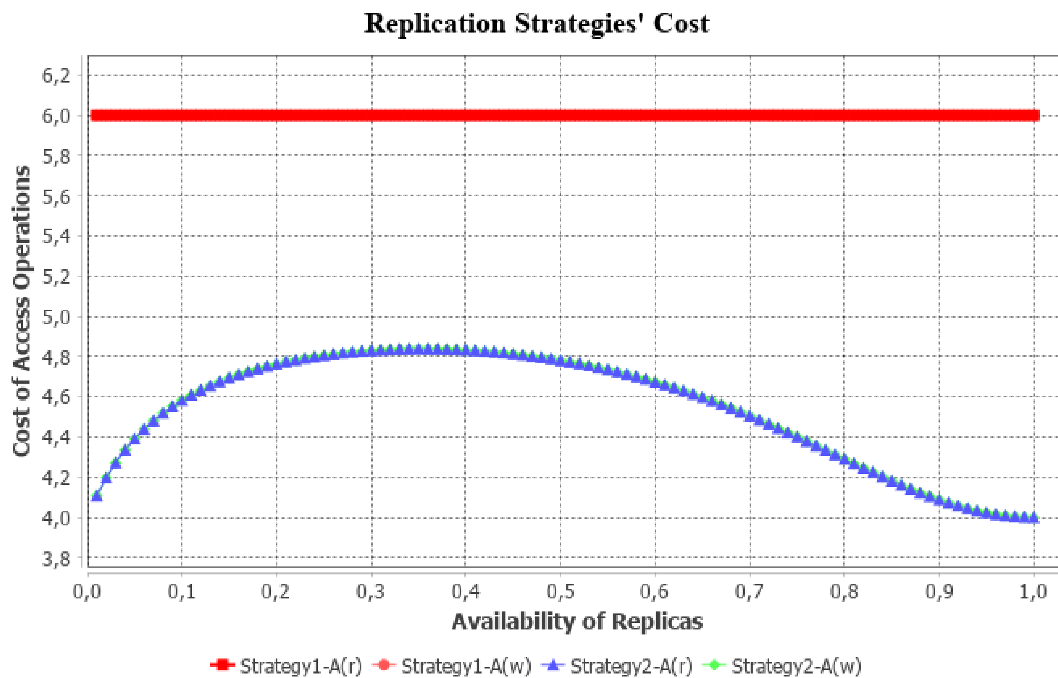

FIGURE 5.3: Hybrid DRS 1, cost of the access operations

an instance of horizontal crossover, which has lowered the cost by a great value while maintaining a very good availability of the access operations.

Figure 5.5 presents and compares the availability of MCS with the proposed hybrid approach of the same number of replicas. Red and pink lines represent the availabilities of the read and write operations, respectively, for the MCS. Whereas blue and green

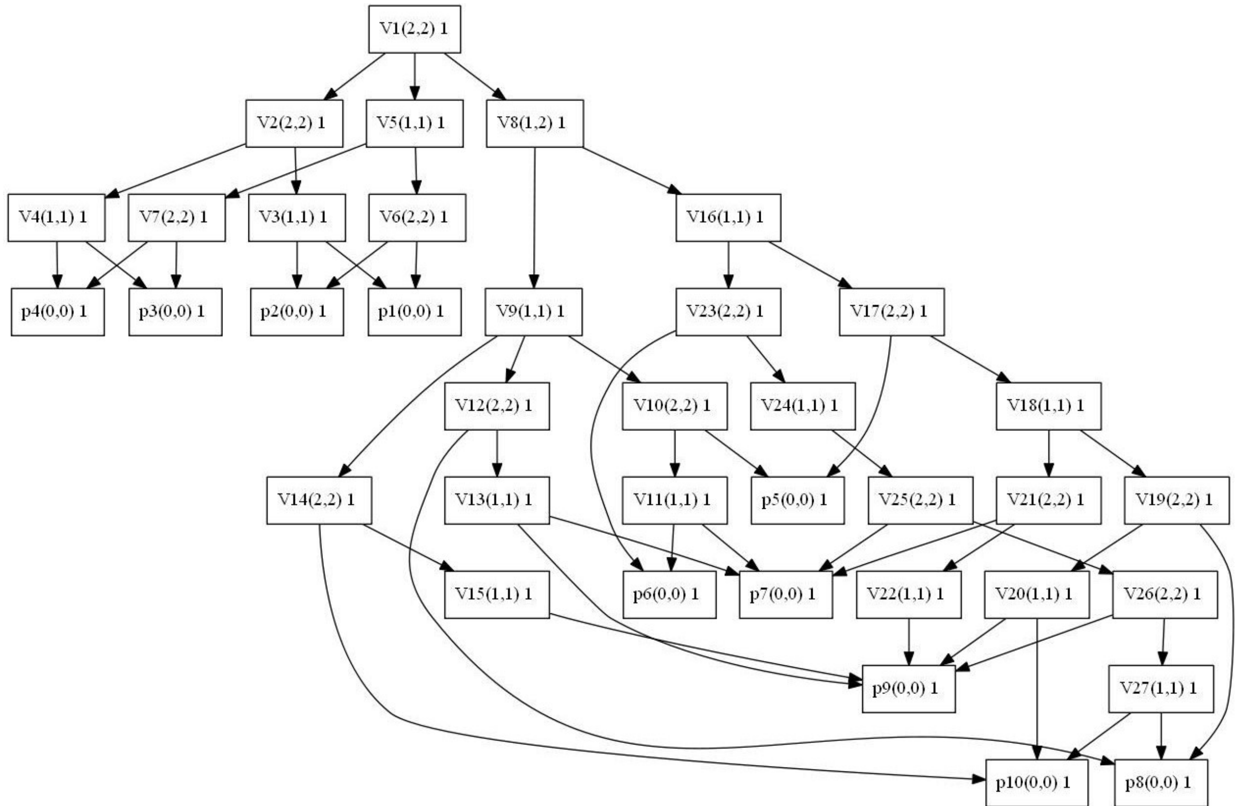

FIGURE 5.4: Hybrid strategy 2

lines show availabilities of read and write operations, respectively, for the hybrid strategy. The latter (hybrid) is competing fairly with the former (MCS), considering the fact that MCS is known to be the best for its availability, particularly for the critical write availability. In comparison, it can be noticed that availabilities are almost the same for the later values of p.

Figure 5.6 enables us a closer view, where it can be observed that for hybrid strategy, the respective availabilities of the access operations converge onto almost the same values for later values of p, which is a very good operation availability considering the strong hardware nowadays.

As for the cost, as shown in Figure 5.7, hybrid DRS is much cheaper as compared to the MCS. It costs almost half of the MCS, in best cases, it only takes three replicas to perform an access operation whereas the cost of the access operations for MCS remains a constant of 11 replicas in total. Here, again, it is evident that the cost has been significantly reduced while not sacrificing availability too much, covering another prospective scenario, where a further reduced cost could be required.

As some powerful examples of newly generated previously unknown voting structures via genetic programming have been demonstrated, now we move on to specifying scenarios, explained earlier.

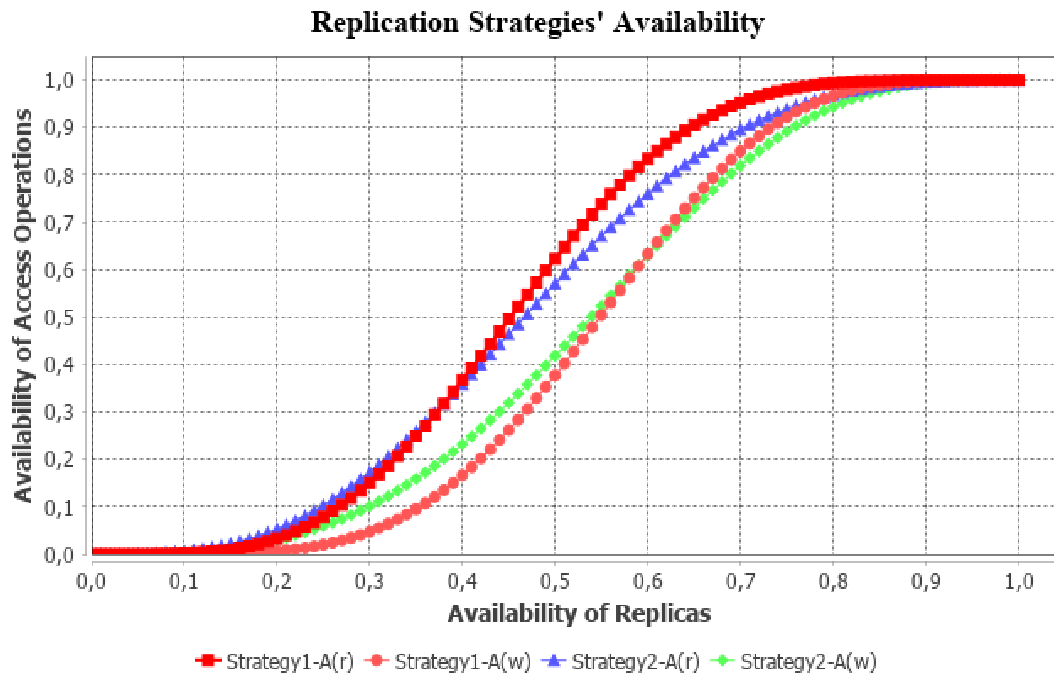

FIGURE 5.5: Hybrid DRS 2, availability of the access operations

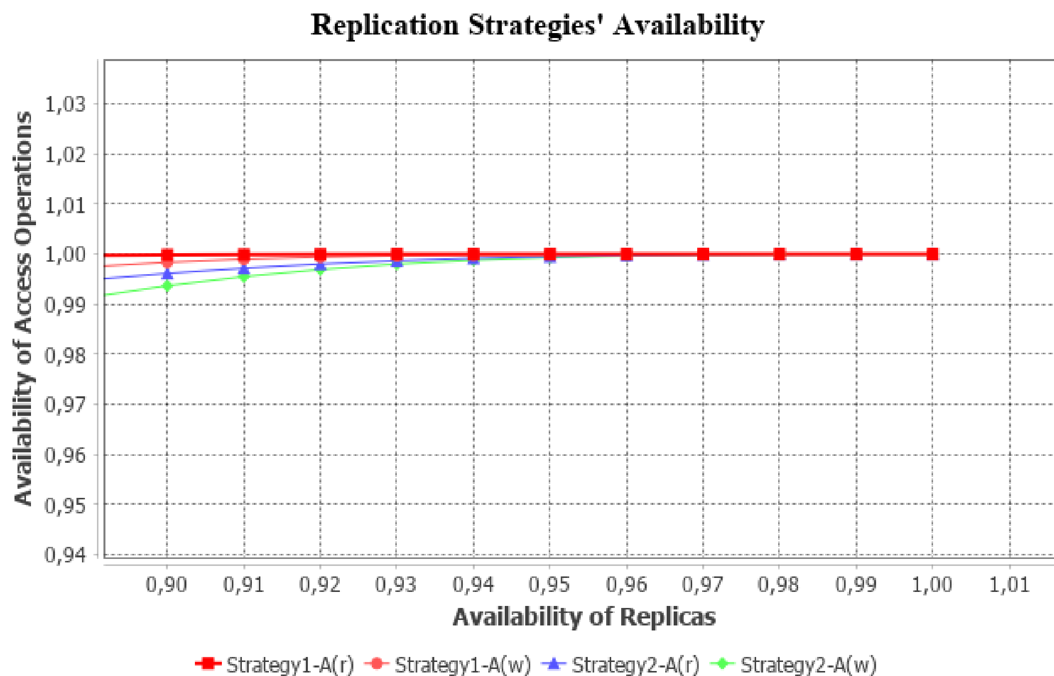

FIGURE 5.6: Hybrid DRS 2, Zoom-in availability graph

5.2.1 System parameter settings for scenario 1

As mentioned earlier, this scenario consists of the desired read (0.8) and write (0.7) availabilities on a replica availability of 0.6, inside the threshold of 16 replicas. The expected read and write costs are set to seven each, but full weightage is given to

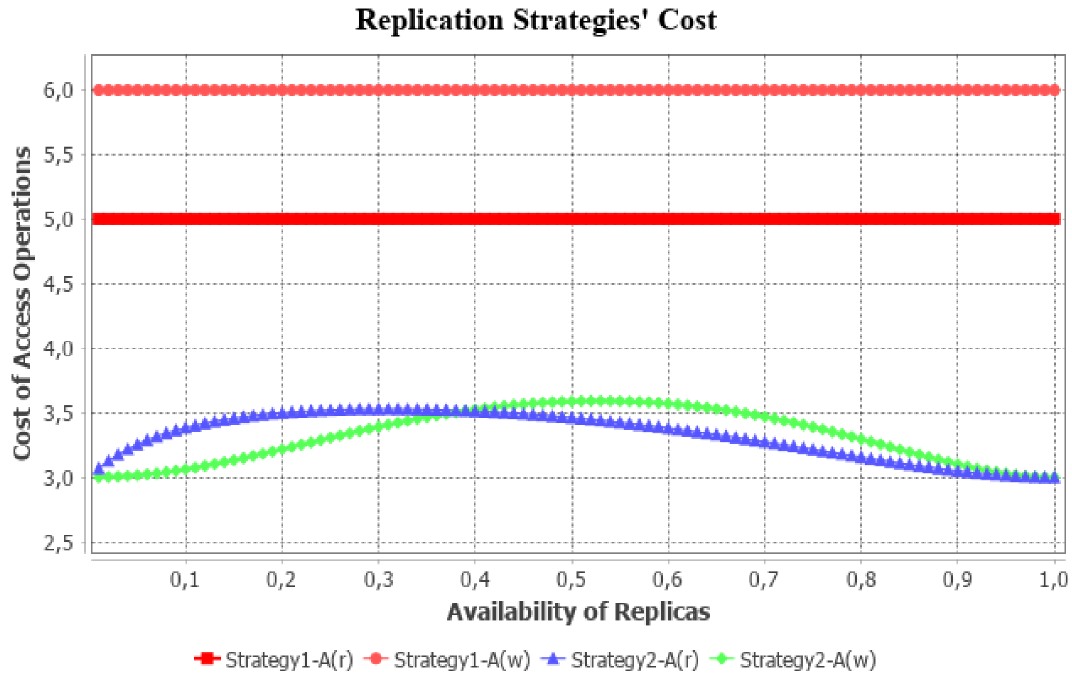

FIGURE 5.7: Hybrid DRS 2, cost of the access operations

availability in this case. Even though node availability is a scalar value, but the strategy is compared on all the discretized values of p to make a fair comparison.

p = 0.6, $\epsilon = 16$, $\alpha = 0.80$, $\beta = 0.70$, $\gamma = 7.0$, $\delta = 7.0$, fw = 1.0

Having defined the scenario, now the system parameters are set to run the algorithm accordingly. Here, the number of parent and offspring strategies are set to six and 15, respectively. The initial population is only used once in the genetic process in the very first generation. The crossovers are performed all the time while the mutation is performed with a probability of 0.2. The system is run to find out an appropriate replication strategy.

$\mu = 6$, $\lambda = 15$, mutationProb = 0.2

5.2.2 Results for scenario 1

The algorithm is run, having set μ and λ to six and 15 respectively, along with a mutation probability of 0.2. Figure 5.8 depicts a 2D representation of generated solutions for the specified problem. It represents all the genetic strategies (being represented by red marks) generated in this entire genetic process, all of them are innovative and exhibit different unique combinations of several strategies combined, thereby contributing to fulfilling different goals w.r.t. the trade-offs between the quality metrics. Here, the y-axis represents the total cost while the x-axis the total availability of the access operations.

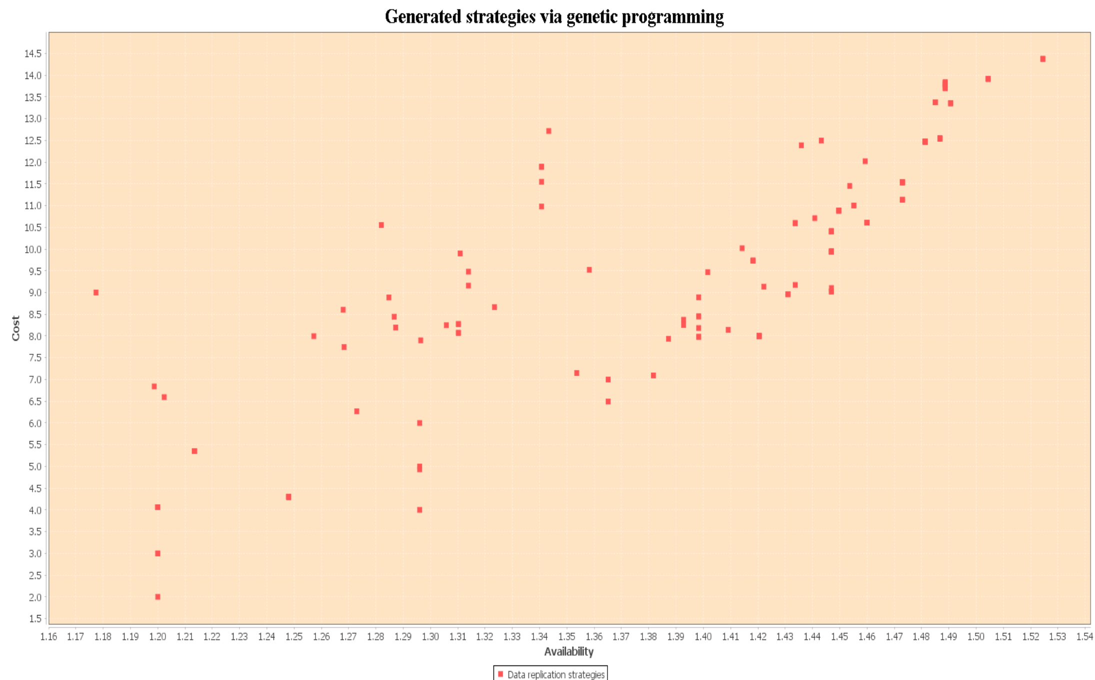

FIGURE 5.8: 2D representation of the generated DRSs

As shown in Figure 5.9, this view can be understood more easily by dividing it into four equal quadrants; quadrant 1 (top right corner) indicates better availabilities at the expense of costs, quadrant 2 (top left corner) shows that availabilities and costs are both worse, quadrant 3 (bottom left corner) represents better costs at the expense of lower availabilities, and quadrant 4 (bottom right corner) offers solutions which are better in both, availabilities and costs. It can be seen that we do not have too many solutions in the fourth quadrant in this case.

An appropriate solution (circled in red) exhibiting 16 replicas, satisfying the criteria, is picked at run-time. Figure 5.10 depicts the fitness of every individual DRS and the way it evolves. The x-axis represents the number of DRSs and the y-axis denotes the fitness value of every individual strategy. The red line indicates the fitness of the DRSs while the pink and blue lines represent the availabilities of read and write operations, respectively. It can be noticed that it starts with only a few strategies of low fitness, which implies that the repository does not have a satisfactory solution to the problem. Then, the fitness improves and begins to evolve gradually through crossover and mutation operators of genetic programming until the loop stops over the desired termination condition.

