A Genetic Programming-Based Multi-Objective Optimization Approach to Data Replication Strategies for Distributed Systems

Syed Mohtashim Abbas Bokhari
Department of Computer Science
University of Oldenburg
Germany
syed.mohtashim.abbas.bokhari@uni-oldenburg.de

Oliver Theel
Department of Computer Science
University of Oldenburg
Germany
oliver.theel@uni-oldenburg.de

***Abstract*—Data replication is the core of distributed systems to enhance their fault tolerance and make services highly available to the end-users. Data replication masks run-time failures and hence, makes the system more reliable. There are many contemporary data replication strategies for this purpose, but the decision to choose an appropriate strategy for a certain environment and a specific scenario is a challenge and full of compromises. There exists a potentially indefinite number of scenarios that cannot be covered entirely by contemporary strategies. It demands designing new data replication strategies optimized for the given scenarios. The constraints of such scenarios are often conflicting in a 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 purpose. In this regard, this research provides 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. 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. Moreover, it makes the notion of hybrid strategies easier to accomplish which otherwise would have been very cumbersome to achieve, therefore, to optimize.**

Keywords— Distributed Systems, Fault Tolerance, Data Replication, Quorum Protocols, Operation Availability, Operation Cost, Voting Structures, Optimization, Pareto Front, Machine Learning, Genetic Programming.

I. INTRODUCTION

Data replication is replicating the copies of the same data over several nodes in anticipation of achieving high availability of the services, but accomplishing high availability is not a straightforward task. Having replicated the data, it has to be managed to avoid inconsistencies, which affect the correctness of the data. Inconsistency means discrepancy in the data among created replicas. Furthermore, the data needs to be exclusively locked for the write operations to avoid conflicts, so that, availability is achieved without destroying the correctness of the data. There exist strategies known as data replication strategies (DRSs), i.e., [12] to control such replicated behavior of a system. These DRSs manage those created replicas, but to choose a certain strategy for a certain scenario is a trade-off between different quality metrics, i.e., load, capacity, availability [1], scalability, and cost [2]. These metrics are often conflicting with each other in a way that one cannot be optimized without deteriorating the others. This could easily fall into the realm of a multi-objective optimization problem [3]. 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. In this regard, our work is an interesting overlap between the concepts of replication in distributed systems and machine learning.

The paper is structured as follows: Section 2 states the problem. Section 3 sheds light on related work and the innovation of our research. Section 4 describes the methodology. Section 5 discusses the implementation aspects of our approach. Section 6 presents the results and contributions. Section 7 concludes the paper by summarizing the key points.

II. PROBLEM STATEMENT

Figure 1 depicts numerous scenarios between the availability of the access operations (read or write) and their costs whereas in our case, consistency is static and adheres to 1SR [4] all the time. 1SR allows a replicated system to behave as a non-replicated system.

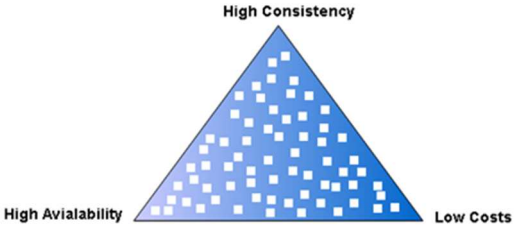

Fig 1. Data replication scenarios

Availability, here, is the probability by which a user successfully performs an access operation and the cost is the average minimal number of replicas, a user needs to access to get the expected result. The availabilities of read and write operations are optimally point-symmetric to each other [5], which implies both cannot be optimized independently. Also, an increase in the cost of a read operation often compromises the write operation's cost. Moreover, the relationship between

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

the costs and availability of the access operations is not linear either. Numerous scenarios exist between their different trade-offs, including the total number of replicas and their individual availabilities. These scenarios cannot be fulfilled by the current contemporary strategies entirely. There is a need of developing new DRSs [6] to satisfy these scenarios. For this, the paper 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) resolving the multi-objective problem, 2) eliminating the diversity between the DRSs in the form of logical and topological differences which may hinder the design of hybrid solutions, 3) finally, but most importantly, developing a machine learning approach to automatically, but intelligently design, as well as optimizing DRSs for specified scenarios.

