GNSGA: A Decentralized Data Replication Algorithm for Big Science Data

1st Xi Wang
Tennessee Technological University
xwang45@tntech.edu

2nd Xusheng Ai
Clemson University
xai@clemson.edu

3rd F. Alex Feltus
Clemson University
ffeltus@clemson.edu

4th Susmit Shannigrahi
Tennessee Technological University
sshannigrahi@tntech.edu

***Abstract*—Domain science applications in fields such as Genomics and High-Energy Particle Physics use geographically distributed data federations for publishing and accessing datasets. Data is typically replicated among data federation nodes to improve efficiency and fault tolerance. While replication strategies are well documented in distributed database instances (e.g., Apache Cassandra), replication among distributed data storage nodes can be ad-hoc. Replication over wide area networks can also require global coordination (or global shared state) which is not ideal when multiple organizations are involved.**

In this paper, we introduce GNSGA, which stands for Greedy Non-dominated Sorting Genetic Algorithm II. It is an optimization algorithm that combines greedy and non-dominated sorting genetic algorithms to solve multi-objective optimization problems. The "greedy" aspect of the algorithm refers to the use of a greedy strategy in the selection of nodes, while the "Non-dominated Sorting Genetic Algorithm II (NSGA-II)" is a fast non-dominated multi-objective optimization algorithm with an elite retention strategy. Replication decisions in GNSGA are based on the local properties and resource availability of the data storage nodes. By incorporating Greedy and NSGA-II algorithms, GNSGA optimizes multiple conflicting objectives to satisfy replica placement constraints such as cost, time, and storage capacity.

We compared GNSGA with popular replica placement strategies, such as closest node replication, shortest transfer time, and a Particle Swarm Optimization (PSO)-based replication algorithm. We performed simulations and an actual deployment on the NSF's FABRIC testbed for evaluation. The results demonstrate that GNSGA consistently selects nodes to reduce replication time by 5.8%-15.4% while satisfying replication constraints (i.e., cost, time, and storage). We also show that GNSGA is beneficial for replicating large files over wide area networks.

***Index Terms*—replication, multi-objective optimization, distributed federation**

I. Introduction

Data-intensive science communities such as genomics and high-energy particle physics generate and utilize content from geographically distributed facilities. To facilitate availability and performance, these datasets are often replicated [7] [4]. Such replication decisions can be based on various parameters such as the distance between nodes, storage capacity, and/or operational knowledge, and often require coordination between nodes. While a replication approach based on centralized decision works well in data centers or environments with a single administrative domain, scientific data federations can have nodes under different administrative domains. Further, nodes may have different storage capacities and bandwidths which makes replication decisions difficult.

We are currently developing a distributed and federated data repository (Hydra) for large scientific datasets [16] based on Named Data Networking (NDN) [24]. It is built as a federation of geographically distributed heterogeneous storage nodes in which researchers can publish scientific datasets from anywhere. The system automatically replicates these datasets to other nodes to maintain the desired degree of replication. The nodes in the federation can be very diverse in terms of resource availability - they may have different hardware, security policies, and available storage and bandwidth, and the conditions are constantly changing. Efficient replication of content in such a dynamic system requires satisfying several conflicting constraints (e.g., storage availability, bandwidth, cost) while identifying the ideal replication candidates. Moreover, a node in the data federation needs to have the option to specify its own preferences (policy based), making it difficult to apply contemporary replication mechanisms.

In this work, we propose a novel algorithm, GNSGA, which takes into account the local conditions and preferences of the nodes, such as storage capacity, bandwidth, and cost of replicating files. The replication decision is completely local and does not require coordination with other nodes. While we present GNSGA in the context of the storage federation we mention above, the algorithm is generic and can be applied to most distributed systems.

In this work, we also introduce a novel parameter - *Favor*. *Favor* is a numerical value that summarizes a node's local conditions and replication preference. In a federated system with multiple nodes, the nodes with the highest Favor values replicate the files that are below the desired degree of replication. If the desired degree of replication of a file X is 3 and its current replication degree is 2, the node in the federation with the highest Favor value replicates this file. Once the replication is complete, the node may change its Favor to reflect new conditions, such as reduced storage capacity.