Figure 5.11 illustrates how the fitness of DRSs grows with every generation. The graph shows the fitness of the best DRSs among every generation. The x-axis represents the number of generations while the y-axis indicates the fitness value of the best replication strategy of a respective generation. It took 10 generations for the system to find a suitable DRS that satisfies the given scenario. Though it is a soft-check as in this case, the write availability is extremely close to 0.7, i.e., 0.694, which suggests that fulfilling the fitness criteria does not necessitate the fulfillment of the scenario constraints. Therefore, for some of the other scenarios, a hard check is done, which checks both the fitness and individual constraints to which the fitness subject. However,

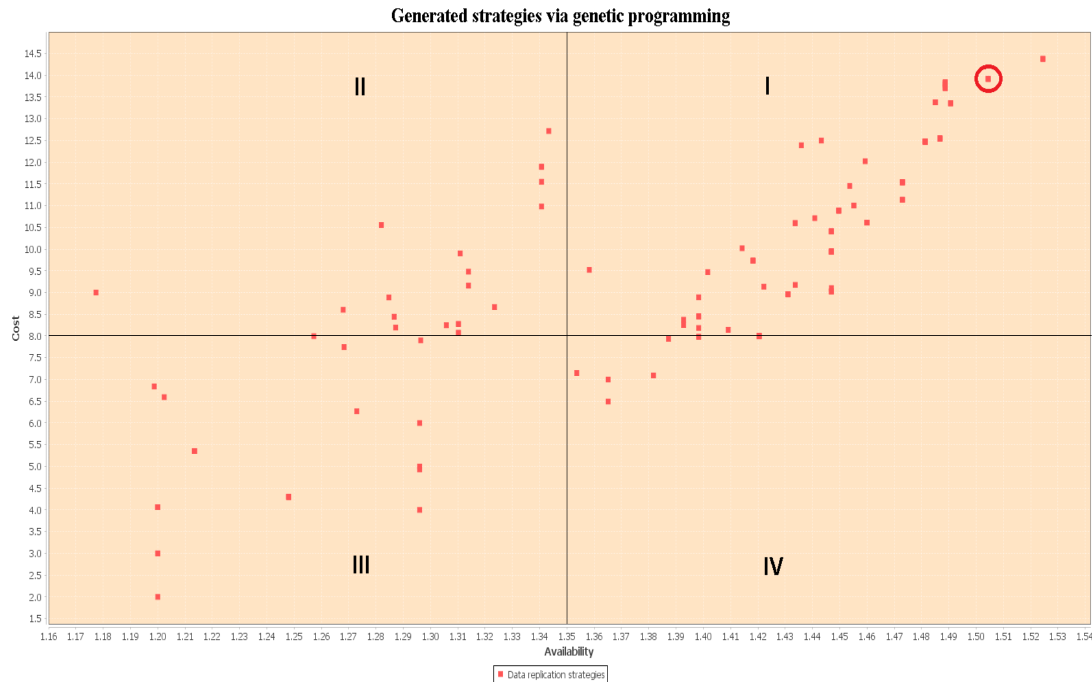

FIGURE 5.9: 2D representation of the generated DRSs

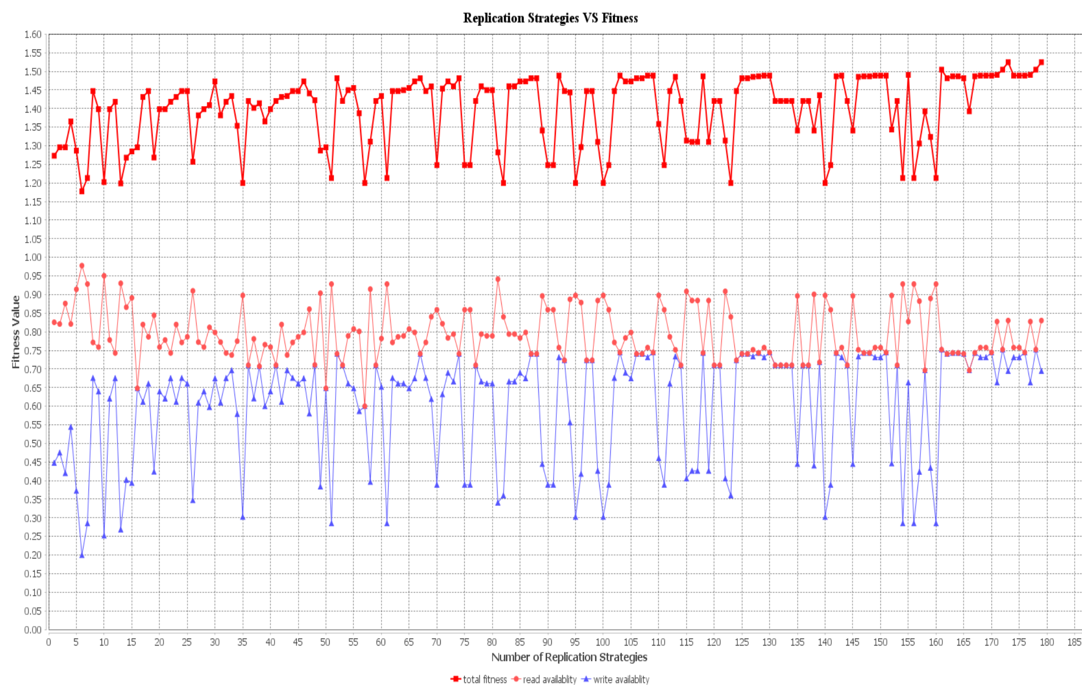

FIGURE 5.10: Fitness graph

in this case, it starts from fitness of 1.365 and gradually but consistently continues to climb up until the desired fitness of 1.525 (even better than the specified one) is achieved.

Figure 5.12 presents the identified optimized strategy DRS constituting 16 replicas in total with certain nodes in the voting structure being more important than others

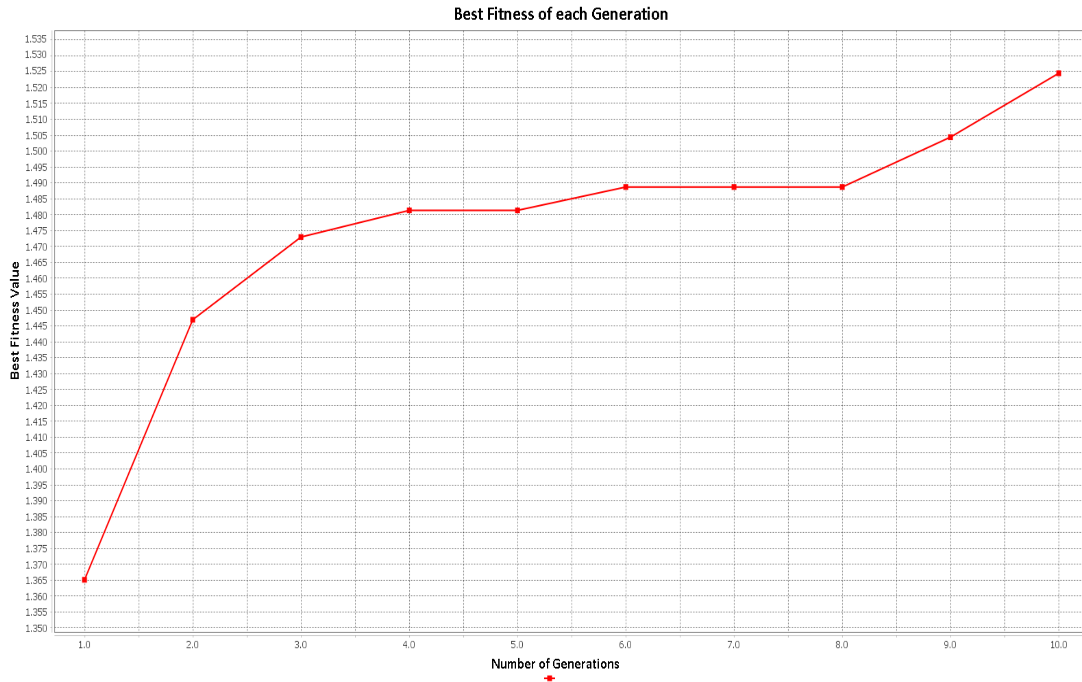

FIGURE 5.11: Populations' evolution

in the collection of quorums. The heterogeneous nature of this structure along with variable votes and quorums together reflects its hybrid nature, providing an up-to-now unknown replication strategy, serving to meet the specified constraints of availabilities and the number of replicas while at the same time being not too expensive w.r.t. the access operations either. As for availabilities of the access operations, it fairly competes with the MCS, which as already mentioned is considered to be the best w.r.t. availability, particularly, for the critical write operation's availability.

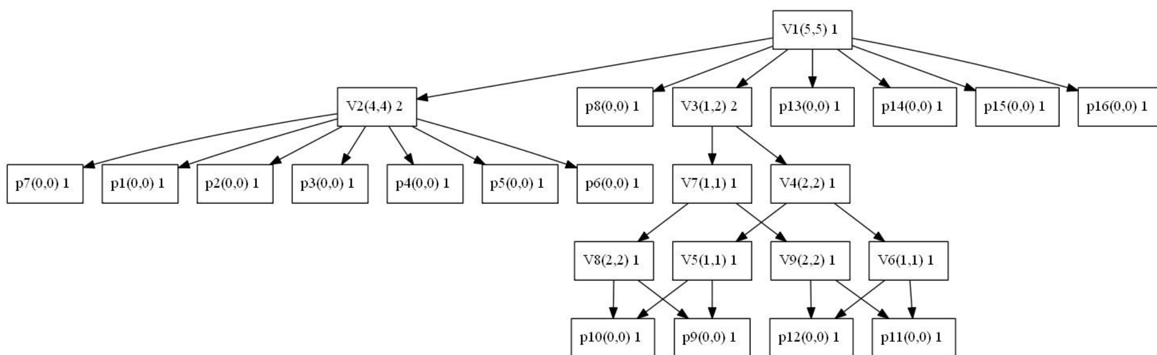

FIGURE 5.12: Optimized hybrid DRS for the given scenario

Figure 5.13 shows the availability graph for the access operations of the identified DRS on discretized values of p. The newly designed DRS fulfills the specified scenario of thresholds. The x-axis represents the node availability while the y-axis indicates the availability of the access operations. The point-symmetry of the graph overtly displays an extremely high availability for access operations.

Figure 5.14 represents a comparison between MCS and the discovered hybrid DRS. Red

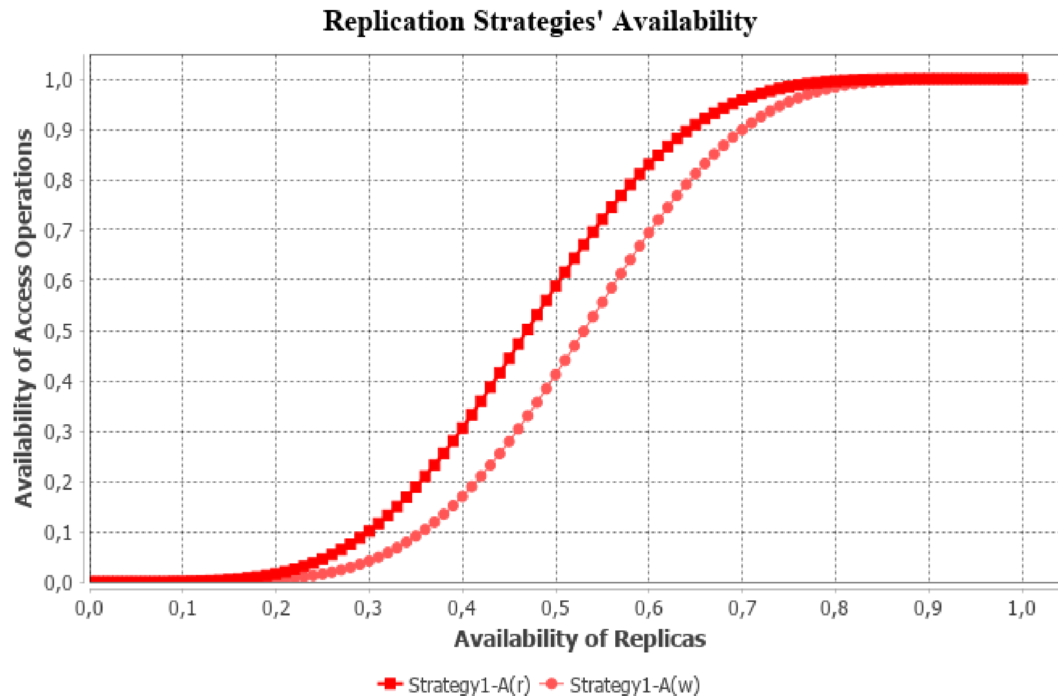

FIGURE 5.13: Availability graph of the read and write operations

(with squares) and pink (with dots) lines represent the availabilities of read and write operations, respectively, for MCS. Blue and green lines depict the availabilities of read and write operations, respectively, for the hybrid strategy. It is evident from the figure that the proposed approach fairly competes with MCS and operation availabilities are better on p values being 0.5 or less while extremely close for all the remaining p values. It can be noticed that operation availabilities converge onto almost the same values for later values of p, which is a very good operation availability considering the strong hardware nowadays. However, the discovered hybrid strategy is far more economical.

Figure 5.15 shows the cost comparison between the two mentioned strategies. Blue and green lines indicate the costs of read and write operations, respectively, for the hybrid DRS. It can be noticed that despite fairly competing with MCS in availabilities, the hybrid replication strategy is very cheap in its cost. It could perform an operation by merely accessing five replicas each; however, MCS of the same size takes 17 replicas in total to perform both access operations. Hence, the operation costs have been significantly decreased while not much compromising on availabilities. Despite the fitness calculation is expensive, considering the stochastic nature, the tests (with the same experimental settings) are repeated at least five times resulting in consistently satisfying results by generating new DRSs (each time) that meet the criteria. This, as a result, is to attain sufficient confidence in the proposed mechanism, which thereby proves its promising potential in delivering quality solutions. Table 5.1 shows the results of the repeated runs.

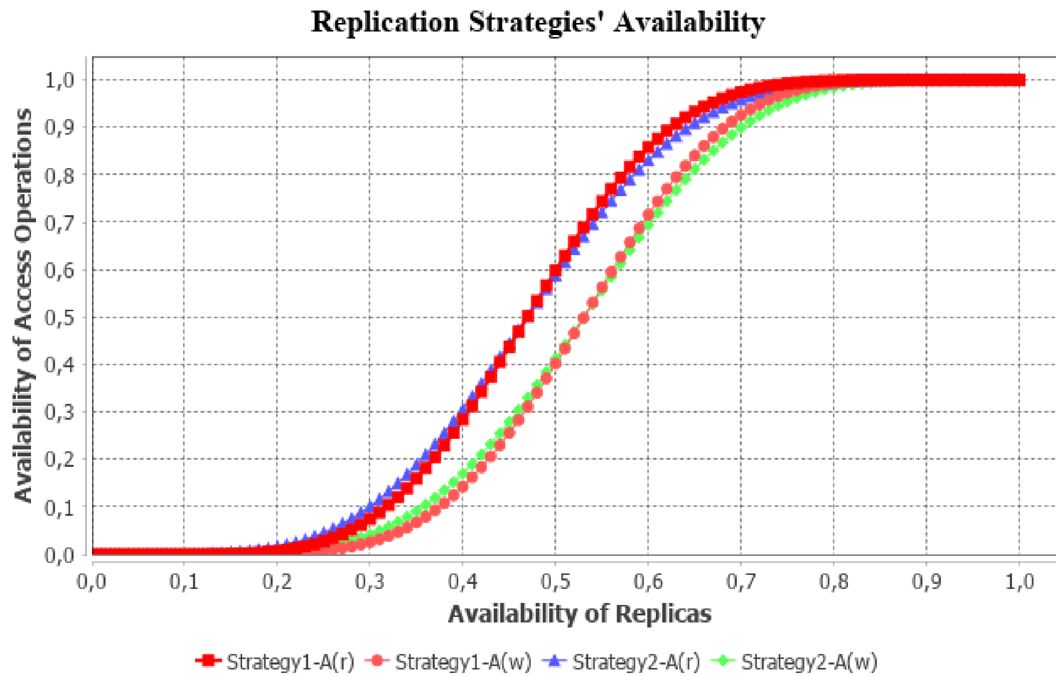

FIGURE 5.14: Availability, MCS vs. hybrid DRS (16 replicas)

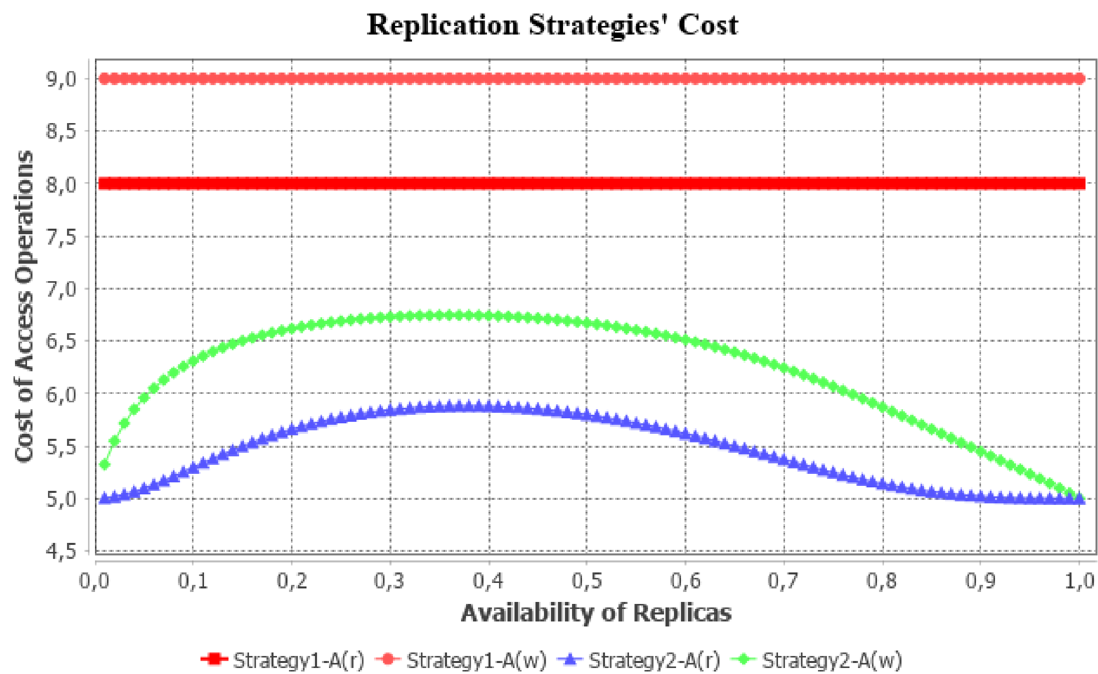

FIGURE 5.15: Cost, MCS vs. hybrid DRS (16 replicas)

5.2.3 System parameter settings for scenario 2

Here, the same scenario is being used, but with a higher expected write availability. The scenario parameters are, hence, as follows.

TABLE 5.1: Results of the GOOGP on repeated runs

Run	Generations	Best fitness	Worst fitness	Avg. fitness	Std. deviation
1	10	1.525	1.366	1.457	0.031
2	8	1.508	1.331	1.456	0.042
3	11	1.504	1.336	1.466	0.044
4	10	1.501	1.365	1.460	0.034
5	5	1.508	1.336	1.436	0.056
Avg.	**8.8**	**1.509**	**1.347**	**1.455**	**0.042**

p = 0.6, ϵ = 16, α = 0.80, β = 0.72, γ = 7.0, δ = 7.0, fw = 1.0

As for the system parameters, the number of parent strategies for each population is set to six and the number of offspring solutions for every generation is restricted to 15. ($\mu + \lambda$) strategy is being used as both parent and offspring strategies are important in solving such replication problems. The mutation should not be so frequent, therefore, being set as 20% while the use of initial solutions in every generation is 30%. The intra-crossover operators are evenly (20% each) distributed for the first three types and 40% for the type 4 operator since we have more room for recombinations here. Syntactically, the way it works is that if the value is 0.2 or equal, it executes the first operator and if the value is greater than 0.2, it moves on to the next operator, for which the value is 0.4 or equal to execute this operator, and with a value greater than 0.4 comes the next inline operator and so on. The intra-mutation probability is evenly distributed since restricting the structure to limited nodes is as much needed as changes in the voting structure attributes.