III. Related work

There are various contemporary solutions in the form of DRSs, i.e., Read-One Write-All [7], the Majority Consensus Strategy (MCS) [8], the Tree Quorum Protocol [9], the Weighted Voting Strategy [10], Hierarchical Quorum Consensus [11], the Grid Protocol [12], the Triangular Lattice Protocol (TLP) [13], etc. These strategies have different semantics and properties for fulfilling different thresholds of objectives, i.e., availabilities, costs, total replicas, etc. Since, there are many trade-offs between these objectives forming different scenarios, there is no single best solution. And as discussed, the solution DRSs are insufficient to cover the scenarios entirely. This brings us to the question of designing new strategies and optimizing them [6]. For this, the paper uses a genetic programming-based approach that enables the system to design holistic hybrid DRSs at run-time and optimize them over several generations of evolution. An optimized strategy can be picked at run-time depending upon the scenario and preferences of certain objectives.

This paper aims to design new hybrid DRSs automatically. In this regard, only a few attempts are found in literature, i.e., [14] and [15], which primarily combine Tree Quorum Protocols with Grid Protocols. It is mainly because the strategies are diverse and exhibit different topologies as well as semantics by which to access replicas, thereby making it very cumbersome to accommodate them into hybrid solutions. For this, we use a unified representation of DRSs known as voting structures to eliminate such differences so that any quorum-based strategy can freely be merged with any other quorum-based strategy. Moreover, expert-based manual designs of optimized DRSs using the concept of voting structures have been presented in [16] and [17], but lack automation which limits the efficiency of the approach, since the search space is huge. Therefore, our approach endeavors to automatically design new solutions and optimize them through machine learning to satisfy the specified scenarios, hence, assisting multi-criteria decision making.

The next section discusses the adopted methodology to address the research problems.

IV. Methodology

Figure 2 shows an abstract representation of our approach. Simplistically, having defined a scenario, it starts from a set of replication strategies being converted into a unified representation of voting structures (representing each a computer program) and stored in a scalable database repository. Machine learning, mainly genetic programming, is then applied to the repository to search or design appropriate solutions and optimize them accordingly.

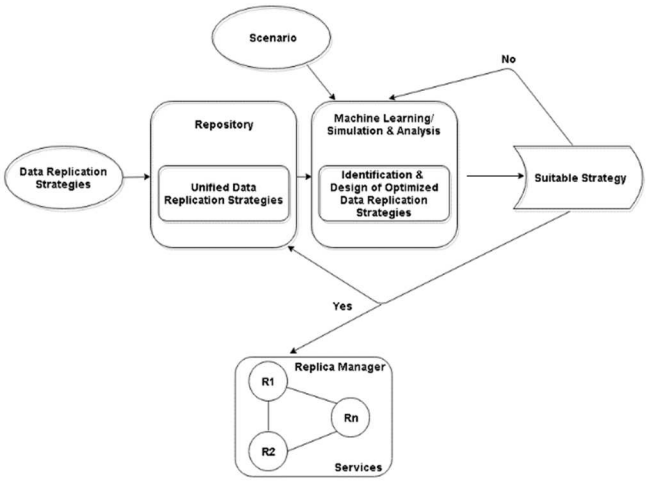

Fig 2. Methodology

The system plots newly designed innovative solutions via genetic programming and an appropriate solution (possibly of higher fitness) satisfying the specified criteria is picked at run-time. The chosen solution is stored back in the repository for future use in the genetic process to improve the solutions or in case, the same scenario comes up again. We explain these aspects in detail in this section.

A. *Voting structures*

The research uses General Structured Voting [18] to eliminate any possibilities of topological and logical impositions in accessing replicas. This acts as a unified representation of all DRSs and makes them flexible and convenient enough for the machine learning framework to work on them for designing and optimizing new solutions. Figure 3 shows the adopted directed acyclic graph (DAG) representation known as a voting structure to embody the quorum mechanism for distributed systems.

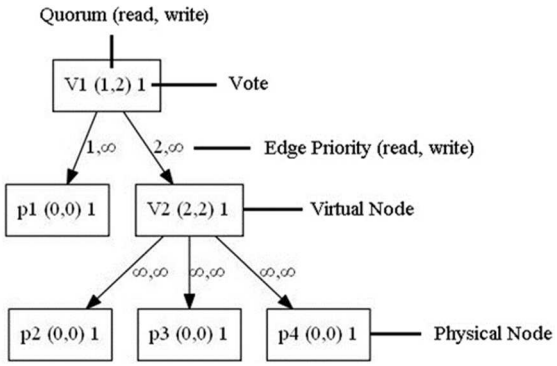

Fig 3. Example of a voting structure

Every individual voting structure is a computer program that is interpreted by our general algorithm at run-time to derive read and write quorum sets. These quorum sets are used to manage replicated objects. A voting structure, hence, is comprised of physical and virtual replicas. Physical are actual replicas, which in the given instance, is four in total while the virtual replicas serve to form groupings of physical and logical replicas. Every node is endowed with a threshold of a minimal number of replicas as quorum rq (wq) for read and write operations, respectively, whereas votes act as a weightage of that node in the collection of a quorum. This voting structure is probed from the top recursively and the quorum for each node to gather per operation has to be less than or equal to the sum of the votes of its children. In some cases, an ordering in the form of an edge priority could also be set (1 being the highest and ∞ being the lowest) to access replicas in a specified order as to reduce the cost. On each level, in general, the quorum has to obey the following rules for a total number of votes V to meet the consistency criterion:

$$rq + wq > V \quad \text{(to avoid read-write conflicts)} \tag{1}$$

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

For example, the voting structure given in Figure 3 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 4 represents a relatively complex example of a TLP comprised of six replicas being modeled as a voting structure. Next, we explain the constraints of a scenario.

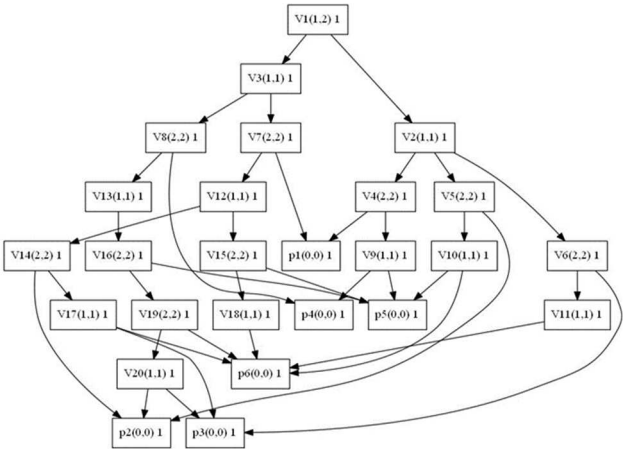

Fig 4. TLP as a voting structure

B. Constraints-based scenario

The scenarios constitute different parameters and thresholds for the replication strategies to adhere to. The scenario parameters 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.

1) Consistency of operations

In our research, the consistency is 1SR which provides high consistency being maintained by the intersection property between every read (write) and write (write) operations of a DRS. This property must be maintained throughout the genetic process otherwise solution becomes invalid.

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 the access operations.

$$n, \varepsilon \in \mathbb{N}^+ \\ \wedge\ n \leqslant \varepsilon \tag{3}$$

3) Availability of access operations

This availability is a probability by which an access operation can be successfully performed by a DRS. For the given instance (Figure 3), the read quorum set (RQS) and write quorum set (WQS) are super-sets of all the RQs wrt. to the full set of replicas (RQ $\cup$ {{p1, p2, p3, p4}}) and WQs (WQ $\cup$ {{p1, p2, p3, p4}}).

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}, {p1,p2,p3,p4}}

(See [5] 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.

$$Ar(p, n) = \Sigma_{\forall q \in RQS}\, p^{|q|}(1-p)^{n-|q|} \tag{4}$$

$$Aw(p, n) = \Sigma_{\forall q \in WQS}\, p^{|q|}(1-p)^{n-|q|} \tag{5}$$

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

$$Ar, Aw,\ \alpha, \beta \in [0, 1] \\ \wedge\ Ar \geqslant \alpha \\ \wedge\ Aw \geqslant \beta \tag{6}$$

4) Cost of access operations

As a cost notion, we use the average minimal cost for a read or a write operation. It is calculated by summing up the minimal operation cost minRQ (minWQ) to build the quorums 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. In the context of the given example (Figure 3), i.e., minRQ ({p1,p2,p3}) is $|\{p1\}| = 1$, minRQ ({p2,p4}) is $|\{p2,p4\}| = 2$, and minWQ ({p1,p2,p3,p4}) is $|\{p1,p2,p3\}| = 3$.

$$Cr(p,n) = \frac{\Sigma_{\forall q \in RQS}\, p^{|q|}(1-p)^{n-|q|} * minRQ(q)}{Ar(p, n)} \tag{7}$$

$$Cw(p,n) = \frac{\Sigma_{\forall q \in WQS}\, p^{|q|}(1-p)^{n-|q|} * minWQ(q)}{Aw(p, n)} \tag{8}$$

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

$$\begin{aligned} & Cr, Cw, \gamma, \delta \in \mathbb{R}^{+} \\ & \quad \wedge Cr \leqslant \gamma \\ & \quad \wedge Cw \leqslant \delta \end{aligned} \tag{9}$$