In our work, we show that instead of replicating content to the closest node or the node with the lowest round trip time (RTT), Favor-based replication can improve the performance of the entire system by taking into account resource variations in federated nodes.

GNSGA is a new replication strategy that can be applied to any distributed system that requires replication. Contrary to the

ISBN 978-3-903176-57-7

classical choices in data replication, GNSGA demonstrates that instead of replicating content to the closest node or the node with the lowest RTT, Favor-based replication can improve the overall system performance by considering resource variations in federated nodes. We also find that when replication involves datasets with multiple files, such as those found in big science applications, the order in which individual files are replicated is important to overall system performance. In our experiment, we find that the optimal replication order involves replicating files in non-decreasing sizes. Specifically, when multiple files exist, the algorithm first confirms that the sum of the replicated file sizes is less than the capacity of the nodes and replicates the smaller files first to achieve the optimal order. However, in the event that all available nodes do not have enough storage to replicate all files, we consider sorting the multiple files from smallest to largest (non-decreasing) and replicating as many files as possible until the node capacity is reached, then replicating the overflow files to the node with the second highest favor value, rather than selecting multiple nodes at the same time. This is because replicating as many files as possible to one node is a simpler process to implement and manage than replicating to multiple nodes at the same time. This reduces the complexity of the system and reduces the risk of errors and failures.

Once replication to a certain node is completed, GNSGA dynamically tunes the replica placement strategy to adapt to changed conditions. Our experimental results show that the GNSGA selected nodes can reduce replication time by 5.8%-15.4% compared to traditional replication mechanisms. We also show that the speed of replicating large files is more significantly improved while meeting storage constraints.

II. Background and Related work

A. Data Replication in Federated Systems

There is a large body of previous work on data replication. In the context of federated systems where nodes are connected over wide area networks, the most common data replication strategies focus on shortest distance [17], lowest cost [12], or random selection [18].

In addition to replication based on one objective such as distance, previous works have also investigated replication aimed at satisfying multiple objectives. For example, Hassan et al. [2] proposed a Multi-Objective Evolutionary (MOE) algorithm with latency, system reliability, and storage as the three objectives to be optimized, and used NSGA-II [5] to find a set of compromise solutions to conflicting goals. However, the algorithm does not consider two key metrics: cost and replication order.

Long et al. [13] proposed a Multi-objective Optimized Replication Management (MORM) algorithm based on an artificial immune algorithm to optimize five objectives: mean file unavailability, mean service time, load variance, energy consumption, and mean access latency. However, MORM simply linearly weighted the five objective functions and transformed the multi-objective into a single objective, making it difficult to evaluate the actual effectiveness of the solution in reality.

In the paper describing the SPLAD approach [19], three node selection policies are mentioned, namely the random selection of nodes, the selection of less charged nodes, and the policy of selecting the less loaded node from two randomly selected nodes. The paper mainly evaluates the impact of global storage load and data copy placement on data loss ratio and storage load distribution.

Guerrero et al. [9] used NSGA-II to optimize the total file failure rate, mean latency time, and migration cost. However, most of the experiments were conducted under node homogeneity, an idealized environment that is not suitable for operation in the real scenario.

Replication strategies based on the Particle Swarm Optimization (PSO) algorithm are widely used because of their simplicity, computational convenience, and fast solution speed [8] [14] [15]. However, the disadvantage of PSO is that it is unfavorable for discrete optimization problems and easily falls into a local optimum, so it often needs to be mixed with other algorithms to obtain better results for federated systems.

B. A distributed storage framework

As mentioned earlier, we are building "Hydra" [16] - a secure, distributed, and decentralized federation of storage servers (nodes) provided by scientific communities. Upon publication of a file/dataset into this federation, the system automatically replicates data to ensure availability in the face of node or network failure. The system runs over NDN and utilizes the State Vector Sync (SVS) [**?**] protocol that lets individual nodes maintain a "global view" of the system. In the replication process, SVS is used to maintain a specific degree of replication to ensure that datasets are available even if individual servers fail. Specifically, files are replicated to nodes based on their available resources and preferences, and the replication algorithm takes into account the constraints of each node. While it is possible to replicate files randomly in the federation, a more optimal replication approach is used to ensure that nodes with different resource capacities and preferences are properly utilized. Our study on decentralized replication in the context of this federation. However, all algorithms we propose here are generic.