μ = 6, λ = 15, mutation Prob = 0.2,
initPopListProb = 0.3,
intraMutationProbs = <0.5, 0.5>,
intraCrossoverProbs = <0.2, 0.4, 0.6, 1.0>

5.2.4 Results for scenario 2

The algorithm is run, having set μ and λ to six and 15 respectively, along with a mutation probability of 0.2. Figure 5.16 depicts a 3D representation of generated solutions for the specified problem. The x-axis shows the availability while the y-axis represents the cost of the access operations. In addition, each strategy is assigned a unique color for uniquely identifying the strategies. This view overtly displays the trade-offs between the objectives of newly generated replication strategies to cover the potential scenarios. Segregating the view into four equal quadrants; quadrant 1 (top right corner) indicates better availabilities at the expense of costs, quadrant 2 (top left corner) shows that availabilities and costs are both worse, quadrant 3 (bottom left corner) represents better costs at the expense of lower availabilities, and quadrant 4 (bottom right corner) offers solutions which are better in both, availabilities and costs. The fourth quadrant is more important in general; however, it depends on the scenario. In this case, an appropriate solution (circled in red) satisfying the criteria, is picked at run-time.

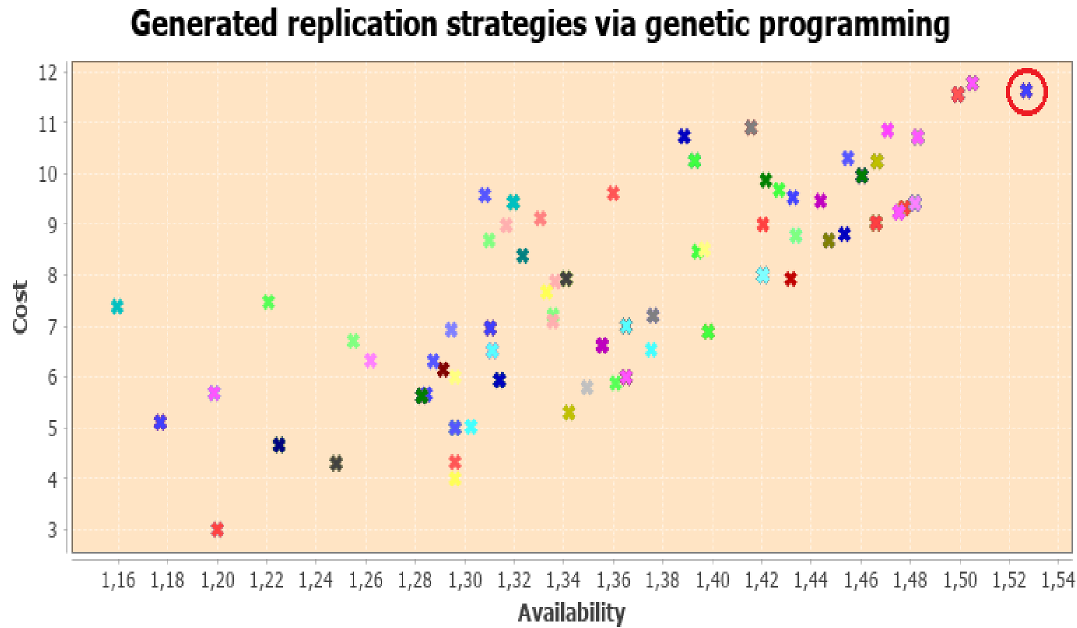

FIGURE 5.16: DRS generated via genetic programming

Figure 5.17 illustrates the same view as a Pareto front, being comprised of non-dominated solutions only, for the given scenario. A solution is considered non-dominated if none of the objectives can be optimized without degrading some of the values of other objectives. It can easily be analyzed, and the solutions of the choice can be picked among their trade-offs between availabilities and costs. The Pareto front shows some of the solution DRSs getting closer to availability of 1.54 for both the access operations. Moreover, it can also be noticed that some of the strategies are quite economical in terms of their cost. For the specified scenario, the system takes 12 generations to come up with an optimized solution. Considering the trade-offs, a strategy adhering to the specified properties can easily be chosen at run-time.

Figure 5.18 shows the fitness of every individual and their evolution, the x-axis is the number of DRSs (each by a unique ID) and y-axis represents the fitness value of every individual DRS. The red line shows the fitness of the replication strategies, whereas the pink and blue lines denote the availabilities of read and write operations, respectively. Here, it can be seen, the strategies start with lower fitness, which means that the database repository does not have adequate solutions to the specified problem. New solutions are generated using the crossover and mutation operators of genetic programming. These solutions evolve gradually through the genetic operators until the loop stops over the intended termination condition.

Figure 5.19 shows the improvements in fitness by every generation and chooses the strategies of the best fitness among each generation. A constant evolutionary trajectory of DRSs can be noticed here. It takes 12 generations for the system to find an adequate DRS satisfying the specified scenario. It starts with the strategies of fitness, i.e., 1.365, which gradually but regularly continues to evolve until the required fitness 1.526 is achieved.

Figure 5.20 shows the identified suitable strategy (up-to-now unknown) optimized for the mentioned scenario. This strategy comprises 16 replicas, which fulfill the threshold

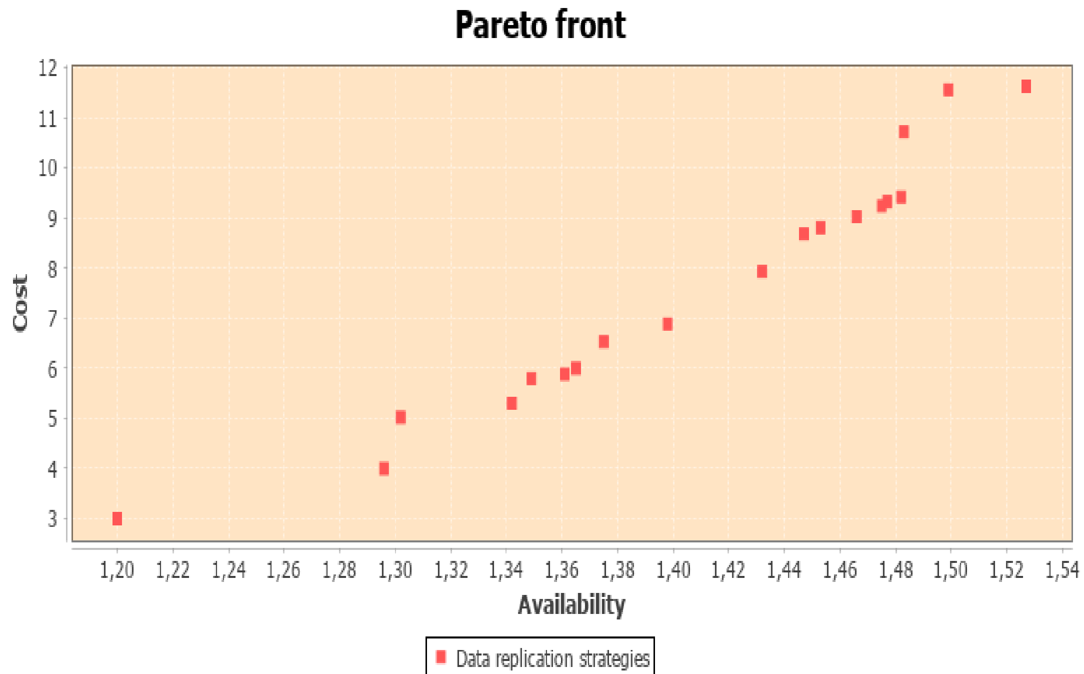

FIGURE 5.17: Pareto front

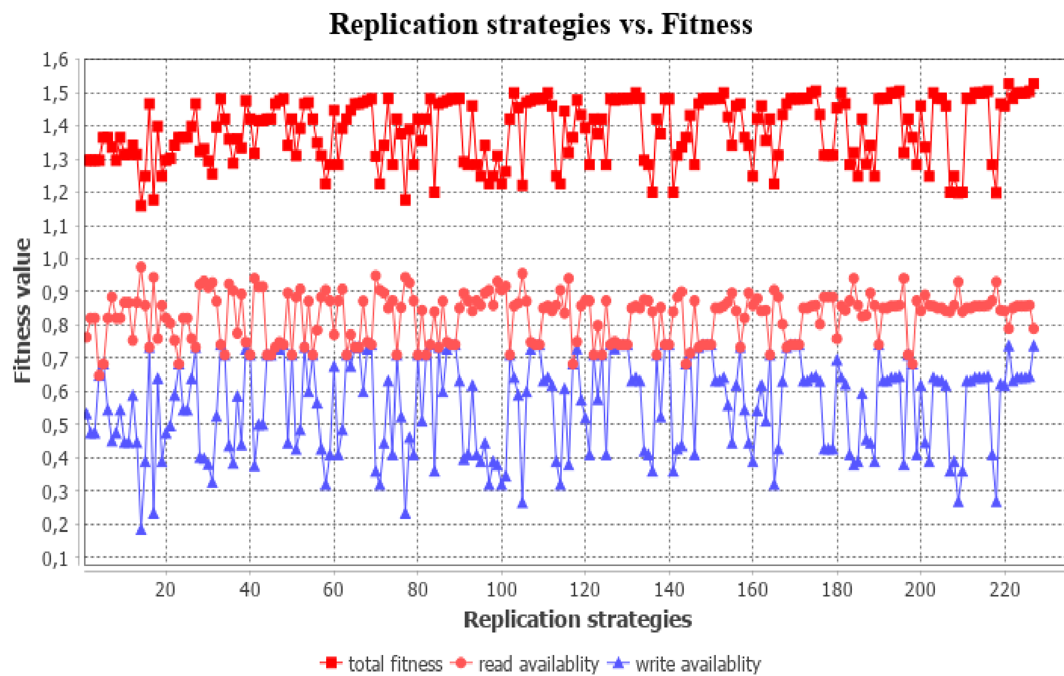

FIGURE 5.18: Scenario, fitness availability analysis

criteria. Furthermore, the varied structures and quorums indicate its hybrid nature, working collectively to serve the purpose and fulfill the given scenario. This new hybrid replication strategy only takes five replicas each to perform an access operation in the best cases, which is very economical.

Figure 5.21 presents the availability graph for the access operations of the newly

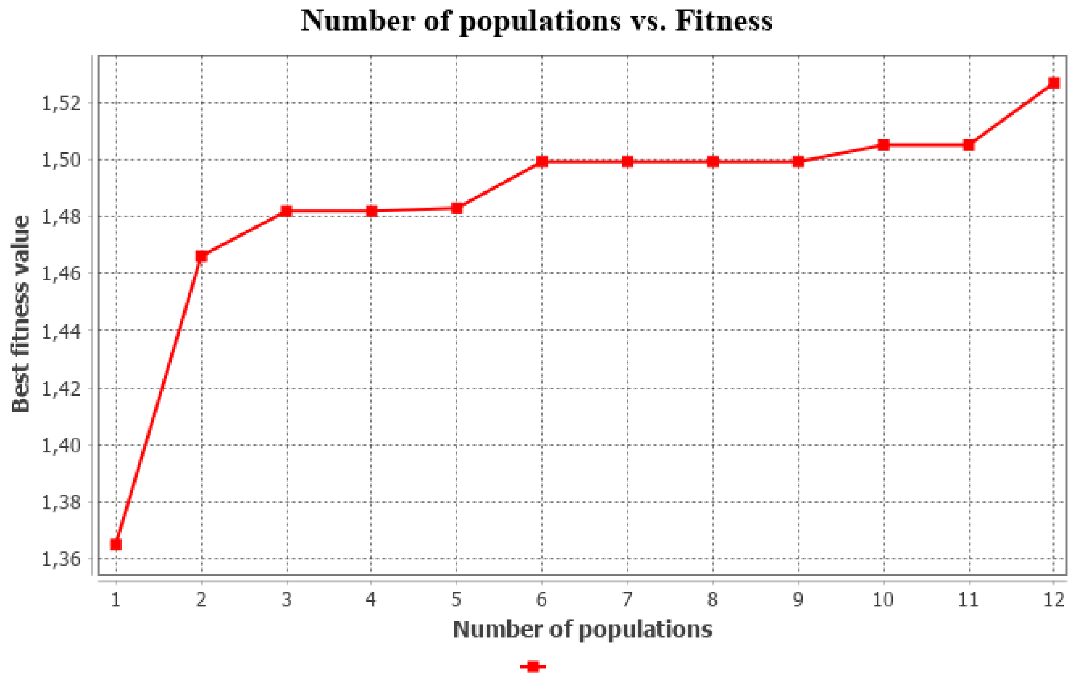

FIGURE 5.19: Scenario, populations' analysis

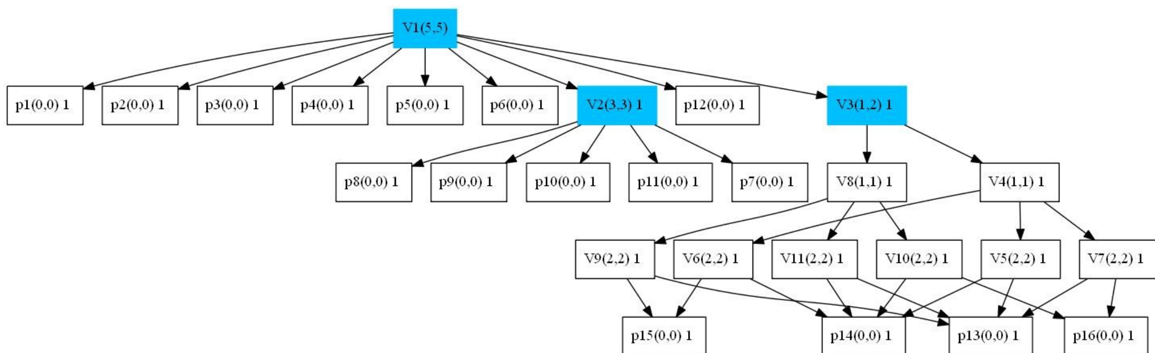

FIGURE 5.20: Scenario, optimized hybrid DRS

designed DRS on discretized values of p. The discovered DRS is optimized for the given scenario. The x-axis indicates the replica availability, whereas the y-axis represents the availability of the access operations. The point-symmetry, as well as the sharp curve of the graphs, overtly display extremely high availabilities for both the access operations, particularly, for the write availability, which is more critical.

Figure 5.22 shows a comparison between MCS and the discovered hybrid DRS. As mentioned before, MCS is considered to be the best in its availability, particularly for the critical write availability. Red and pink lines depict the availabilities of read and write operations, respectively, for MCS. Blue and green lines show the availabilities of read and write operations, respectively, for the hybrid strategy. Again, the sharp curve of graphs reveals very good operation availabilities. It is evident here that the proposed approach fairly competes with MCS and operation availabilities are extremely close, particularly, for the later p values, which are more important.

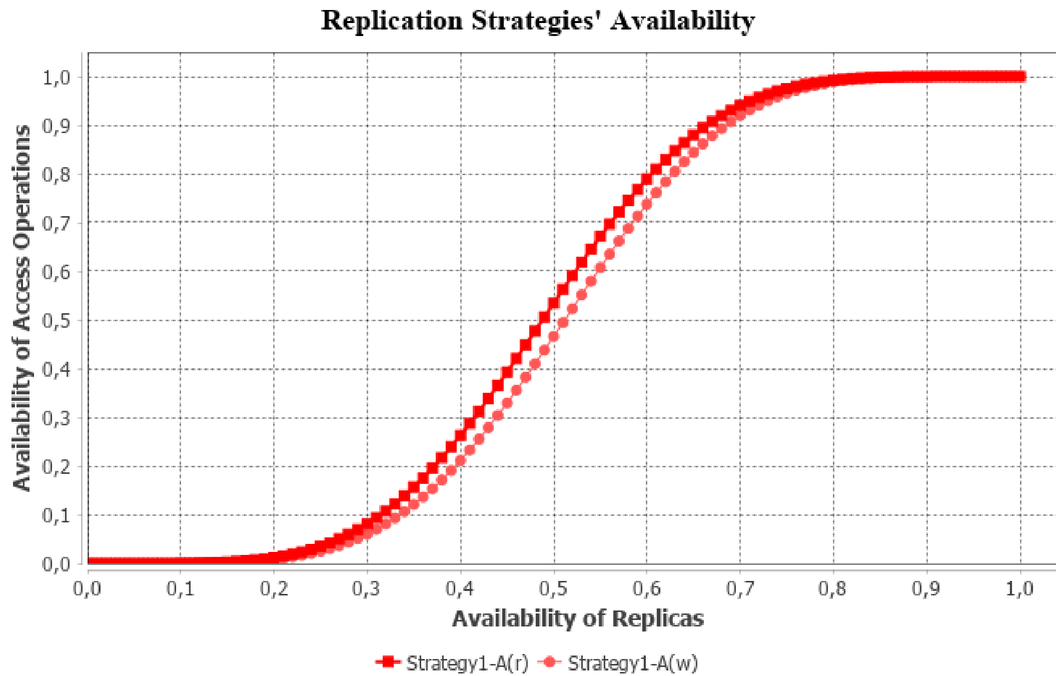

FIGURE 5.21: Availability of the chosen optimized DRS

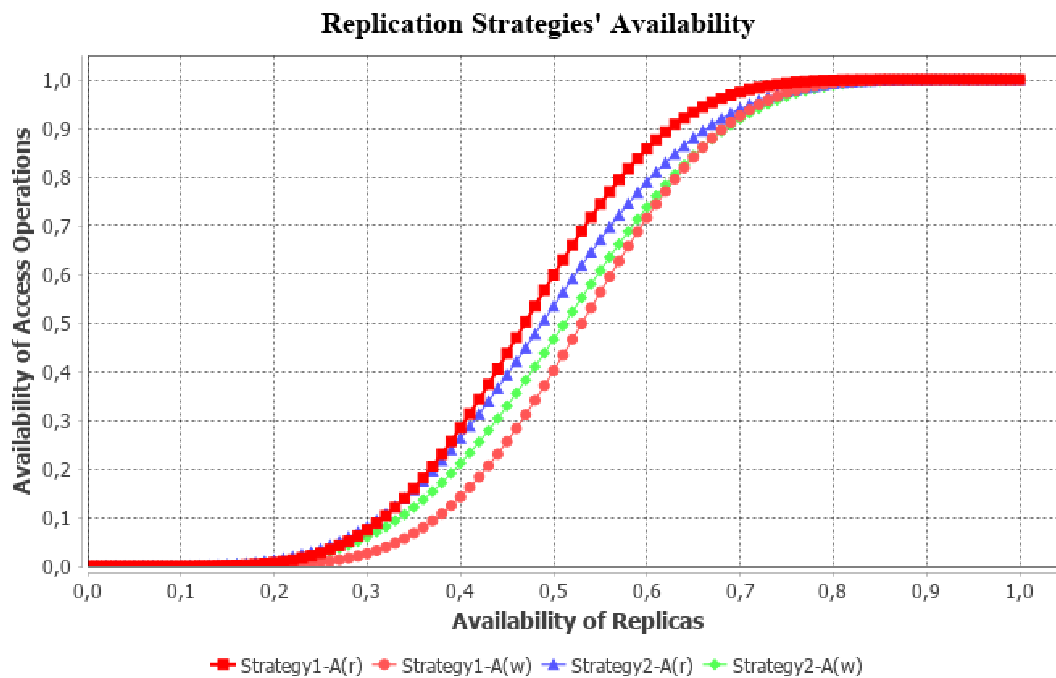

FIGURE 5.22: Availability, MCS vs. hybrid DRS (16 replicas)

Figure 5.23 represents a zoomed-in view of the operation availabilities on higher p values. It can be noticed here that operation availabilities are very close, intermingling, and converge onto almost the same values for later values of p, which is a very good operation availability considering the strong hardware of today.