5) Fitness weightage

It is a weightage (fw) given to any of the concerned objectives to set its importance in the identification or designing of a prospective solution. It is a value between [0,1] for tilting the fitness value towards certain objectives, which by default would remain neutral.

$$fw \in [0,1] \tag{10}$$

6) Probability of individual replicas

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 $pmin \leqslant p \leqslant pmax$.

$$\begin{aligned} & pmin, p, pmax \in [0,1] \\ & \quad \wedge pmin \leqslant p \leqslant pmax \end{aligned} \tag{11}$$

C. Genetic programming-based multi-objective optimization

We use genetic programming to evolve the generations of replication strategies and optimize their constraints. The criteria are manifested in computable functions known as objective functions (mentioned above) 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 paper) whose elements represent objective functions.

The term "optimization" means designing such a solution DRSs which gives the values of all objective functions acceptable to the decision-maker. 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 in the realm of 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 [5], which means that an increase in one results in a decrease in the other operation's availability. The cost of read and write operations are 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.

For this, we use genetic programming (GP) [19] and [20], which is a subset of machine learning mainly used to optimize computer programs. It constitutes of an encoding scheme, random crossover, mutation, a fitness function, and multiple generations of evolution to meet the specified goal. The encoding scheme in our case are DAG-based voting structures. The crossover [21] is mixing up of genetic material of two existing DRSs to create a new child solution. The mutation is the slightly changing of a newly created child DRS. The fitness function is to evaluate a DRS with respect to 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 meeting the demands, with acceptable constraints.

V. Implementation

The section discusses the implementation aspects of our approach. Our system is implemented in JAVA, which is feasible for large applications and has better cross-platform support. This section examines the algorithm for GP, the fitness function, the respective crossover and mutation operators.

A. Fitness function

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 availability, cost, number of replicas, and the fitness-weightage specified in the scenario as parameters. These values are calculated by our objective functions (see Eqs. 4, 5, 7, and 8) and then passed on here. The weightage, as mentioned earlier, determines the importance of certain objectives over others in a desired solution. This weightage is multiplied to 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 Algo. 1. 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 calculateFitness (availValue, costValue, n, fw) {

2         availFitness  = (fw) * (availValue);
3         costFitness   = (1.0 - fw) * (n / costValue);
```

```
4           return (availFitness + costFitness);
  }
```

B. μ and λ

μ represents the number of parent DRSs of a current generation while λ represents the number of offspring DRSs for the next generation. A higher number of these values may provide an opportunity to explore more possibilities, but it also depends on the initial population.

C. Crossover

There are many ways in which the DRSs can be "glued together" 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 thereby inheriting 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 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. It also discards solutions not adhering to these properties.

D. Mutation

Crossover is performed every time while mutation is performed only with a certain probability. Mutation does slightly change 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 (1) and (2) to adhere to 1SR. Beside 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.

E. GP Algorithm

The algorithm implements genetic programming 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. The initial population of voting structures is generated and stored in a database repository. This initial population is retrieved, parsed, and evaluated by the fitness function and, accordingly, the selection is performed to choose μ individual DRSs of higher fitness to a so-called μList for the next generation. The algorithm randomly picks a pair of strategies to apply crossover and mutation operators to generate a new λ number of solutions. This λList carrying λ number of offspring solutions are then evaluated to identify an appropriate strategy. If the desired fitness is still to be achieved, the selection is performed on the elements of the μList and the λList for the next generation to repeat the process of crossovers and mutations on the chosen DRSs of better fitness. Hence, the DRSs are optimized by every generation and better ones are picked. While replication strategies are being generated, we also plot these solutions 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.

Algorithm 2

```
1 Specify a scenario;
2 Calculate scenarioFitness;
3 Specify μ and λ;
4 Initialize μList;    // carrying parent solutions
5 Initialize λList;    // carrying offspring solutions
6 Boolean isFit = false; // determines if a solution meets the criteria
7 Generate initial population of DRSs to the repository
8 Retrieve, parse & store the generated DRSs to initPopList

9  geneticProgramming(initPopList) {
10      Loop through initPopList
11          Calculate fitness;
12          if (fitness ≥ scenarioFitness) {
13              isFit = true;
14              return;
            }
15      END

16      Choose μ best DRSs to the μList;
17      Do {
18          Empty λList;  // empty the list to add new set of
                                                offspring solutions
19          Loop through λ
20              Select randomly DRS1 from μList;
21              Select randomly DRS2 from μList;
22              Perform crossover of DRS1, DRS2;
23              Generate offspring DRSs;
24              Perform mutation on the offspring;
25              Calculate fitness;