C. FABRIC

FABRIC is an everywhere-programmable national instrument consisting of scalable network elements equipped with extensive computation and storage, interconnected by high-speed dedicated optical links [3].

For this work, we utilized the FABRIC testbed to deploy and test our algorithm. We completed the deployment of the federation by configuring nodes on the FABRIC testbed. We used the FABRIC API [1] to request a slice containing multiple nodes. The nodes were set up with default components: 4 CPU cores, 16 GB RAM and 10 GB SSD storage. Layer 2 networks were created between these nodes. After completing the basic VM deployment, we installed the necessary packages on each

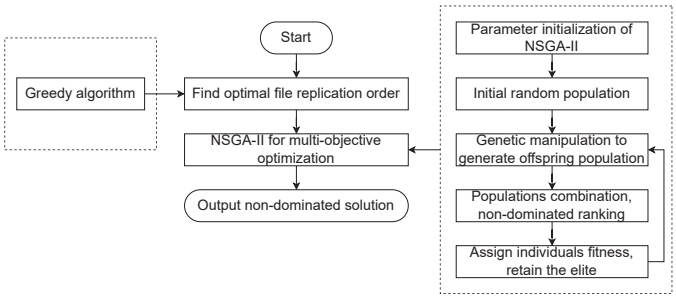

Fig. 1. Flow chart of multi-objective optimization algorithm based on greedy and NSGA-II.

node and used GNSGA to replicate data between these nodes. Note that this work does not make the replication decision in real-time. We are currently working on integrating GNSGA with Hydra.

III. GNSGA

The GNSGA algorithm is dedicated to the study of replication technology in distributed systems, with the main goal of enhancing high reliability and availability. The algorithm can significantly reduce replication time and meet the dynamic needs of users according to object size, storage, and server distribution with the advantages of low latency and low overhead. GNSGA first uses the greedy algorithm to find the optimal file replication order and then applies NSGA-II to select nodes at minimal cost, resulting in a fast multi-objective optimization algorithm. The basic flow of the algorithm is shown in Figure 1.

A. Finding the optimal replication order

As we mentioned earlier, the main purpose of using the greedy algorithm in GNSGA is to find the optimal file replication order and then select the nodes according to the minimum cost, and also to prepare for multi-objective optimization using the NSGA-II algorithm later.

In a real distributed storage deployment such as Hydra, there will be dozens of nodes across various geographical regions. When a dataset is published into the federation, three nodes are chosen from the available pool, assuming the nodes have enough storage to accommodate the data. The time required to replicate each file is denoted as N_i, and we assume that objects can be replicated on any node without interruption. For this work, we assume the degree of replication to be three. However, it can be more or less as dictated by a specific deployment.

The greedy strategy we use is to prioritize files with the shortest replication time. We assign data with the shortest replication time to the first free node, which ensures that the file with the shortest replication time is processed first, thus obtaining the shortest overall replication time.

For example, suppose we have n files to be replicated in an object, and each file replication time is t_i. To minimize the total replication time of n files, we can use a greedy algorithm to find the replication order of these n files. Note that the average replication time, i.e. $\frac{t_0+t_1+...+t_{i-1}}{n}$, is a fixed value, not necessarily the best measure of performance.

Consider a dataset containing three files, $a[i]$, $a[j]$, and $a[k]$, with replication times sorted in ascending order, i.e., $t[a[i]] < t[a[j]] < t[a[k]]$. Here, $[a[i]]$ in the first order can be replicated immediately without any wait time because $[a[i]]$ does not need to wait for others to finish replication. When replicating $a[j]$, the replication time is simply equal to the replication time of $a[i]$ since $a[j]$ replicates after $a[i]$. Similarly, when replicating $a[k]$, we need to account for the replication times of both $a[i]$ and $a[j]$ since $a[k]$ depends on both $a[i]$ and $a[j]$. Therefore, the replication time of $a[k]$ is the sum of the replication times of $a[i]$ and $a[j]$.

The total time to replicate all files is $2 * t[a[i]] + t[a[j]] + t[a[k]]$. Now, let's suppose we swap the order of $a[i]$ and $a[j]$. Since $t[a[i]] < t[a[j]]$, the total replication time after the swap becomes $2 * t[a[j]] + t[a[i]] + t[a[k]]$, which is longer than the previous time.