Figure 5.24 shows the cost comparison between the two mentioned strategies. Blue and

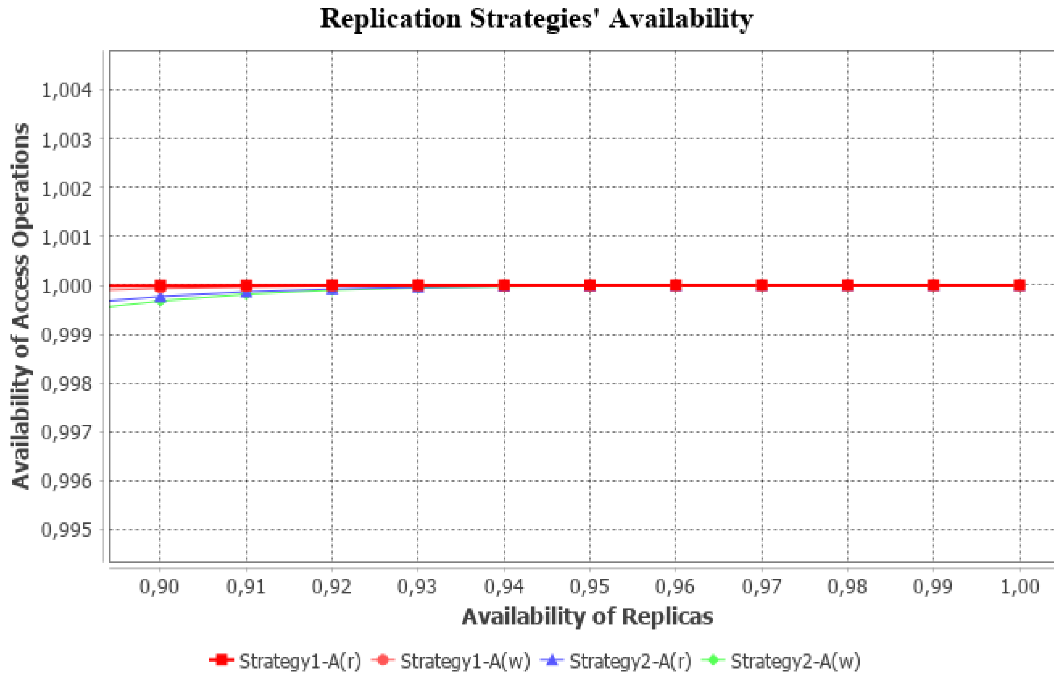

FIGURE 5.23: Zoom-in view of the respective availabilities

green lines indicate the costs of read and write operations, respectively, for the hybrid DRS. It can be noticed that despite fairly competing with MCS in operation availabilities, the hybrid replication strategy is very cheap in its cost, too. It only takes five replicas each for the best case (and at max. six replicas) to execute an access operation, whereas MCS takes a constant of eight replicas for the read and nine replicas for the write operation, respectively. Thus, the proposed approach has significantly reduced the cost and at the same time not sacrificed too much of the availabilities either. Moreover, when compared, the hierarchical strategy (of 16 replicas) outclasses this newly discovered strategy in its read availability, but performs worse in the write availability; however, write availability is more critical to be achieved. Again, the proposed approach, in this case, provides a very good write availability at an economical cost.

Next, a more challenging scenario is specified. This example, subsequently, shows the importance of crossover points, thereby impacting the trade-offs of quality metrics.

5.2.5 System parameter settings for scenario 3

In this example, the availability of the replicas is set to 0.7 while read and write availabilities are set to 0.9 for each operation, inside a total cost of eight for the access operations. The availability is more important than the cost in this scenario, therefore, a weightage of 70% is given to availability and the rest to the cost. These objectives have to be achieved by no more than 16 replicas.

$p = 0.7, \epsilon = 16, \alpha = 0.90, \beta = 0.90, \gamma = 4.0, \delta = 4.0, fw = 0.7$

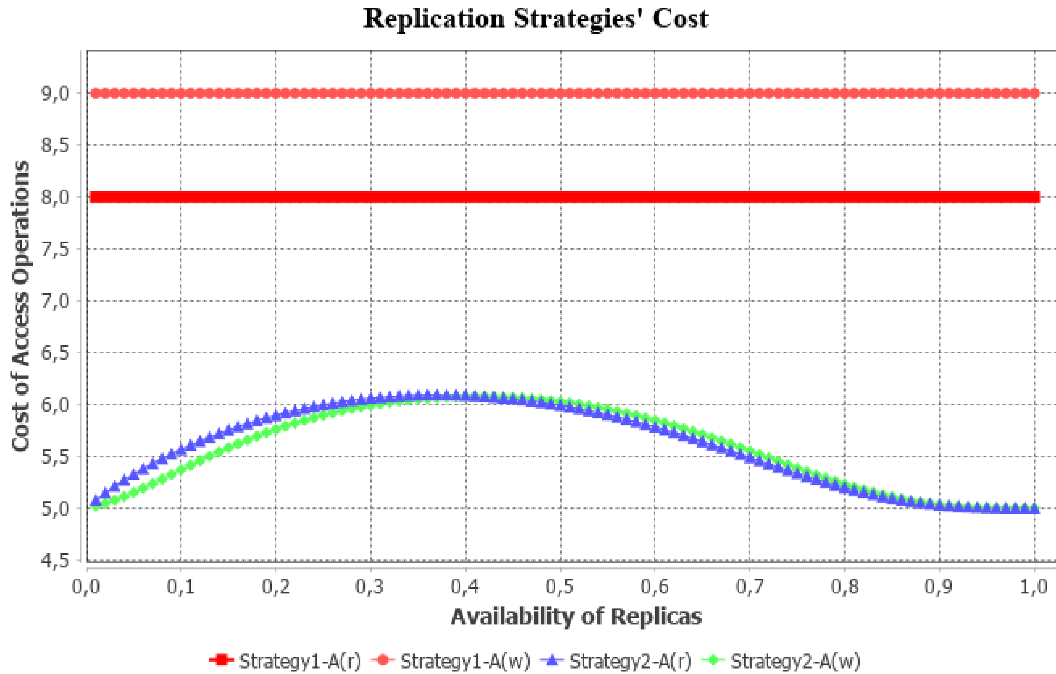

FIGURE 5.24: Cost, MCS vs. hybrid DRS (16 replicas)

Having defined the scenario, now the system parameters are set to run the algorithm accordingly. The number of parent and offspring strategies are set to six and 15, respectively. The initial population is only used once in the genetic process in the very first generation. The crossovers are performed all the time while the mutation is performed with a probability of 0.2. Hence, having kept the system parameters and to the same values of six and 15, respectively, on a mutation probability of 0.2, the system is run. Also, to keep the mechanism more flexible, the individual thresholds for this scenario are not checked, and the stopping criteria, hence, are entirely based on fitness. Additionally, we only used TLP instances for this scenario as an initial population.

$\mu = 6$, $\lambda = 15$, mutationProb = 0.2

5.2.6 Results for scenario 3

Figure 5.25 illustrates the Pareto front comprised of non-dominated solutions for the given scenario. It can easily be analyzed and the solutions of the choice can be picked among their trade-offs between availabilities and costs. Here, each strategy is assigned a unique color to further ease up the decision-making. We have some DRSs in the fourth quadrant (bottom right corner) indicating significantly good solutions concerning both objectives.

The Pareto front for scenario 3 shows some of the solution DRSs getting closer to the availability of 1.8 of both the access operations. Moreover, it can also be noticed that some of the strategies are quite economical in terms of their cost, even better than the expected values. For the specified scenario, the system takes three generations to come up with an optimized solution. Considering the trade-offs, Figure 5.26 represents the

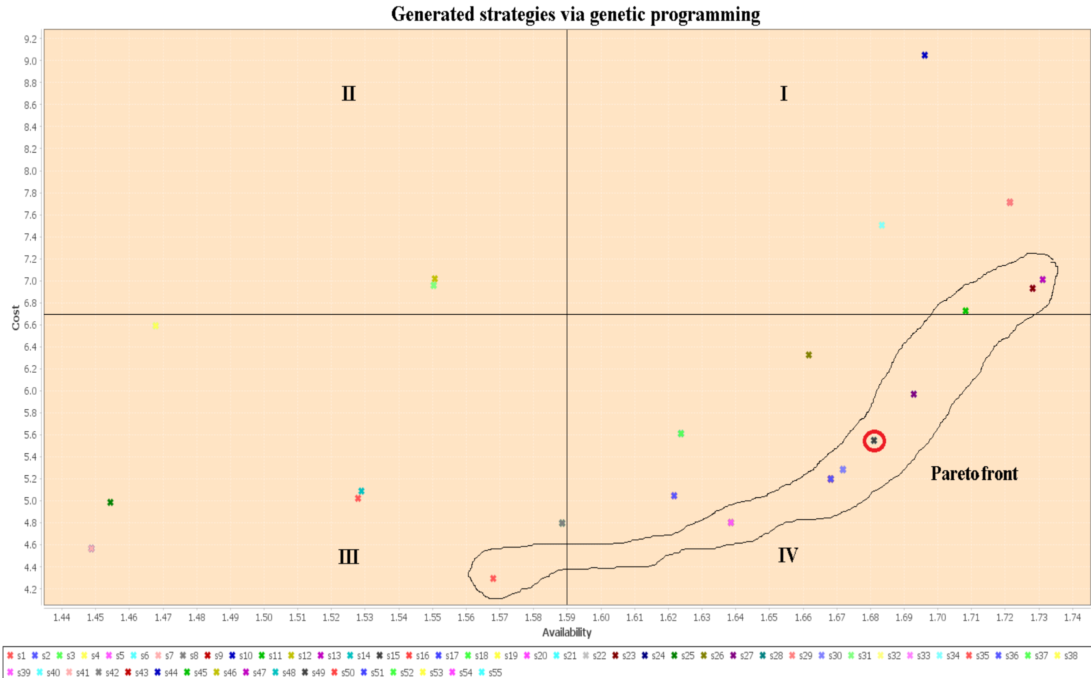

FIGURE 5.25: Pareto front view for scenario 3

chosen hybrid DRS comprising 14 replicas in total, which is better than the specified threshold of 16 replicas. The chosen strategy (circled red in the Pareto front) constitutes several atomic substructures of the Triangular Lattice Protocol and has a fitness of 1.934, which is better than the desired value of 1.86.

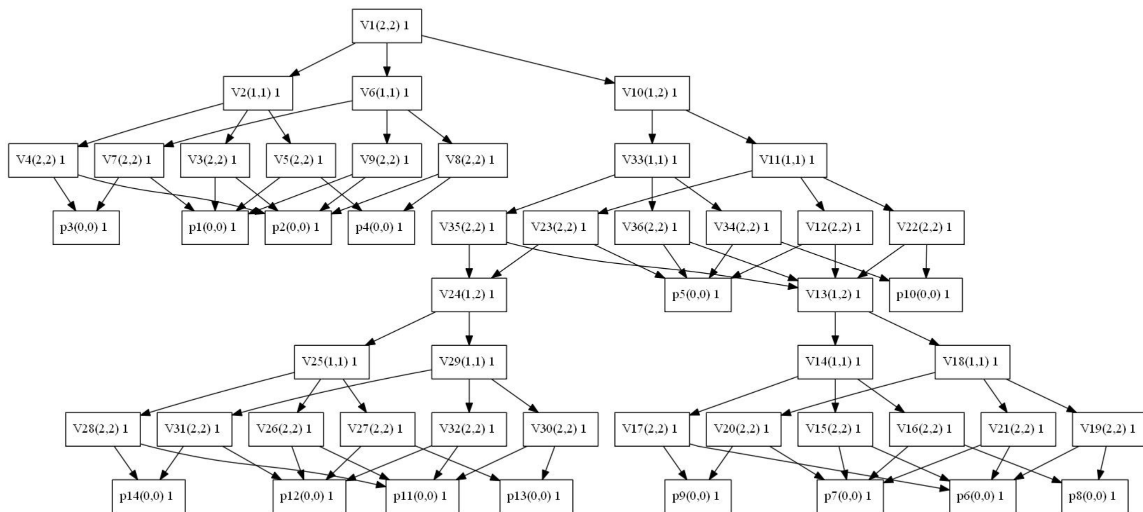

FIGURE 5.26: An optimized DRS for scenario 3

Figure 5.27 shows the availability graph of the generated new hybrid DRS on the discretized values of p. The x-axis represents the availability of replicas while the y-axis represents the availability of the access operations. The red line (with squares) shows the availability of the read operation while the pink dotted line indicates the availability of the write operation. It can be seen that the availabilities are good, too, but most importantly, the costs of the access operations are noticeably low.

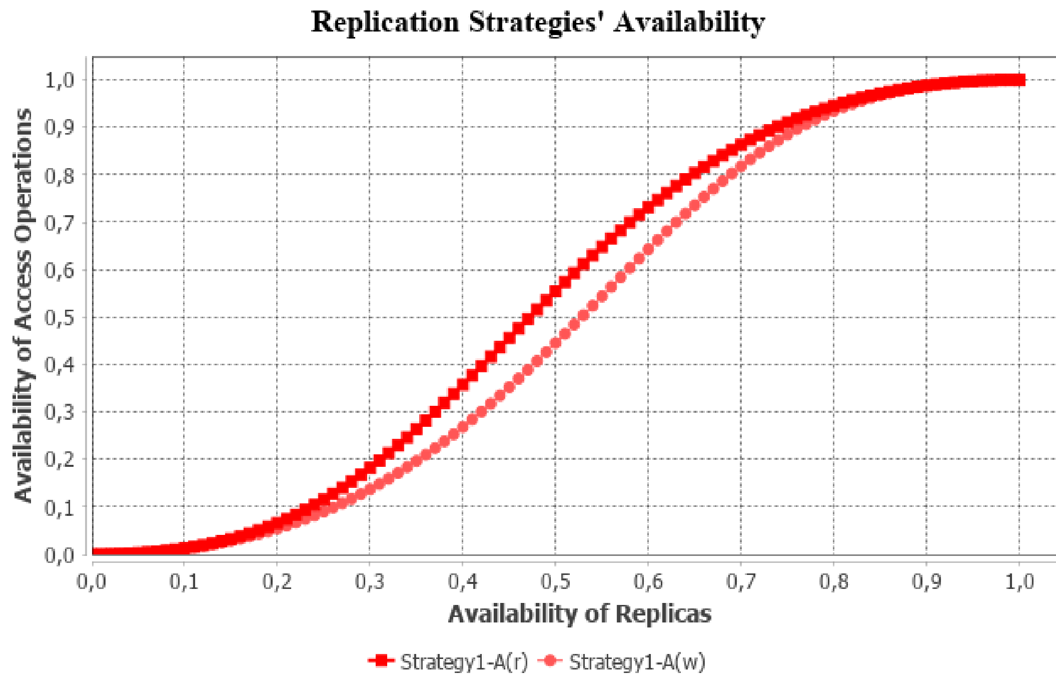

FIGURE 5.27: Availability of the mentioned DRS

Figure 5.28 represents the cost on the discretized values of p. The access operations for the chosen DRS are very cheap where, in the best cases, it only takes two replicas each to perform an operation out of 14 replicas (which is even cheaper than the TLP). Even in the worst cases, the cost remains closer to three replicas each, which is very cheap while not sacrificing too much on the availabilities either.

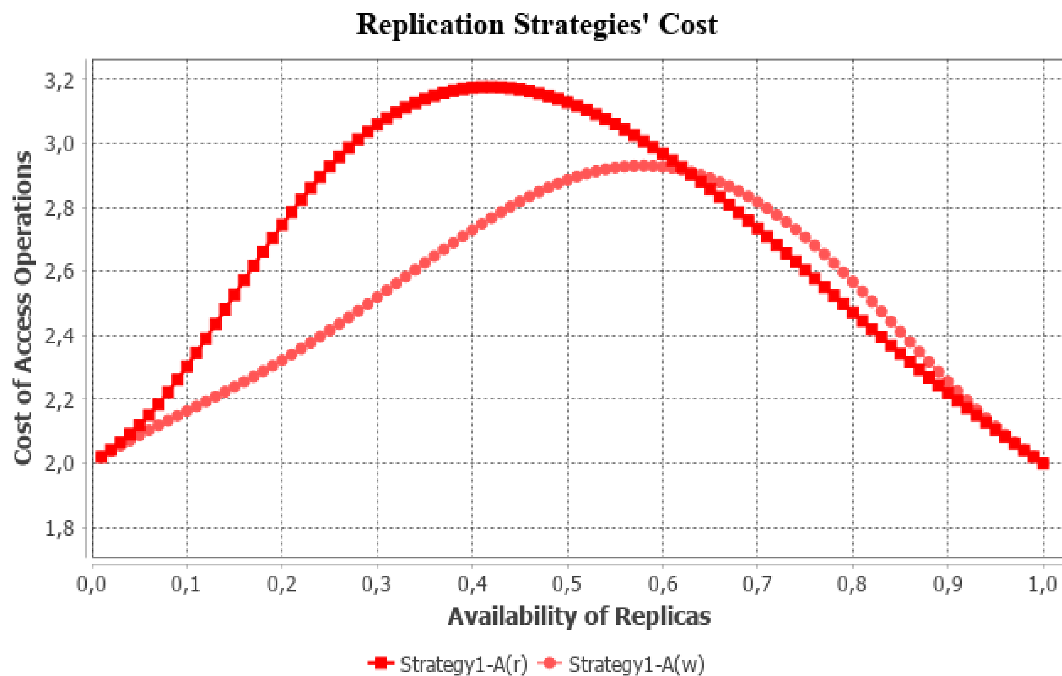

FIGURE 5.28: Cost of the mentioned DRS

The crossover points for DRSs do matter and affect the values of objectives. Another possible solution from the Pareto front is shown in Figure 5.29, where the same building blocks are combined by the GP, but slightly different than in 5.26. However, it has significantly increased the availability of the access operations by slightly compromising on the cost, but not being too heavily either.

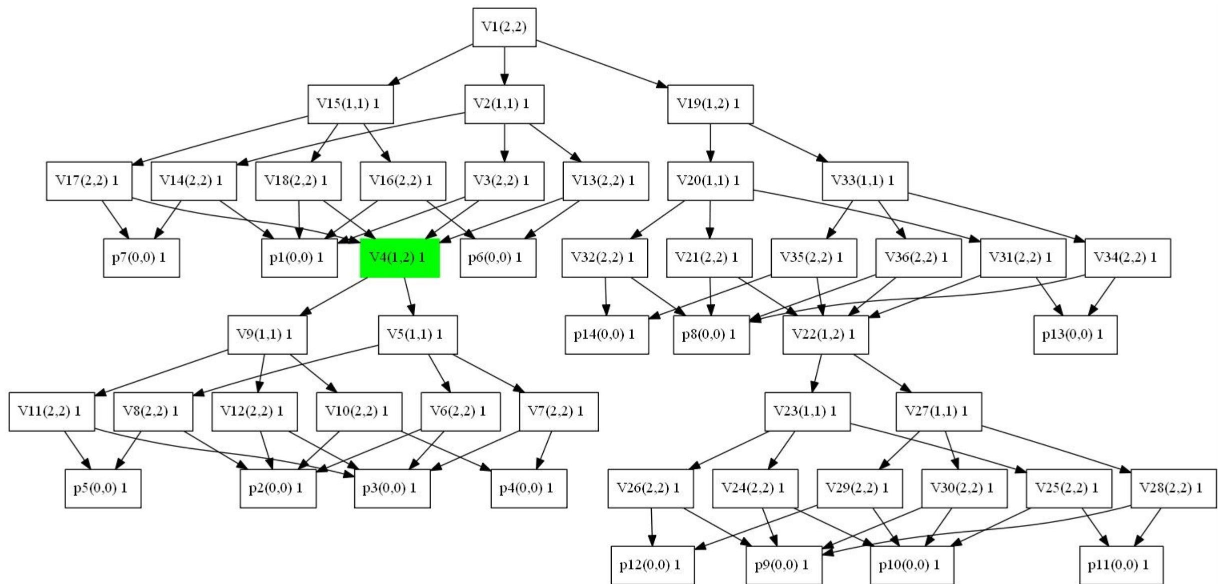

FIGURE 5.29: Another optimized DRS with a slightly different crossover point

Figure 5.30 presents a zoom-in view of the operation availabilities on higher values of p. The comparison graphs indicate that this slight change in the structure of the strategy has resulted in a significant increase in both availabilities of read and write operations of the latter DRS (Figure 5.29). It can be noticed that the availability difference is prominent because of this slight change in the structure of the hybrid strategy. It has resulted in a different outcome of relatively higher operation availabilities.

Figure 5.31 shows the difference between the costs of the two Pareto solutions on the given discretized values of p. The latter strategy with higher availabilities of access operations has compromised on the cost by one (which again indicates that both cannot be achieved at the same time), where, for the best case, it generates costs of three replicas for a read as well as for a write operation.