26              if (fitness ≥ scenarioFitness) {
27                  isFit = true;
28                  Store offspring DRS into the
                                                repository;
                }
29              Add offspring DRSs to the λList;
30          END

31          Select μ best DRSs to the μList from
            (μList + λList) for next generation;
        }

32      While (!isFit);
    }
```

VI. Experiments & Results

This section sheds light on the results of the proposed techniques and discusses their properties. First, different scenarios are defined as examples to demonstrate the functioning of our approach and find solutions to the problems, accordingly.

A. *Scenario 1*

In the first scenario, the availability of replicas p is set to 0.6 and the expected read and write availabilities are set to 0.80 and 0.72, respectively. 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 cost for the moment not to be important, therefore, full weight is assigned to availability.

$$p = 0.6;\ \varepsilon = 16;\ \alpha = 0.80;\ \beta = 0.70;$$
$$\gamma = 7.0;\ \delta = 7.0;\ fw = 1.0;$$

The algorithm is run, having set μ and λ to six and 15 respectively, along with a mutation probability of 0.2. Figure 5 depicts a 2´D representation of generated solutions for the specified problem. The x-axis represents the availability while the y-axis represents the cost of the access operations. It 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.

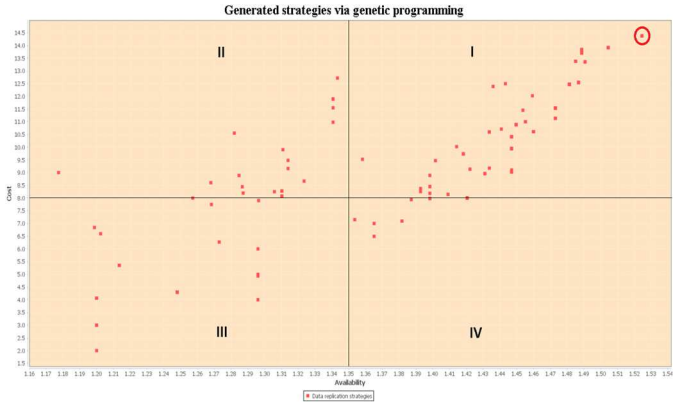

Fig 5. Generated DRSs for scenario 1

An appropriate solution (circled in red) satisfying the criteria, is picked at run-time. Figure 6 presents the selected DRS constituting 16 replicas in total with certain nodes in the voting structure being more important than others in the collection of quorums. The heterogeneous nature of this structure along with variable votes and quorums together, serve to meet the specified constraints of availabilities and the number of replicas while at the same time being not too expensive in cost either. As for availabilities of the access operations, it fairly competes the MCS which is considered to be the best wrt. availability, particularly, for the critical write operation's availability.

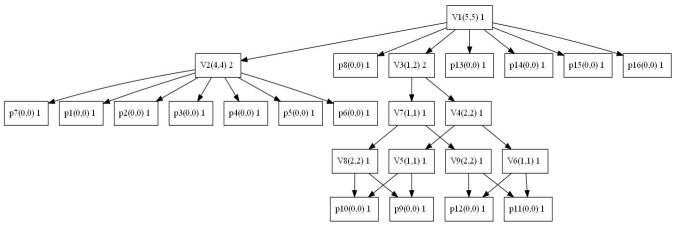

Fig 6. An optimized DRS for scenario 1

Figure 7 represents a comparison between MCS and the discovered hybrid DRS. Red and pink 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 our 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.