This example highlights that the order in which files are replicated can significantly impact the overall replication time in real-world systems with multiple storage servers and the need for file redundancy. This is especially important in fields like genomics, where replicating large datasets containing thousands of files across multiple nodes while minimizing the total replication time.

To address this challenge, a greedy algorithm that assigns files to nodes based on their replication time can be used. This algorithm prioritizes files with shorter replication times, leading to significant reductions in both time and search space. The goal of this approach is to efficiently find the best replication order and improve system performance.

In this work, we assume that there is no parallel transfer of files for replication. In the future, we plan to accommodate such scenarios in our algorithm.

B. Multi-objective optimization

Typically, the case of a single optimal solution does not generally arise for multi-objective problems, but rather a set of solutions available to the decision maker. Therefore, the use of NSGA-II (Non-dominated Sorting Genetic Algorithm II) [5] to further find the optimal nodes for replica placement with a guaranteed reduction in replication time and the cost is central to the $Favor$ computation. NSGA-II is improved from NSGA, which uses an elite-preserving strategy to guarantee the convergence of the algorithm and the quality of the Pareto-optimal set, which refers to the non-dominated set of the entire feasible decision space [20].

To minimize replication time and cost, we can use a multi-objective optimization function expressed as follows:

$$\begin{cases} F(favor) = min(f_1(favor), f_2(favor)) \\ favor = (cost, time) \in R^2 \end{cases} \tag{1}$$

where $F(favor)$ contains two objective functions; $f_i(favor)$ $(i = 1, 2)$ is the i-th objective function; $favor$ is the solution vector with two-dimensional variables $cost$ and $time$; and R^2 is the space of decision variables.

To determine Pareto domination, $\forall cost, time \in favor$, if $f_i(cost) \leq f_i(time)(i = 1, 2)$ and if $\exists i \in \{1, 2\}$ such that $f_i(cost) < f_i(time)$, then we say that $cost$ dominates $time$, expressed mathematically as $(cost \succ time)$, where $cost$ is non-dominated and $time$ is dominated.

It's worth noting that when dealing with three or more objectives, NSGA-III [6] is recommended over NSGA-II used in GNSGA. NSGA-III is a reference point-based many-objective optimization algorithm that emphasizes non-dominant solutions close to a set of provided reference points. It's applicable to multi-objective test problems with 3 to 15 objectives. However, we do not utilize NGSA-III for our algorithm and will investigate it in a future study.

C. Time Complexity

This section analyzes the complexity of the GNSGA algorithm. We first call the sort() function according to file size with average time complexity of $O(NlogN)$, and the time complexity of the for loop to find the optimal solution according to the greedy strategy is $O(N)$. Therefore, the time complexity of the Greedy algorithm is $O(NlogN)$. For NSGA-II, the time complexity consists of three parts: fast non-dominated sort: $O(M(2N)^2)$; calculating the crowding distance assignment: $O(M(2N)log(2N))$; constructing the partially ordered set (sorting): $O(2Nlog2N)$; So the total time complexity is $O(MN^2)$ [5]. Therefore, the time complexity of the GNSGA algorithm is $O(NlogN + MN^2)$,), where M is the number of objectives and N is the number of files.

IV. Replication Model

A. Replication model description based on GNSGA

The node with the files to be replicated (hereafter referred to as the client node) directs replication requests to the appropriate nodes based on the Favor of the other nodes and the location and load of each node. We assume this node ingests the files from the publisher and make is available for replication to the other nodes in the federation. In this work, we utilize a large FABRIC topology spanning ten geographic regions in the US. The ten nodes are located in MAX (University of Maryland), STAR (StarLight), UTAH (University of Utah), TACC (UT Austin), MICH (University of Michigan), NCSA (UIUC), DALL (Dallas), SALT (Salt Lake City), WASH (Washington DC), and MASS (Massachusetts), and have different distances from the client node. For more details, see table I.

To visualize the problem, the client node and other nodes are generated on one coordinate, based on the actual latitude and longitude. The client node is the node with file(s) replication requests. In the simulation experiments, two different coordinates are set for the client node of the experiments, as detailed in Section V.