Hence, this proposed machine learning mechanism efficiently combines replication strategies as a single voting structure to achieve the desired fitness through genetic programming. Similarly, any realistic scenario depending upon the requirements or nature of an application can be defined and the system can accordingly generate formerly unknown solutions through genetic programming by overtly displaying their trade-offs to analyze them and make decisions dynamically at run-time. This automatic mechanism is strong enough to discover new replication strategies that cannot be easily found manually, considering the very huge search space.

Next, the parameters of Scenario 2 are used, which differ from Scenario 1 in the write availability up to two decimal places. As described earlier, even this slight change to some decimal places impacts the availability greatly in real-time, [20]. So, these slight

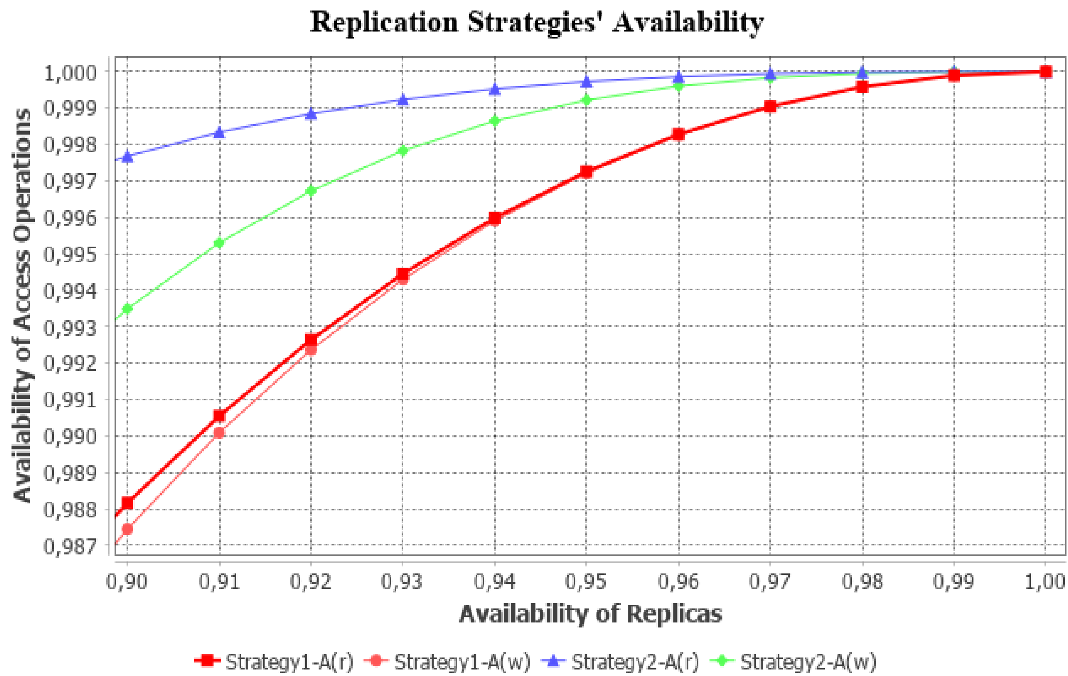

FIGURE 5.30: Availability comparison of the two Pareto front solutions

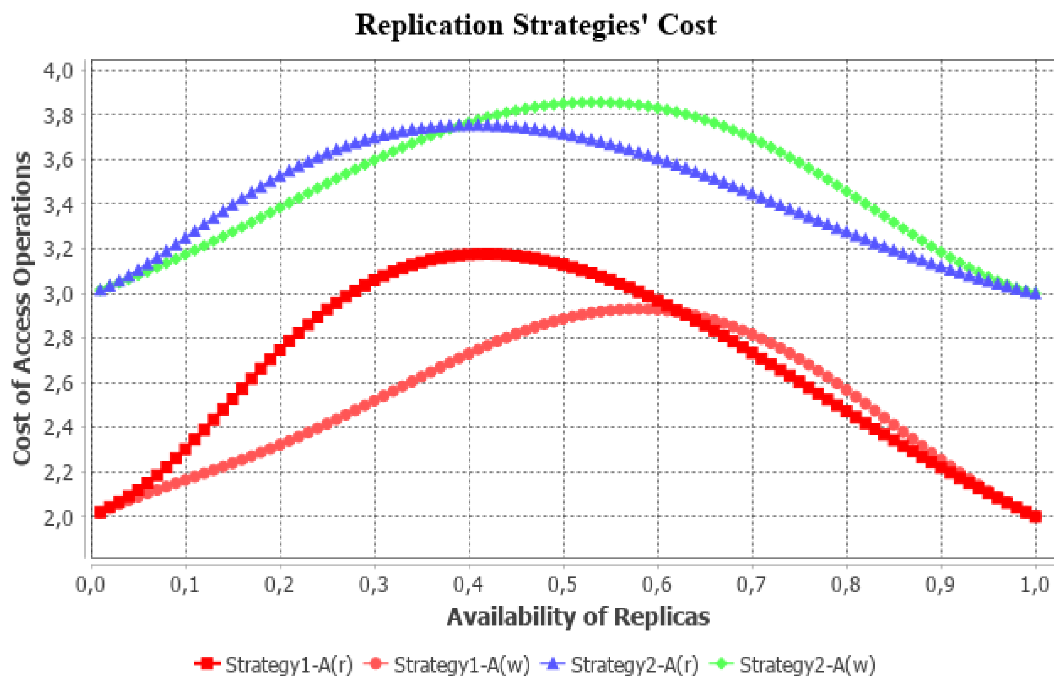

FIGURE 5.31: Cost comparison of the two Pareto front solutions

changes can make a huge difference, therefore, are hard to achieve. In this regard, the next examples demonstrate the flexibility and ease of the proposed approach in generating replication strategies, even with a slight use of the relevant genetic operators for the given constraints.

5.2.7 System parameter settings for scenario 4

As mentioned earlier, here, we would like to use the same scenario parameters as given scenario 2, but with slightly different system parameters, i.e., different mutation and intra-mutation probabilities. The scenario parameters are as follows, which boil down to the fitness of 1.520 to be achieved. Again, the strategy is compared on all the discretized values of p for the realistic approach.

p = 0.6, ϵ = 16, α = 0.80, β = 0.72, γ = 7.0, δ = 7.0, fw = 1.0

For the system to run, the number of parent strategies for each population is set to six and the number of child solutions for every generation is restricted to 15, with a mutation probability of 0.3 on the offspring strategies. The intra-crossover and intra-mutation probability distributions are also given. ($\mu + \lambda$) is used as both parent and offspring strategies are important in solving such replication problems. The mutation should not be so frequent, therefore, being set as 20% while the use of initial solutions in every generation is 30%. The intra-crossover operators are evenly (20% each) distributed for the first three types and 40% for the type 4 operator since we have more room for recombinations here. The intra-mutation distribution is 40%, 60%, respectively, to put more focus on changing the voting structure attributes than restricting the structure to limited nodes.

μ = 6, λ = 15, mutation Prob = 0.2,
initPopListProb = 0.3,
intraMutationProbs = <0.4, 1.0>,
intraCrossoverProbs = <0.2, 0.4, 06, 1.0>

5.2.8 Results for scenario 4

Figure 5.32 represents the fitness of every individual strategy involved in the genetic process. Here, the x-axis shows the number of replication strategies while the y-axis shows the fitness, as well as the availability of the access operations. The red line represents the fitness of every individual strategy while the pink and blue lines represent the availabilities of the read and write operations, respectively. It starts with a few strategies and designs around 270 new replication strategies. An evolving trend can be noticed here that strategies start from a lower fitness and gradually gains a better fitness level over several generations of evolution to meet the desired criteria eventually.

Figure 5.33 represents a constant evolutionary trajectory of these generations; it plots the best replication strategy out of every generation to analyze how fitness grew over several generations. It starts from fitness of 1.36 and goes through several optimization phases, then eventually stops at the desired fitness of 1.523 (better than the specified 1.520). The y-axis shows the fitness value, whereas the x-axis represents the number of generations. Here, the system takes 14 generations of evolution to achieve the defined criteria.

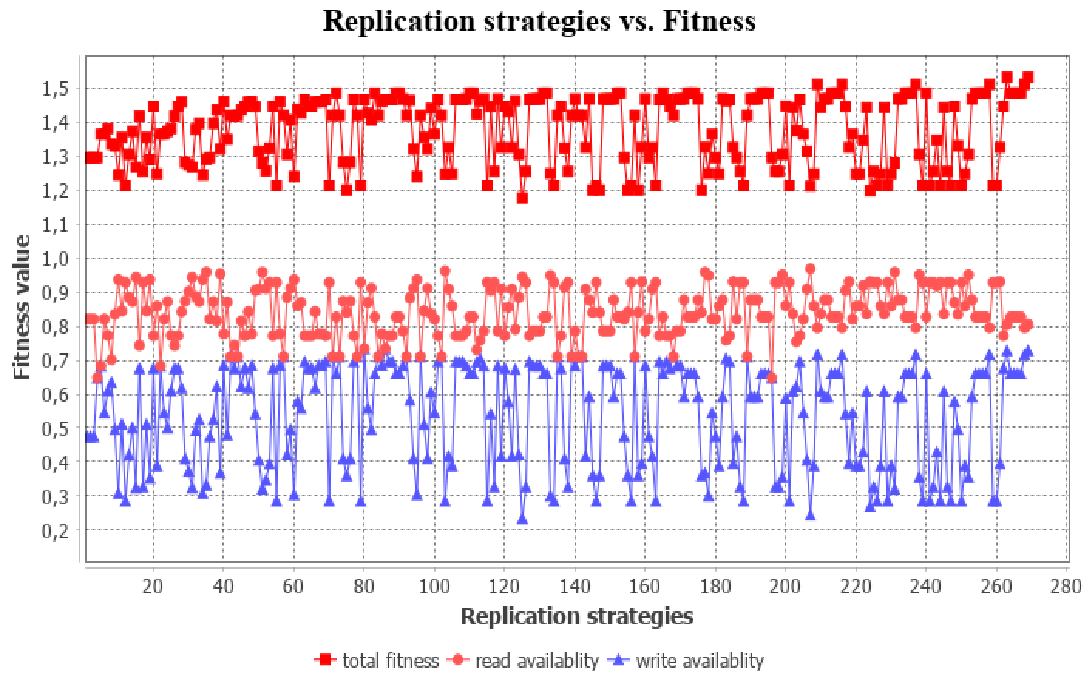

FIGURE 5.32: Fitness availability analysis of the generated DRSs

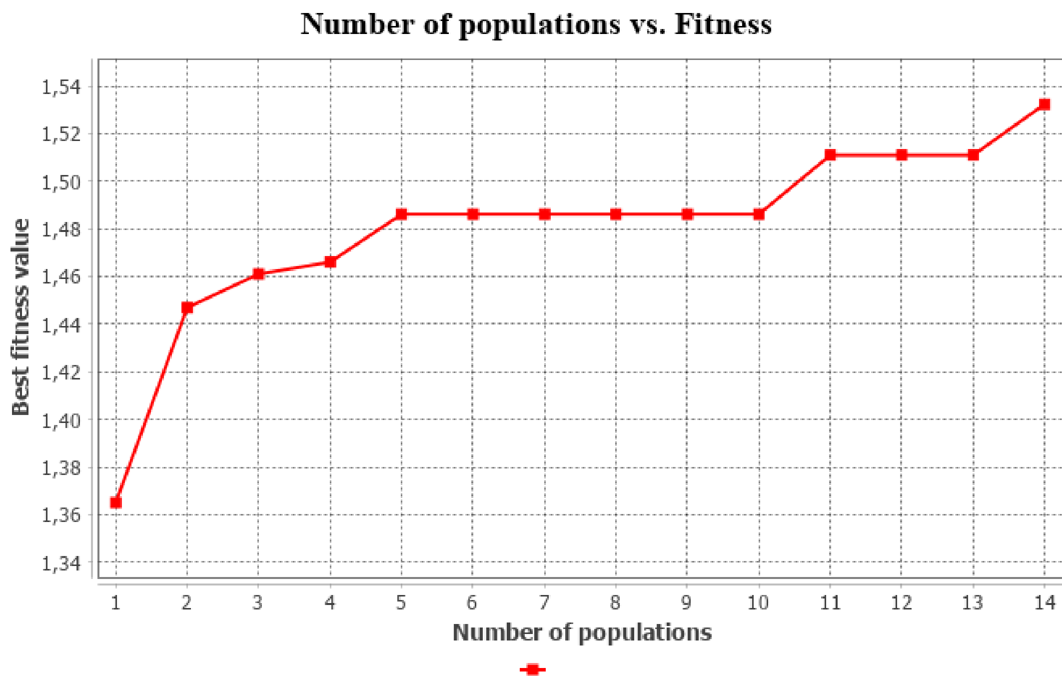

FIGURE 5.33: Populations' analysis

Figure 5.34 depicts operational costs (sum of the read and write costs) of each replication strategy. The x-axis represents replication strategies and the y-axis their respective costs. It starts from lower costs, which increase over time for later generations, but for higher availability values. This implies a higher cost may often result in higher availabilities. The graph starts with the strategies, where there is not much difference

between the total cost and the total number of replicas, which gradually fades away for the later strategies that prove the system is optimizing the DRSs in terms of their cost. Because now it takes fewer replicas to execute the access operations out of the total replicas. The algorithm stops over a desired optimized strategy comprised of 16 replicas with a total cost (sum of read and write costs) of almost 12 replicas on given p (at best 10 replicas).

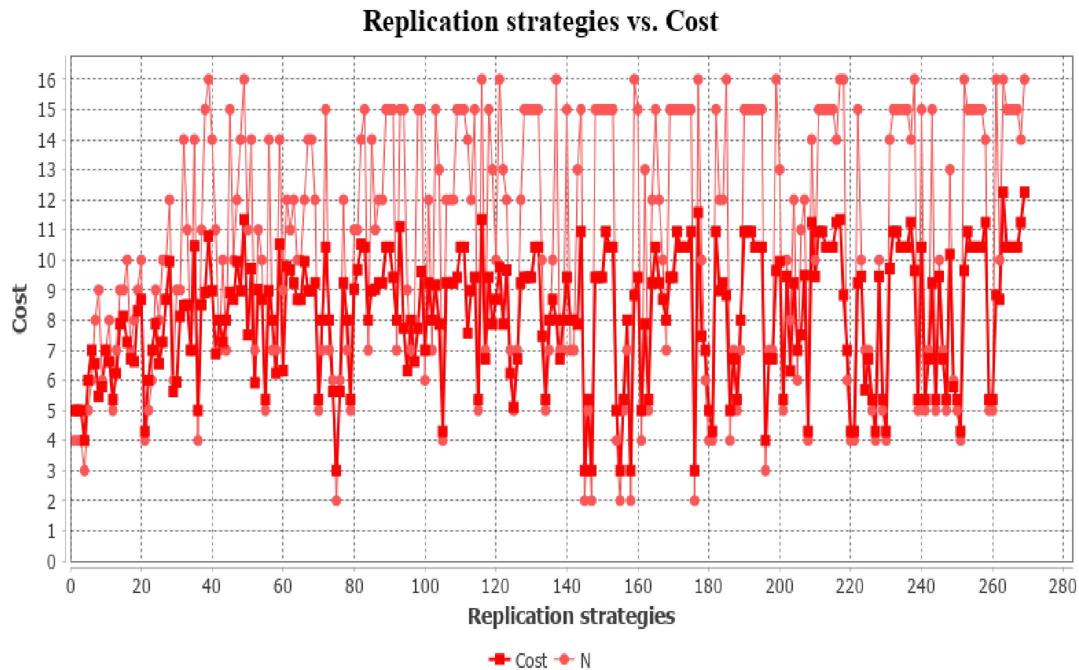

FIGURE 5.34: Costs analysis of the generated DRSs

Figure 5.35 shows the chosen hybrid strategy optimized for the scenario. It can be seen that it is comprised of various substructures of replication strategies including the modified versions of Grid Protocol and Majority Consensus Strategy (MCS) along with their varied structures and quorum sizes, thereby indicating its heterogeneous nature. It blends multiple concepts flexibly to meet the specified criteria, which would definitely not have been easily possible in their orthodox representations. Not only this helps to achieve high availability, but at the same time not too expensive either. Here, it could take merely five replicas to perform an operation, which is very cheap.

Figure 5.36 shows the availability (y-axis) comparison between the famous MCS and the newly discovered hybrid strategy on the discretized values of p (x-axis). These results are very close to MCS in availabilities but far cheaper in costs. The point-symmetry, as well as the sharp curve of the graphs, shows quite good availability values for both the access operations, which proves the effectiveness of this approach. Blue and green lines represent the availabilities of read and write operations, respectively, for the hybrid DRS. It can be noticed that the discovered strategy's availabilities are very close to MCS, particularly, for the later p values for both the access operations.

Figure 5.37 gives a closer look at higher node availabilities p. Again, it can be seen that the availabilities of the hybrid strategy are extremely close and converging onto almost the same values for later p, which is quite good considering the strong hardware nowadays.

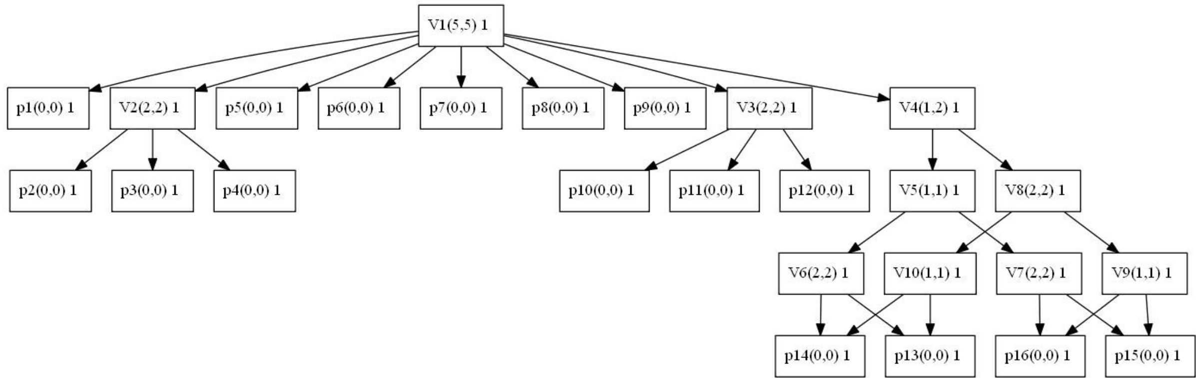

FIGURE 5.35: Discovered optimized hybrid DRS

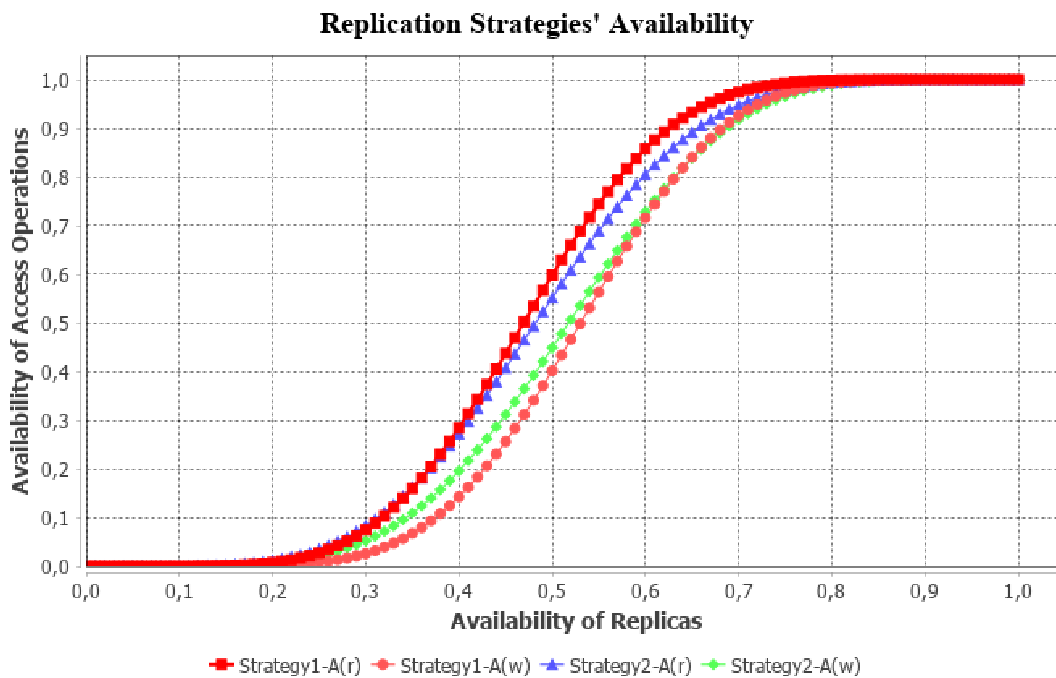

FIGURE 5.36: Availability, MCS vs. hybrid DRS (16 replicas)

Figure 5.38 shows a cost comparison between the two mentioned strategies. It can be seen the latter (hybrid strategy) is very cheap as compared to the former one, in both the access operations, where it only takes five replicas each to execute an operation for the best cases while MCS takes a constant cost of 17 replicas in total. Hence, significantly high availability has been achieved at a reduced cost, too. Again, when compared to ours, a hierarchical strategy of 16 replicas performs far worst in the critical write availability (on a better read availability though).