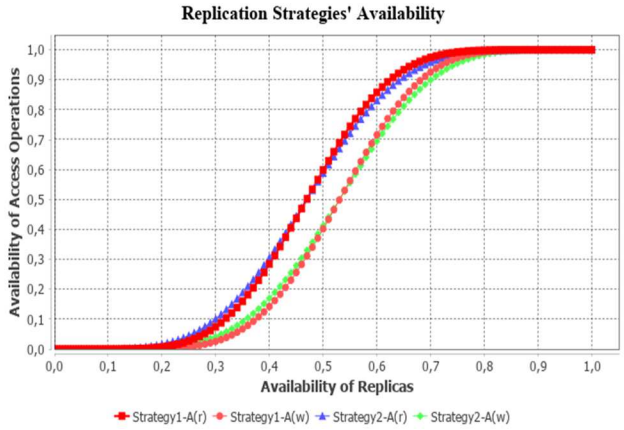

Figure 7: Availability, MCS vs. hybrid DRS (16 replicas)

Figure 8 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, we have significantly decreased the operation costs while not much compromising on availabilities.

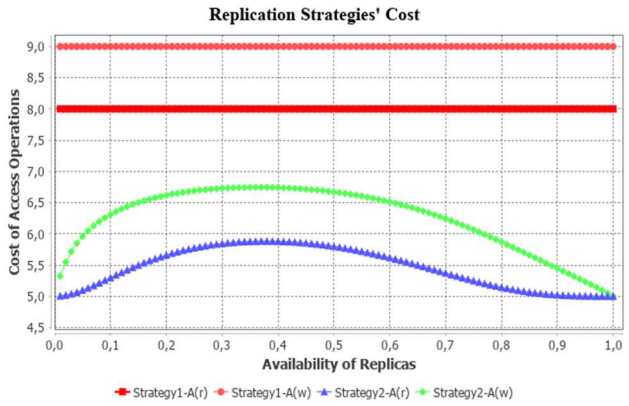

Figure 8: Cost, MCS vs. hybrid DRS (16 replicas)

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

B. Scenario 2

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.

$$p = 0.7;\ \varepsilon = 16;\ \alpha = 0.90;\ \beta = 0.90;$$
$$\gamma = 4.0;\ \delta = 4.0;\ fw = 0.7;$$

Having kept the system parameters μ and λ to the same values of six and 15, respectively, on a mutation probability of 0.2, the system is run. Figure 9 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 the objectives.

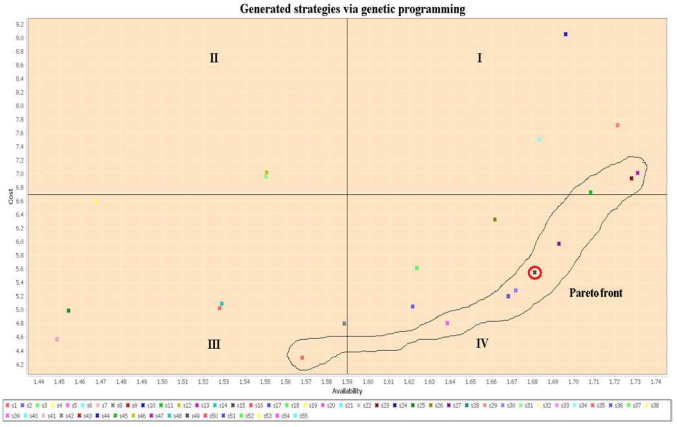

Fig 9. A Pareto front view for scenario 2

The Pareto front for scenario 2 shows some of the solution DRSs getting closer to an 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 one. For the specified scenario, the system takes three generations to come up with an optimized solution. Considering the trade-offs, Figure 10 represents the 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.

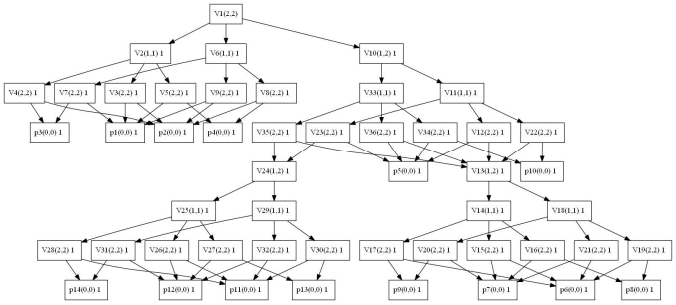

Fig 10. An optimized DRS for scenario 2

Figure 11 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 line (with circles) indicates the availability of the write operation. It can be seen that the availabilities are good, too, but most importantly, the cost of the access operations… noticeably low.

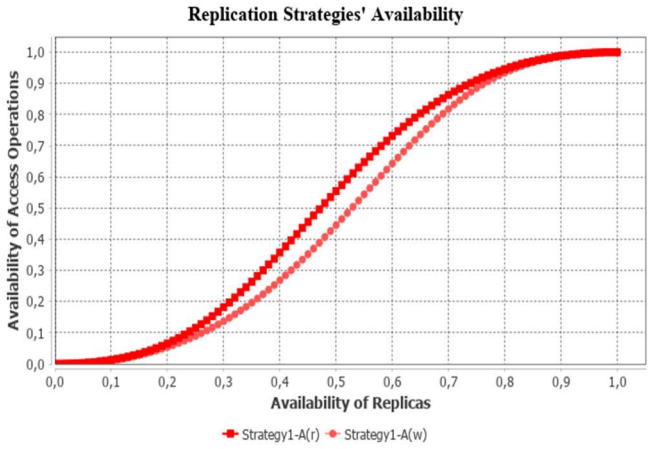

Fig 11. Availability of the chosen DRS

Figure 12 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.

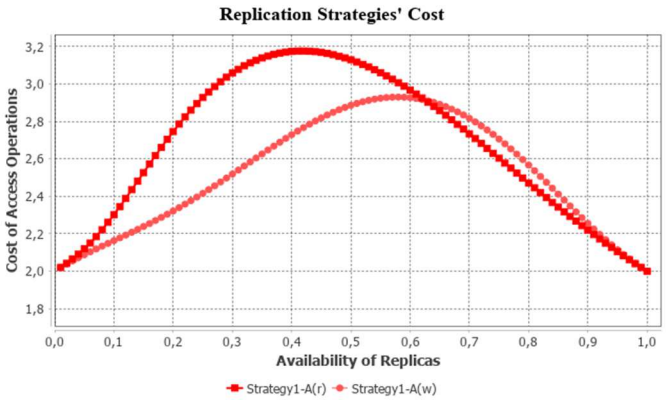

Fig 12. Cost of the chosen DRS

The crossover points for DRSs do matter and affect the values of objectives. Another prospective solution from the Pareto front is shown in Figure 13, where the same building blocks are combined by the GP, but slightly differently than in Figure 10. However, it has significantly increased the availability of the access operations by slightly compromising on the cost, but not being too heavily either.

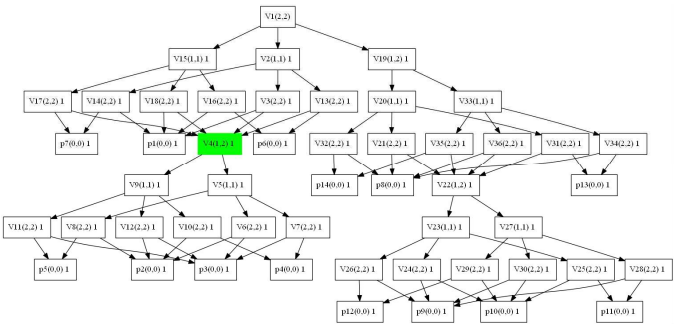

Fig 13. Another optimized DRS with a slightly different crossover point

Figure 14 presents a comparison between the availabilities of both strategies on the discretized values of p. The symmetry of the graphs indicates 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 13).