Hence, the framework proposed in this dissertation is strong and at the same time flexible enough to discover innovative solutions up-to-now the unknown. If an improved read availability is required at an economical cost too, the next example depicts another solution generated by the proposed approach with a higher read availability. Figure 5.39 shows the generated hybrid DRS via genetic programming comprising 16 replicas. This hybrid strategy is also very diverse; however, it includes slightly different

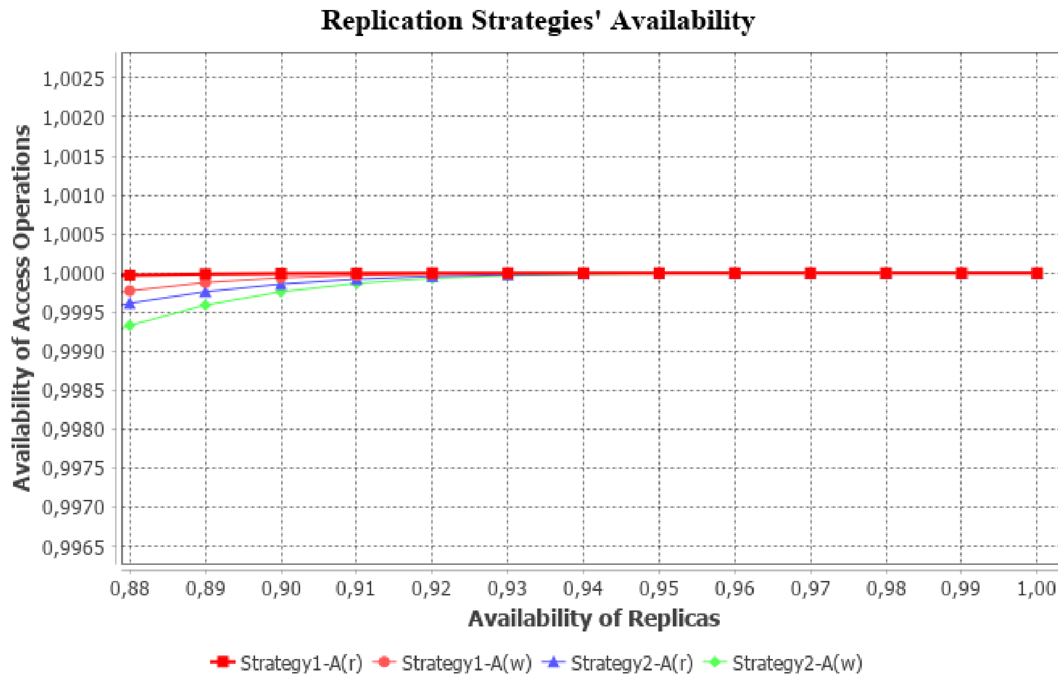

FIGURE 5.37: Zoom-in view of the respective availabilities

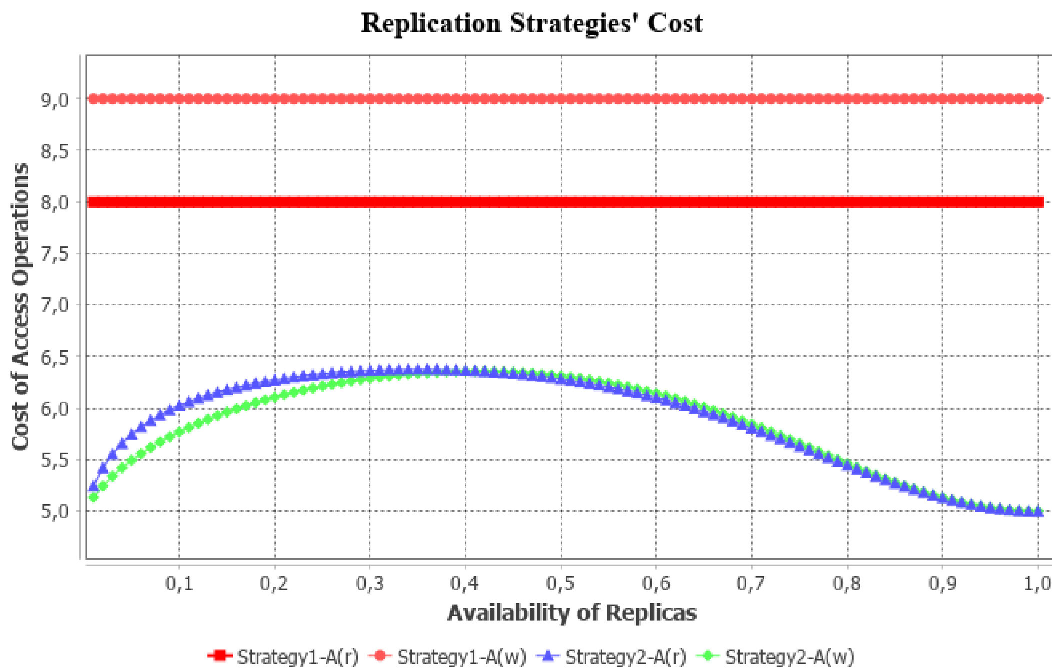

FIGURE 5.38: Cost, MCS vs. hybrid DRS (16 replicas)

substructures and holds uneven votes for some of the nodes (i.e., V2 and V3). Such a combination results in a significant increase in its read operation's availability.

Figure 5.40 shows the availability of this newly discovered strategy along with the other ones. Yellow and pink lines represent the availabilities of read and write operations, respectively, for the most recent strategy. Here, we can see that the availabilities are

FIGURE 5.39: Another optimized hybrid DRS

again extremely close to each other.

FIGURE 5.40: Availability comparison between MCS and other hybrid DRSs

Figure 5.41 represents a zoomed-in view of these availabilities. It can be evidently seen here that the read availability of this latest strategy has been improved. This latter strategy (3rd strategy, given in Figure 5.39) is exceeding the former one (2nd strategy, given in Figure 5.35) in its read availability at the expense of write availability a bit.

Figure 5.42 allows us to take a further closer look at these values (on higher p), which again shows that there is not too much difference between these availabilities. The hybrid strategies are competing fairly with the contemporary MCS.

Figure 5.43 shows the cost of these strategies on discretized replica availabilities. Yellow and pink lines show the operation costs of read and write, respectively, for the 3rd replication strategy (Figure 5.39). It can be seen that the cost is still being restricted to

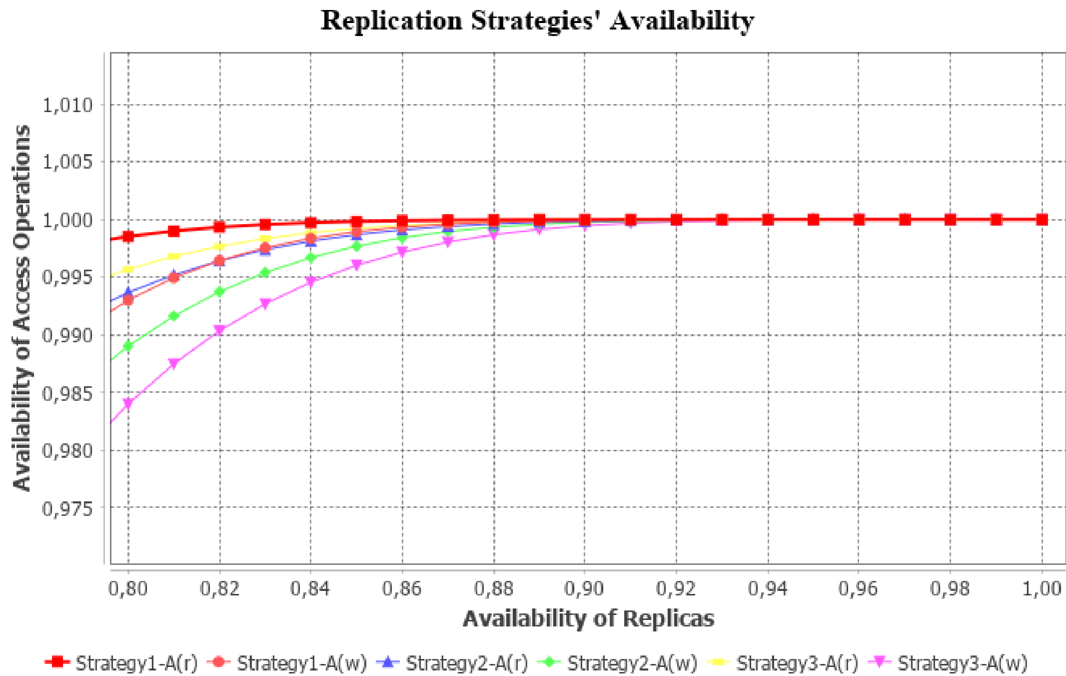

FIGURE 5.41: Availability comparison between MCS and other hybrid DRSs

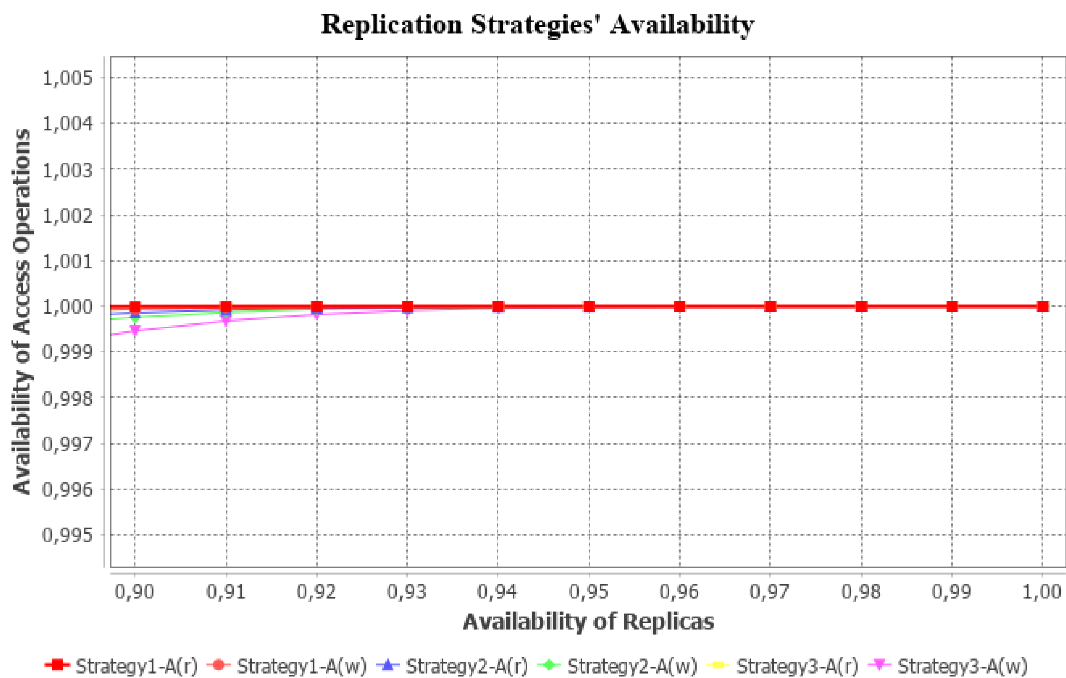

FIGURE 5.42: Availability comparison between MCS and other hybrid DRSs

five replicas for each operation for the best cases (the same as the 2nd strategy, Figure 5.35). However, the read availability has been increased while not being too expensive in the costs. In the same way, we can easily discover new replication strategies satisfying the constraints, automatically.

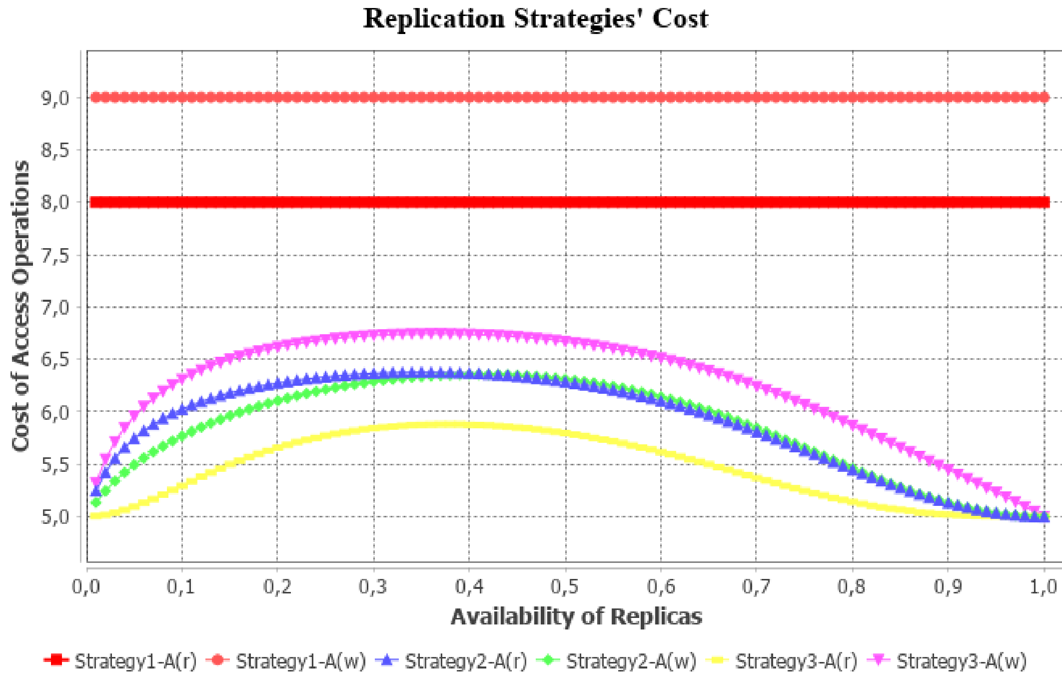

FIGURE 5.43: Cost comparison between MCS and other hybrid DRSs

Even if a further reduced cost is required, i.e., three replicas rather than five (even cheaper than the famous TLP) to cover another prospective scenario. A mutation operation is performed by the proposed genetic operator to alter the votes and quorums, accordingly, to produce the desired outcome. The proposed approach in this dissertation is so flexible and powerful enough to generate consistent (valid) solutions. Having performed the mutation, Figure 5.44 shows the hybrid strategy with modified properties. It can be noticed here that the votes for V2 and V3 (in pink) have been changed (from two to one) while votes for p13 and p14 (in green) have been altered (from one to two). This slight use of the mutation operator has significantly reduced the cost further to a desired lower value.

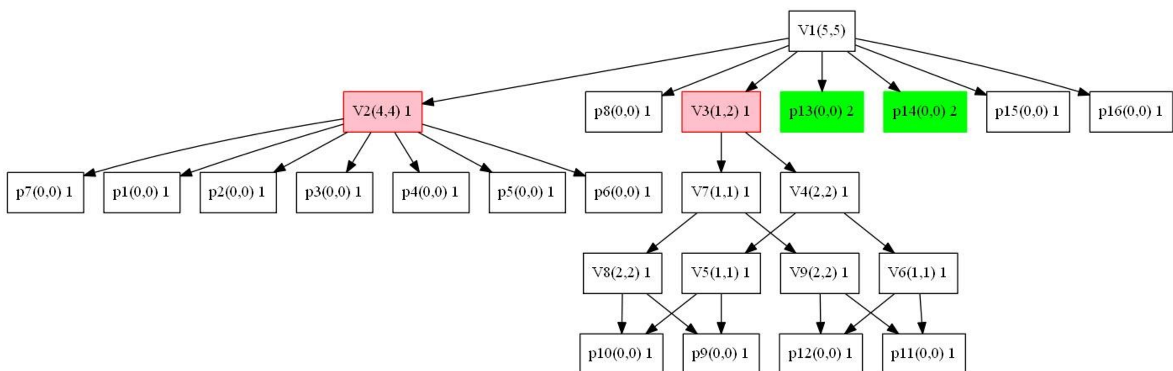

FIGURE 5.44: A mutated hybrid DRS

First, we take a look at the availabilities, Figure 5.45 shows the availabilities of this mutated DRS (Figure 5.44) along with the other three strategies (MCS and hybrid DRSs given in Figure 5.35 and Figure 5.39, respectively). Sky blue and light pink lines

represent the read and write availabilities of the mutated DRS, respectively. Again, the availabilities are very close to each other, intermingling as shown in the figure. It can be noticed here that the mutated replication strategy has slightly comprised on its availabilities (denoted by sky blue and light pink lines), but not too much.

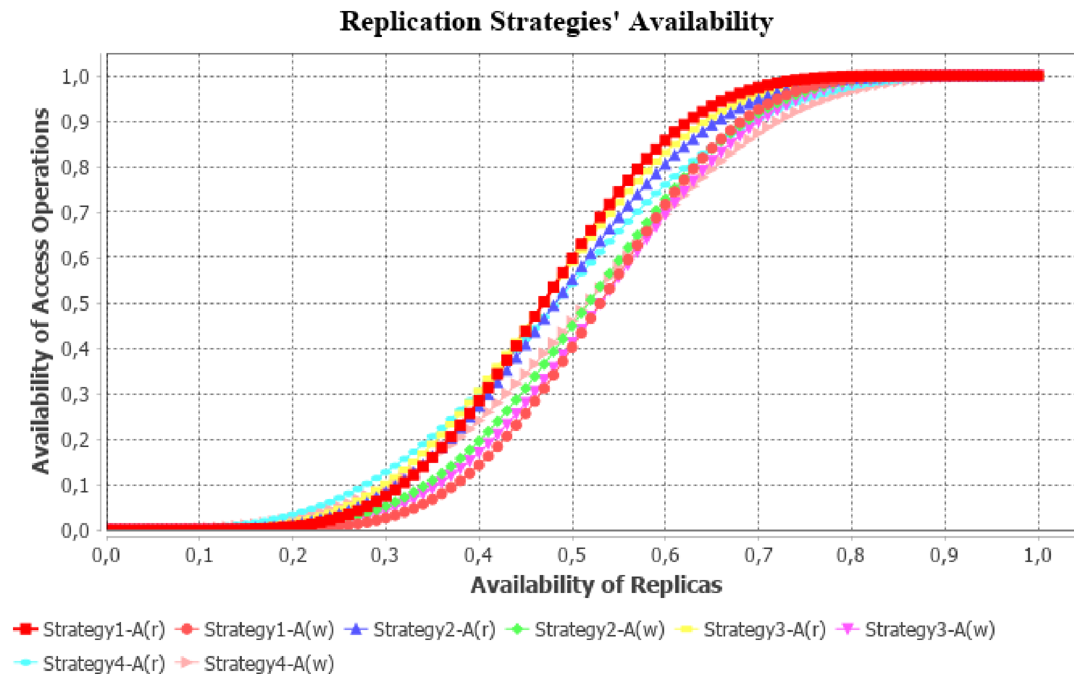

FIGURE 5.45: Availabilities of MCS, hybrid, and mutated DRSs

Figure 5.46 shows a zoomed-in view of the availabilities of this newly discovered strategy along with the other ones. Here, it can be seen that both the availabilities (denoted by Sky blue and light pink lines) of the mutated strategy are compromised a bit. However, on a replica availability higher than 0.9, the difference starts to contract, and the graph indicates that there is not too much difference between these availabilities as they are converging onto almost the same values. Thus, the hybrid strategies are competing fairly with the contemporary MCS.

Figure 5.47 shows a cost comparison of the mentioned replication strategies. Sky blue and light pink lines represent the read and write costs of a mutated replication strategy, respectively. As we see here, this hybrid strategy is very cheap, and only takes three replicas each for the access operations out of a total of 16 replicas, which is even cheaper than the famous TLP. Hence, these results are far cheaper than the famous MCS and TLP while maintaining very good availabilities of the access operations.

Hence, our approach generates new strategies automatically, which otherwise would not be discovered easily. This automatic mechanism based on genetic programming has made the construction of new solutions so easy to accomplish. This approach is so flexible that every strategy can be transformed into this voting structure and all these different concepts can be easily merged to form new solutions while considering the trade-offs between different quality metrics. Genetic programming intelligently designs those replication strategies through multi-crossover and multi-mutation operators, keeping the prospective solutions while discarding the others to constantly improve them in every generation to eventually meet the desired criteria. This avoids

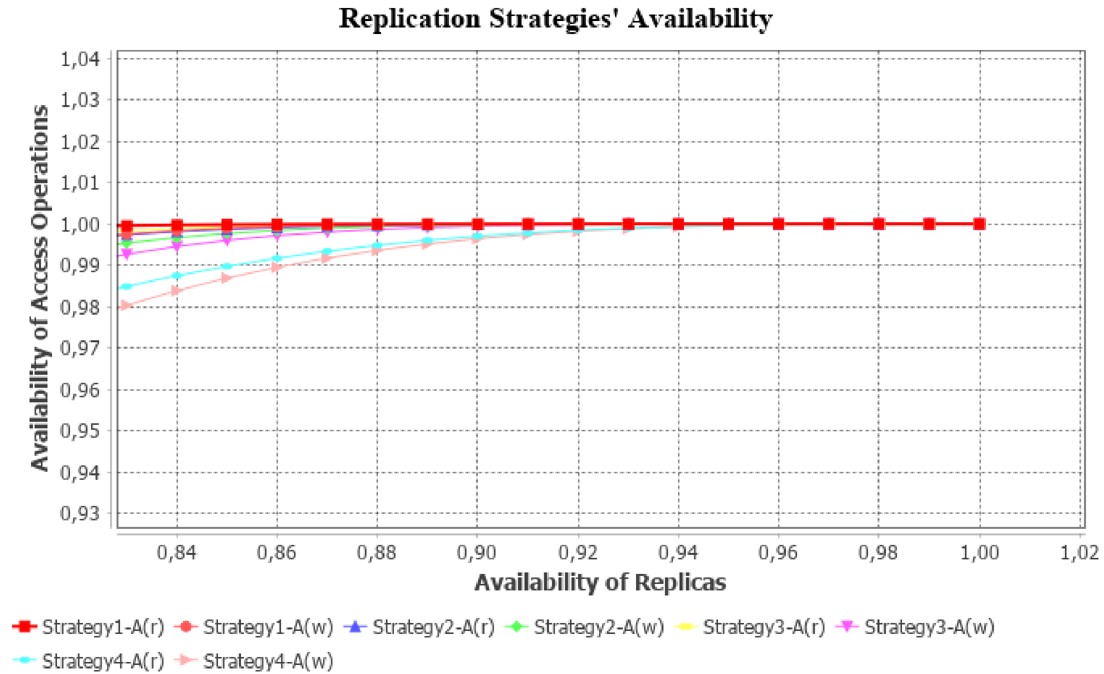

FIGURE 5.46: Availabilities of MCS, hybrid, and mutated DRSs

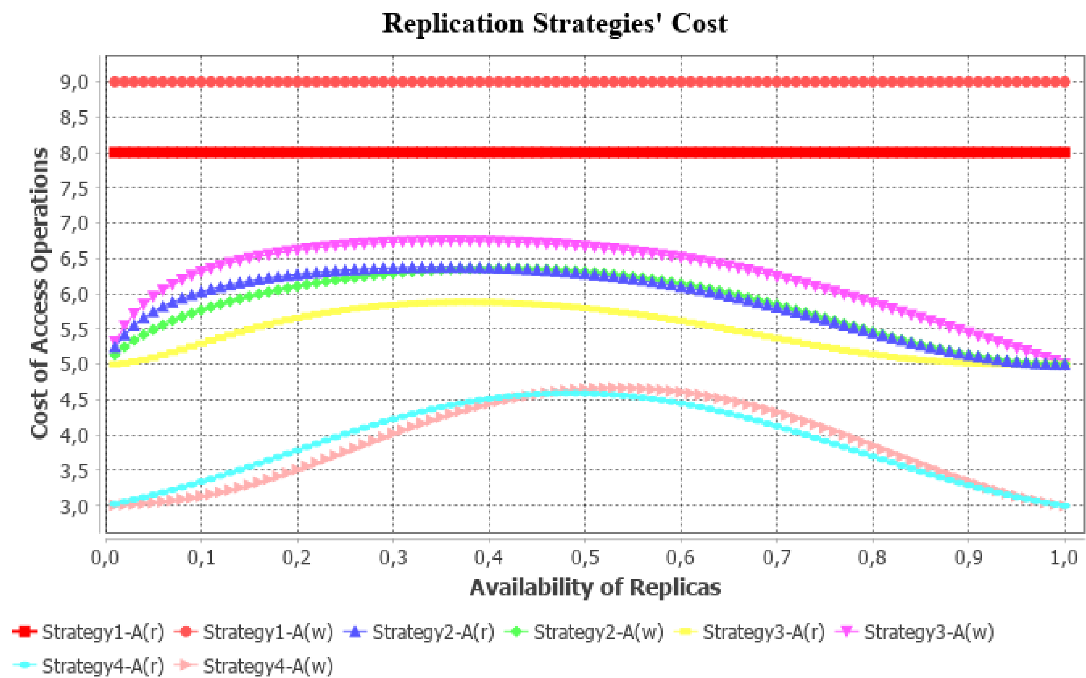

FIGURE 5.47: Costs of MCS, hybrid, and mutated DRSs

brute-forcing the combinations for DRSs, which would be too computation expensive to pursue. Figure 5.48 shows auto-generated wide ranges of access operations´ availability graphs through genetic programming, the strategies of suitable choices of objectives can be easily chosen. Therefore, this automatic mechanism to flexibly "glue" replication strategies together, in a certain fashion, on certain locations, to make

them optimized, opens up new possibilities of designing new replication strategies, up-to-now unknown.

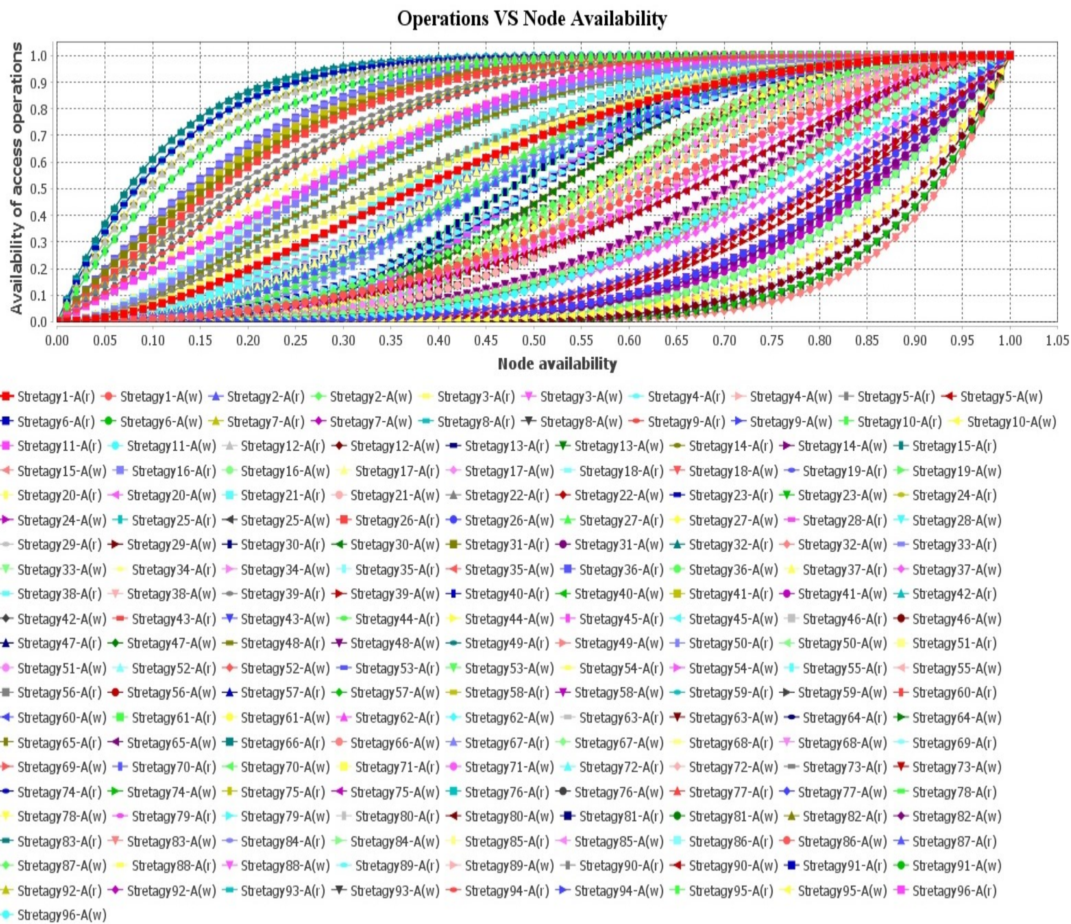

FIGURE 5.48: Auto-generated availability graphs via genetic programming

5.3 Summary

This research uses a genetic programming-based multi-objective optimization approach that endeavors to not only identify but also design new data replication strategies and optimize their conflicting objectives as a single-valued metric. It demonstrates the designing of replication strategies that are innovative and such combinations have not been explored yet, which may open whole new doors in replication and fault tolerance. The introduced multi-crossover and multi-mutation operators to replication, strengthens our machine learning framework, at the same time guaranteeing consistency of the solutions, to generate innovative hybrid replication strategies.

The research demonstrates the usefulness of this genetic programming-based automatic mechanism by reducing the cost significantly while not comprising too much of the availabilities of the access operations. In this chapter, different scenarios are defined, and having defined the system parameters, the system generates new competitive replication strategies. This proves that our approach is very effective and extremely

flexible to offer competitive results w.r.t. the contemporary strategies as well as generating novel strategies even with a slight use of relevant genetic operators. Hence, the research not only provides an intelligent, automatic mechanism to generate new replication strategies but also eases up the decision-making so that relevant strategies with satisfactory trade-offs of constraints can easily be picked and used from the generated solutions at run-time.

Chapter 6

Conclusions and future work

6.1 Summarization and contributions

The work initially demonstrates the usefulness of a hybrid approach based on voting structures for introducing new hybrid DRSs (heterogeneous strategies combined), which may potentially fulfill uncovered application-scenarios for replicated data. This idea utilizes voting structures [11] as a key element to this hybrid approach in data replication. Such a unified representation of DRSs makes it very convenient to perform "crossovers" and merge any quorum-based DRS with other quorum-based DRSs in anticipation of adopting the best properties of the two. It initially combines, models manually cutting-edge replication strategies as unified voting structures, evaluates their performances, and compares the results with contemporary strategies.

This idea of manual designs leads to the automatic generation of new application-optimized DRSs satisfying the needs of given application-specific scenarios. In this regard, this dissertation combines the concepts of fault tolerance in distributed systems with genetic programming. It proposes an innovative, automated mechanism for designing new (up-to-now) unknown hybrid optimized DRSs for specified application-specific scenarios for which no optimal strategy may exist. It uses voting structures in the context of genetic programming to generate new optimized hybrid DRSs irrespective of their varied topologies and patterns for accessing replicas. The proposed mechanism allows replication strategies to evolve as computer programs over several generations and attain a constant evolutionary trajectory. It intelligently designs DRSs without trying to brute force all the possible combinations since the search space is huge. The novel approach does not only consider the availability aspect, but also the cost aspect and successfully models a scenario into a replication strategy.

This GOOGP-based approach, later on, acts as a building block to introduce new multi-type crossover, as well as mutation operators to gain some more fine-grained control over the algorithm in anticipation of designing appropriate solutions, accordingly. These newly introduced genetic operators allow the system to be more effective in exploring more and more possible ways by which to combine different DRSs. Such use of multi-crossover and multi-mutation operators in the context of voting structures is quite unprecedented, which provides more leverage in combining replication strategies together, in a more flexible manner.

Moreover, the research addresses the non-trivial multi-objective optimization problem of DRSs, where no single solution exists that simultaneously optimizes each objective. It explicitly illustrates the trade-offs of newly generated DRSs through a Pareto front

view, hence, makes the decision-making process very simple and convenient. In this process, new DRS are generated, optimized over several generations, and relevant optimized strategies exhibiting suitable properties are picked at run-time. The proposed approach is very effective and extremely flexible to offer competitive results w.r.t. the contemporary strategies as well as generating novel strategies even with a slight use of relevant genetic operators. It aims to reduce the cost of the access operations while not comprising on the availabilities too much, thereby making systems more reliable. This automatic mechanism based on General object-orientated genetic programming has the potential to open whole new doors to easily explore the unknown territory of DRSs, which would not have been easily discovered otherwise.

6.2 Future work

The multifaceted nature of the task urges many possibilities of potential improvements. Therefore, as a part of the future work, an interesting work could be the use of rule-based mining to derive some rules of thumb from statistical data of replication strategies to find some hidden rules out of the operators that which operators to use more in optimizing a particular problem. Considering the stochastic nature of the technique, a more detailed statistical analysis will be carried out to attain sufficient confidence in the results of genetic programming, specifically in the case where it is very computationally expensive to calculate the fitness. In this regard, the tuning of parameters (adaptive, self-adaptive; or predetermined) to find the optimal settings for the problems will also be in our focus. This might include hyper-parameter settings, parameter tuning, as well as control towards adaptive and non-adaptive genetic algorithms. Selection of optimal hyper-parameters through neural networks or self-adaptive algorithms based on the success rate is certainly a possibility, but it may deteriorate the performance in our case. Examining the impacts of initial populations and operators in resolving a problem since convergence can be achieved faster with a better population, i.e., single operator vs multiple operators, each operator on a different population, different operators on each population, etc. A focus can also be on multi-objective optimization using Non-dominated Sorting Genetic Algorithms based on crowding distance or Euclidean distance to yield better results.

The optimization of the algorithm for the scalability to a higher number of replicas, which may include the possibilities and ways to calculate the fitness efficiently as well as using high-performance computing to do the job or even parallelizing different processes. Developing means to generate initial populations effectively, and also, scalability of the project w.r.t. incorporating more and more DRSs as well as converting the other state-of-the-art strategies (automatically) to voting structures. Incorporation of other fault models, as well as cost models, and their integration into our genetic approach to widen the horizon and further gain some fruitful results. Also, the distributed systems are dynamic in nature, which means, over time, networks and devices can fail, and new nodes can join or depart the system. Therefore, given that scenarios can change, the current strategy might not be optimal anymore and a new strategy needs to be calculated. This will require some form of sensing the network/nodes to receive this information and to trigger a reevaluation of the strategy. However, with a dynamic approach, there are other issues to consider such as oscillating behaviors (i.e., switching back and forth between same/similar strategies). Moreover, the development of more complex crossover and mutation operators with some more complex

system parameter settings may also be in our focus, to further strengthen our approach, gain some more fine-grained control over the algorithm (for designing appropriate solutions, accordingly), as well as comparing with the state-of-the-art.

6.3 Dissertation publications

S. M. A. Bokhari, O. Theel, Designing New Data Replication Strategies Automatically, In: *Agents and Artificial Intelligence, Lecture Notes in Artificial Intelligence (LNAI), Springer International Publishing, pp. 308-331* (2021).

S. M. A. Bokhari, O. Theel, Use of Genetic Programming Operators in Data Replication and Fault Tolerance, In: *Proceedings of the 26th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 290-299*, Hong Kong (2020).

S. M. A. Bokhari, O. Theel, Introducing Novel Crossover and Mutation Operators into Data Replication Strategies for Distributed Systems, In: *Proceedings of the 25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 21-30*, Perth, Australia (2020).

S. M. A. Bokhari, O. Theel, A Genetic Programming-based Multi-objective Optimization Approach to Data Replication Strategies for Distributed Systems, In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC, WCCI), pp. 1–9*, Glasgow, Scotland (2020).

S. M. A. Bokhari, O. Theel, Design of Scenario-based Application-optimized Data Replication Strategies through Genetic Programming, In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART), pp. 120-129*, Valletta, Malta (2020).

S. M. A. Bokhari, O. Theel, A Flexible Hybrid Approach to Data Replication in Distributed Systems, Computing Conference (SAI), In: *Advances in Intelligent Systems and Computing (AISC), Springer, Volume 1(1228), pp. 196-207*, London, UK (2020).

For more details regarding these publications, please see Section 1.4.

Bibliography

[1] Peter G. Neumann. “Illustrative risks to the public in the use of computer systems and related technology”. In: *ACM Special Interest Group on Software Engineering (SIGSOFT) Software Engineering Notes* 19.1 (1994), 16–29.

[2] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. 1987. ISBN: 978-0-201-10715-9.

[3] D. Agrawal and A. Abbadi. “The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data”. In: *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB), Morgan Kaufmann* (1990), 243–254.

[4] S. Y. Cheung, M. H. Ammar, and M. Ahamad. “The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 4.6 (1992), 582–592.

[5] H. Robert. “A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases”. In: *ACM Transactions on Database Systems (TODS)* 4.2 (1979), 180–207.

[6] M. Naor and A. Wool. “The Load, Capacity, and Availability of Quorum Systems”. In: *SIAM Journal on Computing* 27.2 (1998), 423–447.

[7] J. Ricardo et al. “How to Select a Replication Protocol According to Scalability, Availability, and Communication Overhead”. In: *Proceedings 20th IEEE Symposium on Reliable Distributed Systems (SRDS)* (2001), 24–33.

[8] O. Theel and H. Pagina. “Optimal Replica Control Protocols Exhibit Symmetric Operation Availabilities”. In: *Proceedings of the 28th International Symposium on Fault-Tolerant Computing (FTCS-28), IEEE Computer Society Press* (1998), 252–261.

[9] K. Miettinen. *Nonlinear Multi-objective Optimization*. Kluwer Academic, Boston, 1999. ISBN: 978-1-461-55563-6.

[10] S. M. A. Bokhari and O. Theel. “A Flexible Hybrid Approach to Data Replication in Distributed Systems”. In: *Computing Conference (SAI), Advances in Intelligent Systems and Computing (AISC), Springer* 1228.1 (2020), 196–207.

[11] O. Theel. “General Structured Voting: A Flexible Framework for Modelling Cooperations”. In: *Proceedings of the 13th International Conference on Distributed Computing Systems, IEEE Computer Society Press* (1993), pp. 227–236.

[12] O. Theel. “Rapid Replication Scheme Design using General Structured Voting”. In: *Proceedings of the 17th Annual Computer Science Conference* (1994), 669–677.

[13] H. Pagnia and O. Theel. “Priority-based Quorum Protocols for Replicated Objects”. In: *Proceedings of the 2nd International Conference on Parallel and Distributed Computing and Networks (PDCN), IASTED* (1998), 530–535.

[14] S. M. A. Bokhari and O. Theel. "Design of Scenario-based Application-Optimized Data Replication Strategies through Genetic Programming". In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART), Scitepress* (2020), 120–129.

[15] S. M. A. Bokhari and O. Theel. *Designing New Data Replication Strategies Automatically*. 2021, pp. 308–331. ISBN: 978-3-030-71158-0.