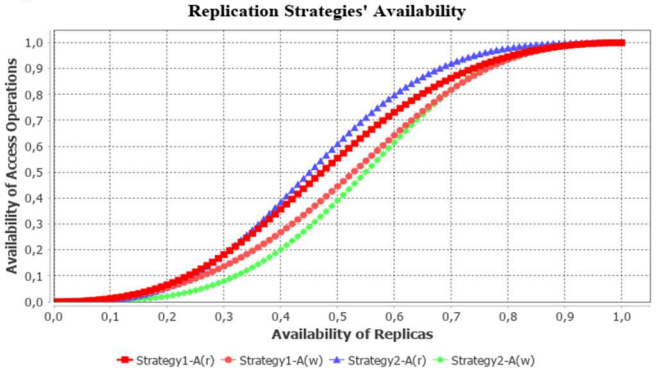

Fig 14. Availability comparison of the two Pareto front solutions

Figure 15 presents a zoom-in view of the operation availabilities on higher values of p. 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.

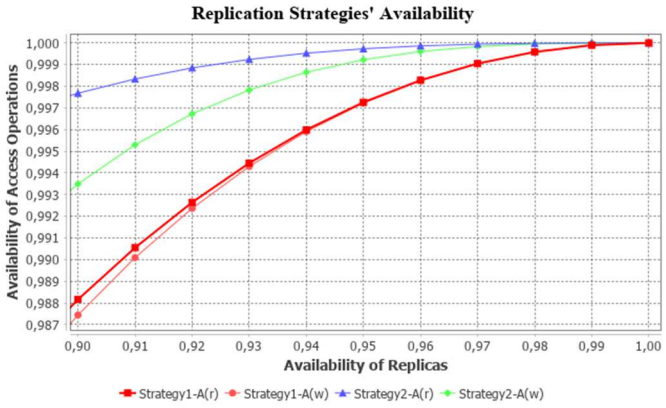

Fig 15. Zoom-in view of the comparison

Figure 16 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.

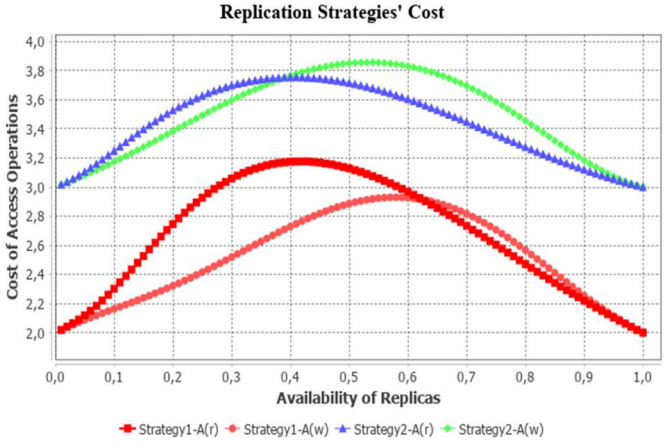

Fig 16. Cost comparison of the two Pareto front solutions

Hence, our machine learning mechanism efficiently combines replication strategies as a single voting structure to achieve the specified objectives 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 innovative, 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.

VII. CONCLUSIONS

The paper combines the concepts of replication and genetic programming and provides a mechanism based on genetic

programming to automatically design and generate innovative hybrid solutions in the form of unknown DRSs. It addresses a non-trivial multi-objective optimization problem of DRSs where, no single solution exists that simultaneously optimizes each objective. The proposed machine learning framework gains leverage through the voting structure to generate application-optimized replication strategies by exploiting their heterogeneity. 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. It tries to reduce the cost while not comprising on the availability too much. This automatic approach provides opportunities to discover new optimized replication strategies which are up-to-now unknown.

As for future work, we intend to introduce multi-crossover and multi-mutation operators with more complex system parameter settings, which will give us some more fine-grained control over the algorithm in anticipation of designing appropriate solutions, accordingly.

REFERENCES

[1] M. Naor and A. Wool, "The Load, Capacity, and Availability of Quorum Systems," SIAM Journal on Computing, vol. 2, issue 2, pp. 423-447, 1998.

[2] R. Jimenez-Peris, M. Patino-Martınez, G. Alonso, and B. Kemme, "How to Select a Replication Protocol According to Scalability, Availability, and Communication Overhead," in Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems (SRDS), 2001.

[3] K. Miettinen, "Nonlinear Multi-objective Optimization," Kluwer Academic, Boston, 1999.

[4] P. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems," ISBN-13 978-0201107159, Addison Wesley, p. 370, 1987.