[16] S. M. A. Bokhari and O. Theel. "A Genetic Programming-based Multi-objective Optimization Approach to Data Replication Strategies for Distributed Systems". In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC, WCCI), IEEE* (2020), 1–9.

[17] S. M. A. Bokhari and O. Theel. "Use of Genetic Programming Operators in Data Replication and Fault Tolerance". In: *Proceedings of the 26th IEEE International Conference on Parallel and Distributed Systems (ICPADS), IEEE* (2020), pp. 290–299.

[18] S. M. A. Bokhari and O. Theel. "Introducing Novel Crossover and Mutation Operators into Data Replication Strategies for Distributed Systems". In: *Proceedings of the 25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), IEEE* (2020), pp. 21–30.

[19] A. Avižienis et al. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Transactions on Dependable and Secure Computing, IEEE* 1.1 (2005), 11–33.

[20] C. Storm. *Specification and Analytical Evaluation of Heterogeneous Dynamic Quorum-based Data Replication Schemes*. 2012. ISBN: 978-3-834-82381-6.

[21] K. Echtle. *Fehlertoleranzverfahren (in German)*. 1990. ISBN: 978-3-642-75765-5.

[22] F. Cristian et al. "Atomic Broadcast: from Simple Message Diffusion to Byzantine Agreement". In: *Information and Computation* 118.1 (1995), 158–179.

[23] P. Jalote. *Fault Tolerance in Distributed Systems*. 1994. ISBN: 978-0-13-301367-2.

[24] F. Cristian. "Understanding Fault-tolerant Distributed Systems". In: *Communications of the ACM* 34.2 (1991), 56–78.

[25] B. W. Lampson. "Atomic transactions. In Distributed Systems – Architecture and Implementation". In: *Lecture Notes in Computer Science, Springer* 105 (1981), 246–265.

[26] B. W. Lampson and H. E. Sturgis. "Crash Recovery in a Distributed Data Storage System". In: *Unpublished technical report, Xerox Palo Alto Research Center* 105 (1979), 246–265.

[27] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem". In: *Transactions on Programming Languages and Systems, ACM* 4.3 (1982), 382–401.

[28] D. Powell. "Failure Mode Assumptions and Assumption Coverage". In: *Proceedings of the 22nd International Symposium on Fault-tolerant Computing (FTCS-22)* (1992), 386–395.

[29] T. D. Chandra and S. Toueg. "Unreliable Failure Detectors for Reliable Distributed Systems". In: *Journal of the ACM* 2.43 (1996), 225–267.

[30] M. J. Fischer, N. A. Lynch, and M. S. Paterson. "Impossibility of Distributed Consensus with one Faulty Process". In: *Journal of the ACM* 32.2 (1985), 374–382.

[31] M. Hirt and U. Maurer. "Complete Characterization of Adversaries Tolerable in Secure Multiparty Computation". In: *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC)* (1997), 25–34.

[32] T. Warns. “Structural Failure Models for Fault-Tolerant Distributed Computing”. In: *Ph.D. Thesis, Department of Computer Science, University of Oldenburg, Germany* (2009).

[33] P. Bernstein and N. Goodman. “An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases”. In: *ACM Transactions on Database Systems (TODS)* 9.4 (1984), 596–615.

[34] D. Gifford. “Weighted Voting for Replicated Data”. In: *Proceedings of the Seventh ACM Symposium on Operating Systems Principles (SOSP)* (1979), 150–162.

[35] A. Kumar. “Hierarchical Quorum Consensus: A New Algorithm for Managing Replicate Data”. In: *IEEE Transactions on Computers* 40.9 (1991), 996–1004.

[36] C. Wu and G. G. Belford. “The Triangular Lattice Protocol: A Highly Fault Tolerant and Highly Efficient Protocol for Replicated Data”. In: *Proceedings of the 11th Symposium on Reliable Distributed Systems (SRDS)* (1992), 66–73.

[37] J. Jajodia and D. Mutchler. “Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database”. In: *ACM Transactions on Database Systems (TODS)* 15.2 (1990), pp. 230–280.

[38] O. Theel and T. Strauß. “Automatic Generation of Dynamic Coterie-based Replication Schemes”. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (1998), 1606–1613.

[39] S. C. Choi and H. Y. Youn. “Dynamic Hybrid Replication Effectively Combining Tree and Grid Topology”. In: *The Journal of Supercomputing* 59.3 (2012), 1289–1311.

[40] M. Arai et al. “Analysis of Read and Write Availability for Generalized Hybrid Data Replication Protocol”. In: *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)* (2004), 143–150.

[41] Yong-Ju Lee, Hag-Young Kim, and Cheol-Hoon Lee. “Cell Approximation Method in Quorum Systems for Minimizing Access Time”. In: *Cluster Computing* 12 (2009), 387–398.

[42] O. Theel. “Meeting the Application’s Needs: A Design Study of a Highly Customized Replication Scheme”. In: *Proceedings of the Pacific Rim International Symposium on Fault Tolerant* (1993), 111–117.

[43] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001. ISBN: 978-0-471-87339-6.

[44] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992. ISBN: 978-0-262-11170-6.

[45] W. Banzhaf et al. “Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications”. In: *Morgan Kaufmann Publishers Inc.* (1998).

[46] R. Schadek and O. Theel. “Increasing the Accuracy of Cost and Availability Predictions of Quorum Protocols”. In: *Proceedings of the 22nd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)* (2017), 98–103.

[47] R. Schadek, O. Kramer, and O. Theel. “Predicting Read- and Write-Operation Availabilities of Quorum Protocols based on Graph Properties”. In: *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART)* 2 (2018), 550–558.

[48] J. Ricardo et al. “Are Quorums an Alternative for Data Replication?” In: *ACM Transactions on Database Systems* 28.3 (2003), 257–294.

[49] Daniel Barbara and Héctor García-Molina. "The Vulnerability of Vote Assignments". In: *ACM Transactions on Computer Systems* 4.3 (1986), 187–213.

[50] Shun Yang Cheung, Mustaque Ahamad, and Mostafa H. Ammar. "Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data". In: *IEEE Transactions on Data and Knowledge Engineering* 1.3 (1989), 387–397.

[51] Christos H. Papadimitriou and Martha Sideri. "Optimal Coteries". In: *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC)* (1991), 75–80.

[52] Mirjana Obradovic and Piotr Berman. "Voting as the Optimal Pessimistic Scheme for Managing Replicated Data". In: *Proceedings of the 9th Symposium on Reliable Distributed Systems (SRDS)* (1990), 126–135.

[53] Akhil Kumar, Michael Rabinovich, and Rakesh K. Sinha. "A Performance Study of General Grid Structures for Replicated Data". In: *In Proceedings of the 13th International Conference on Distributed Computer Systems (ICDCS)* (1993), 178–185.

[54] Mitchell L. Neilsen. "Quorum Structures in Distributed Systems". In: *Ph.D. Thesis, Kansas State University, Kansas, KS, U.S.A* (1992).

[55] O. Theel and H. Pagnia. "General Design of Grid-based Data Replication Schemes using Graphs and a few Rules". In: *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)* (1995), 395–403.

[56] Rich Caruana and Alexandru Niculescu-Mizil. "An Empirical Comparison of Supervised Learning Algorithms". In: *Proceedings of the 23rd international conference on Machine learning* (2006), pp. 161–168.

[57] Ankur A. Patel. "Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data". In: *O'Reilly Media* (2019).

[58] O. Chapelle, B. Schölkopf, and A. Zien. "Semi-Supervised Learning (Adaptive Computation and Machine Learning)". In: *MIT Press* (2006).

[59] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction". In: *MIT Press* (2018).

[60] X. Yu and M. Gen. "Introduction to Evolutionary Algorithms". In: *Springer Book* (2010).

[61] M. Amrane et al. "Breast Cancer Classification using Machine Learning". In: *Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)* (2018), pp. 1–4.

[62] Z. Wang, K. Fu, and J. Ye. "Learning to Estimate the Travel Time". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining* (2018), pp. 858–866.

[63] D. Wang et al. "A Semi-supervised Graph Attentive Network for Financial Fraud Detection". In: *Proceedings of the IEEE International Conference on Data Mining (ICDM)* (2019), pp. 598–607.

[64] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

[65] D. Fogel. "Evolutionary Computation: The Fossil Record". In: *Wiley-IEEE Press* (1998).

[66] J. F. Miller. "Cartesian Genetic Programming". In: *Natural Computing Series, Springer* (2011).

[67] J. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975. ISBN: 978-0-262-27555-2.

[68] O. Kramer. *Genetic Algorithm Essentials*. Vol. 679. Springer, 2017. ISBN: 978-3-319-52156-5.

[69] I. Rechenberg. "Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". In: *Fromman-Holzboog* (1971).

[70] H. Schwefel. "Numerische Optimierung von Computer-Modellen. Birkhäuser". In: *Fromman-Holzboog* (1977).

[71] L. Fogel, A.J. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966. ISBN: 978-0-471-26516-0.

[72] R. Friedberg. "A Learning Machine: Part I". In: *IBM Journal of Research and Development* 2 (1958), pp. 2–13.

[73] R. Friedberg and B. Dunham. "A Learning Machine: Part II". In: *IBM Journal of Research and Development* 3 (1959), 282–287.

[74] S. F. Smith. "A Learning System Based on Genetic Adaptive Algorithms". In: *Ph.D. Thesis, University of Pittsburgh* (1980).

[75] R. Forsyth. "BEAGLE A Darwinian Approach to Pattern Recognition". In: *Kybernetes* 10.3 (1981), 159–166.

[76] N. L. Cramer. "A Representation for the Adaptive Generation of Simple Sequential Programs". In: *Proceedings of the International Conference on Genetic Algorithms and their Applications, ACM* (1985).

[77] D. Dickmanns, J. Schmidhuber, and A. Winklhofer. "Der genetische Algorithmus: Eine Implementierung in Prolog". In: *Fortgeschrittenenpraktikum , Institut für Informatik, Technische Universität München* (1987).

[78] J. Schmidhuber. "Evolutionary Principles in Self-referential Learning". In: *Diploma thesis, Institut für Informatik, Technische Universität Mänchen* (1987).

[79] R. Raghavjee and N. Pillay. "A Comparison of Genetic Algorithms and Genetic Programming in Solving the School Timetabling Problem". In: *Proceedings of the 4th World Congress on Nature and Biologically Inspired Computing (NaBIC)* (2012), pp. 98–103.

[80] M. Orlov and M. Sipper. "Genetic Programming in the Wild: Evolving Unrestricted Bytecode". In: *Proceedings of the 11th Conference on Genetic and Evolutionary Computation* (2009), 1043–1050.

[81] M. Orlov and M. Sipper. "FINCH: A System for Evolving Java (Bytecode)". In: *Genetic Programming Theory and Practice VIII, Springer* 8 (2010), 1–16.

[82] M. Orlov and M. Sipper. "Flight of the FINCH Through the Java Wilderness". In: *IEEE Transactions on Evolutionary Computation* 15.2 (2011), 166–182.

[83] H. Al-Sahaf et al. "A Survey on Evolutionary Machine Learning". In: *Journal of the Royal Society of New Zealand* 49 (2019), 205–228.

[84] Y. Bi, B. Xue, and M. Zhang. "An Evolutionary Deep Learning Approach using Genetic Programming with Convolution Operators for Image Classification". In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)* (2019), 3197–3204.

[85] L. Shao et al. "Feature Learning for Image Classification via Multi-objective Genetic Programming". In: *IEEE Transactions on Neural Networks and Learning Systems* 25 (2014), 1359–1371.

[86] H. Al-Sahaf et al. "Automatically Evolving Rotation-invariant Texture Image Descriptors by Genetic Programming". In: *IEEE Transactions on Evolutionary Computation* 21 (2017), 83–101.

[87] Y. Bi, B. Xue, and M. Zhang. "An Automatic Feature Extraction Approach to Image Classification using Genetic Programming". In: *Proceedings of the International Conference on the Applications of Evolutionary Computation* (2018), 421–438.

[88] Kai Staats. Last accessed in September 2020. URL: http://geneticprogramming.com.

[89] J. McCarthy. "Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I". In: *Communications of the ACM* (1960), 184–195.

[90] R. Poli and W. B. Langdon. "Foundations of Genetic Programming". In: *Springer* (2002).

[91] R. Poli, W. B. Langdon, and N. F. McPhee. "A Field Guide to Genetic Programming". In: *Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk* (2008).

[92] C. Ryan, J. J. Collins, and M. O. Neill. "Grammatical Evolution: Evolving Programs for an Arbitrary Language". In: *Proceedings of the European Conference on Genetic Programming* (1998).

[93] R. McKay et al. "Grammar-based Genetic Programming: A Survey". In: *Genetic Programming and Evolvable Machines* 11.3 (2010).

[94] M. O'Neill and C. Ryan. "Grammatical Evolution". In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), 349–358.

[95] M. O'Neill and C. Ryan. "Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language". In: *Springer* (2003).

[96] L. Spector and A. Robinson. "Genetic Programming and Autoconstructive Evolution with the Push Programming Language". In: *Genetic Programming and Evolvable Machines 3* (2002), 7–40.

[97] J. F. Miller, P. Thomson, and T. C. Fogarty. "Designing Electronic Circuits Using Evolutionary Algorithms: Arithmetic Circuits: A Case Study". In: *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications* (1998), 105–131.

[98] J. F. Miller. "An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach". In: *Proceedings of the Genetic and Evolutionary Computation Conference* (1999), 1135–1142.

[99] J. F. Miller and P. Thomson. "Cartesian Genetic Programming". In: *Proceedings of the European Conference on Genetic Programming, Springer LNCS* 1802 (2000), 121–132.

[100] A. V. Aho, J. D. Ullman, and J. E. Hopcroft. "Data Structures and Algorithms". In: *Addison–Wesley* (1983).

[101] N. Deo. "Graph Theory with Applications to Engineering and Computer Science". In: *Prentice-Hall* (2004).

[102] G. Chartrand, L. Lesniak, and P. Zhang. *Graphs and Digraphs, Fifth Edition*. 2010. ISBN: 978-1-439-82627-0.

[103] S. Louis and G. J. E. Rawlins. "Using Genetic Algorithms to Design Structures". In: *Technical Report TR326, Department of Computer Science, Indiana University* (1990).

[104] S. Louis and G. J. E. Rawlins. "Designer Genetic Algorithms: Genetic Algorithms in Structure Design". In: *Proceedings of the International Conference on Genetic Algorithms, Morgan Kauffman* (1991), 53–60.

[105] S. Louis. "Genetic Algorithms as a Computational Tool for Design". In: *Ph.D. thesis, Department of Computer Science, Indiana University* (1993).

[106] A. J. Turner and J. F. Miller. "The Importance of Topology Evolution in NeuroEvolution: A Case Study using Cartesian Genetic Programming of Artificial Neural Networks". In: *Proceedings of the Thirty-third SGAI International Conference on Artificial Intelligence, Springer International Publishing* (2013), 213—226.

[107] A. J. Turner and J. F. Miller. "Cartesian Genetic Programming encoded Artificial Neural Networks: A Comparison using Three Benchmarks". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM* (2013), pp. 1005–1012.

[108] A. M. Ahmad et al. "Breast Cancer Detection Using Cartesian Genetic Programming evolved Artificial Neural Networks". In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)* (2012), 1031—1038.

[109] M. M. Khan, G. M. Khan, and J. F. Miller. "Developmental Plasticity in Cartesian Genetic Programming Artificial Neural Networks". In: *Special session on Artificial Neural Networks and Intelligent Information Processing in Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO), Scitepress* (2011), pp. 197–198.

[110] M. M. Khan, G. M. Khan, and J. F. Miller. "Evolution of Neural Networks using Cartesian Genetic Programming". In: *Proceedings of 12th IEEE Congress on Evolutionary Computation (CEC)* (2010).

[111] G. M. Khan, J. F. Miller, and M. M. Khan. "Evolution of Optimal ANNs for Non-Linear Control Problems Using Cartesian Genetic Programming". In: *Proceedings of the International Conference on Artificial Intelligence (ICAI), CSREA Press* (2010).

[112] R. Poli. "Parallel Distributed Genetic Programming". In: *Tech. Rep. CSRP-96-15, School of Computer Science, University of Birmingham* (1996).

[113] R. Poli. "Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming". In: *Proceedings of the International Conference on Genetic Algorithms* (1997), 346–353.

[114] S. Silva and E. Costa. "Dynamic Limits for Bloat Control in Genetic Programming and a Review of Past and Current Bloat Theories". In: *Genetic Programming and Evolvable Machines 10* (2009), 141–179.

[115] J. F. Miller and S.L. Smith. "Redundancy and Computational Efficiency in Cartesian Genetic Programming". In: *IEEE Transactions on Evolutionary Computation* 10.2 (2006), 167–174.

[116] R. Abbott. "Object-oriented Genetic Programming, an Initial Implementation". In: *Proceedings of the International Conference on Machine Learning: Models, Technologies and Applications* (2003).

[117] S. M. Lucas. "Exploiting Reflection in Object Oriented Genetic Programming". In: *European Conference on Genetic Programming (EuroGP), LNCS* 3003 (2004), pp. 369–378.

[118] W. S. Bruce. "Automatic Generation of Object-oriented Programs using Genetic Programming". In: *Proceedings of the 1st Annual Conference on Genetic Programming* (1996), pp. 267–272.

[119] T. White, J. Fan, and F. Oppacher. “Basic Object Oriented Genetic Programming”. In: *Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (2011), pp. 59–68.

[120] M. F. Brameier and W. Banzhaf. “Linear Genetic Programming”. In: *Genetic and Evolutionary Computation Series, Springer* (2007).

[121] Steven H. Lai A. Homaifar Charlene X. Qi. “Constrained Optimization via Genetic Algorithms”. In: *Simulation* 4 (1994), pp. 242–253. URL: 62.

[122] G. Singh and K. Deb. “Comparison of Multi-modal Optimization Algorithms based on Evolutionary Algorithms”. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (2006), pp. 1305–1312.

[123] B. Korte and J. Vygen. *Combinatorial Optimization, Fifth Edition*. Springer, 2011. ISBN: 978-3-642-24487-2.

[124] K. Deb. “Multi-objective Optimization”. In: *Search Methodologies, Springer* (2014), pp. 403–449.

[125] G. Syswerda. “Simulated Crossover in Genetic Algorithms”. In: *Foundations of Genetic Algorithms (FOGA), Morgan Kaufmann* (1993), 239–255.

Acronyms

1SR one-copy serializability

DRS data replication strategy

rq read quorum (a threshold of a min. replica number to execute a read operation)

wq write quorum (a threshold of a min. replica number to execute a write operation)

MCS majority consensus strategy

ROWA read-one write-all

TQP tree quorum protocol

WVS weighted voting strategy

HQC hierarchical quorum consensus

C-Cover column cover
CC-Cover complete column cover

TLP triangular lattice protocol

H-Cover horizontal cover

V-Cover vertical cover

DAG directed acyclic graph

EA evolutionary algorithm

GA genetic algorithm

ES evolutionary strategies

GP genetic programming

EP evolutionary programming

LGP linear genetic programming

GE grammatical evolution

CGP cartesian genetic programming

ANN artificial neural network

GOOGP general object-oriented genetic programming

RQ read quorum (the set of replicas of a read quorum)

WQ write quorum (the set of replicas of a write quorum)

RQS read quorum set (the super-set of all the RQs)

WQS write quorum set (the super-set of all the WQs)

FW fitness weightage

Symbols

n number of replicas

ϵ a threshold of number of replicas

$\mathbb{N}^+$ natural numbers excluding zero

p probability of individual replicas

A_r availability of read operation
A_w availability of write operation

q sum of the replicas forming a quorum

α a threshold of availability of the read operation

β a threshold of availability of the write operation

C_r cost of read operation
C_w cost of write operation

$minRQ$ minimum quorum necessary to execute a read operation
$minWQ$ minimum quorum necessary to execute a write operation

$\mathbb{R}^+$ real positive numbers

γ a threshold of cost of the read operation

δ a threshold of cost of the write operation

μ number of parent strategies in a generation

λ number of offspring strategies in a generation