[5] O. Theel and H. Pagnia, "Optimal Replica Control Protocols Exhibit Symmetric Operation Availabilities," in Proceedings of the 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 252-261, 1998.

[6] S.M.A. Bokhari and O. Theel, "A Flexible Hybrid Approach to Data Replication in Distributed Systems," Computing Conference (SAI), London, UK, 2020 (to be published).

[7] P. Bernstein and N. Goodman, "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases," ACM Transactions on Database Systems (TODS), vol. 9, issue 4, pp. 596–615, 1984.

[8] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Transactions on Database Systems, vol. 4, issue 2, pp. 180–207, 1979.

[9] 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), pp. 243–254, 1990.

[10] D. K. Gifford, "Weighted Voting for Replicated Data," in Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP), pp. 150-162, 1979.

[11] A. Kumar, "Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data," IEEE Transactions on Computers, vol. 40, issue 9, pp. 996-1004, 1991.

[12] S. Y. Cheung, M. Ammar, and M. Ahamad, "The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data," IEEE Transactions on Knowledge and Data Engineering, vol. 4, issue 6, 1992.

[13] C. Wu and 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), IEEE Computer Society Press, 1992.

[14] M. Arai, T. Suzuki, M. Ohara, S. Fukumoto, K. Iwasak, and H. Youn, "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), pp. 143-150, 2004.

[15] S. C. Choi and H. Y. Youn, "Dynamic Hybrid Replication Effectively Combining Tree and Grid Topology," The Journal of Supercomputing, vol. 59, issue 3, pp. 1289-1311, 2012.

[16] O. Theel, "Rapid Replication Scheme Design using General Structured Voting," in Proceedings of the 17th Annual Computer Science Conference, Christchurch, New Zealand, pp. 669-677, 1994.

[17] 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), Brisbane, Australia, pp. 530-535, 1998.

[18] O. Theel, "General Structured Voting: A Flexible Framework for Modelling Cooperations," in Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS), pp. 227-236, 1993.

[19] J. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection," MIT Press, Cambridge, 1992.

[20] W. Banzhaf, F. Francone, R. Keller, and P. Nordin, "Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications," Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[21] G. Syswerda, "Simulated Crossover in Genetic Algorithms," In Foundations of Genetic Algorithms (FOGA), pp. 239–255, (1992).