The number of files with replication requests in the client node is N, and the corresponding file size is denoted as C_N_i, where $i = 1, 2, ..., N$. Other nodes are named according to their location, the number of nodes is denoted as S, and the storage capacity of each corresponding node is denoted as C_S_j, where $j = 1, 2, ..., S$.

TABLE I
NODES ARE SELECTED BASED ON THE SHORTEST DISTANCE.

Nodes	Closest Node	Distance (km)
MICH	STAR	326.15
UTAH	SALT	5.09
TACC	DALL	280.84
WASH	MAX	24.50
NCSA	STAR	206.64
DALL	TACC	280.84
MAX	WASH	24.50
MASS	MAX	511.78
SALT	UTAH	5.09
STAR	NCSA	206.64

It's important to note that GNSGA replicates files rather than chunks, which simplifies the replication process and reduces overhead. Otherwise, GNSGA would need to account for the relationship between the chunk and files to ensure that each file is fully replicated. In addition, the size of the chunks would need to be carefully chosen to balance replication overhead and transmission efficiency.

In a distributed system with many designs such as load balancing, web server, application code, database server, etc., the instability of any node will affect the availability of data. When using GNSGA, M can be set to 4 when the desired replication degree is 3. This means that the 4 nodes with the highest favor value ranking voluntarily replicate the data and the 4th node is the alternate/backup node. When the communication between nodes fails and the service becomes unavailable, GNSGA automatically selects the node with the fourth-highest favor value for the replication operation.

B. Objectives and Constraints

The main focus of our optimization is to reduce the time and cost of file replication between nodes, and the notations used in the GNSGA algorithm are explained in Table II. In the above context, the model for selecting nodes to place replicas by $Favor$ can be described as:

where the objective function (2) seeks to minimize the replication time (network time + read/write time) and (3) seeks to minimize the replication cost.

Constraint (4) requires that the total size of files replicated by a node does not exceed the node storage limit.

A "1" in constraint (5) means that the i-th file is replicated on the j-th node, and 0 means that the i-th file is not replicated on the j-th node, i.e., it is not selected.

Constraint (6) indicates a node can replicate up to N files.

Constraint (7) implies that M nodes (M replicas) are required for file replication.

$$f_1(favor) = \sum_{i=1}^{N}\sum_{j=1}^{S} C\_N_i/(\alpha \cdot (D_i \cdot x_{ij}/RTT)) + \sum_{i=1}^{N} C\_N_i/2v \tag{2}$$

$$f_2(favor) = c_1 + c_2 \tag{3}$$

TABLE II
MEANING OF THE NOTATIONS

Notation	Meaning
N	Number of files to be replicated
C_N_i	Size of different files (GB)
α	Bandwidth set (Gbps)
M	Number of replicas
S	Number of nodes
C_S_j	Storage capacity allocated to files per node (GB)
D_i	Distance from the client node to the i-th nodes (km)
ν	File read/write speeds (GB/s)
c_1	Cost of bandwidth per month ($)
c_2	Cost of read/write speed per month ($)

s.t.

$$\sum_{i=1}^{N} C\_N_i \cdot x_{ij} \leq C\_S_j \tag{4}$$

$$x_{ij} = \begin{cases} 1 \\ 0 \end{cases} \tag{5}$$

$$\sum_{i=1}^{N} C\_N_i \leq N \tag{6}$$

$$\sum_{j=1}^{S} x_{ij} = M \tag{7}$$

C. Replication Benchmarks

When discussing replica placement, the first thing that comes to mind is choosing the closest node for replication, perhaps the fastest. However, ignoring distant but potentially less costly nodes and forcing the closest node to replicate may waste a lot of storage space and bandwidth.

The simple formula of $time = distance/speed$ tells us that to get a faster time, we need to reduce distance. Servers of different physical distances show that servers closer together do have better latency performance. Since each of the 10 nodes has its own latitude and longitude, to find the shortest physical distance between two nodes, all we have to do is to calculate the spherical distance between the two nodes.

In our proof of concept experiment, we use two distance formulas, the Vincenty formula [22] and the Haversine formula [21]. In the case of considering only the distance between two nodes, we use the Vincenty formula, which is an ellipsoidal model with several iterations and high theoretical accuracy. The Haversine formula will be applied to GNSGA and the upgraded PSO algorithms, because the Haversine formula is faster than Vincenty and more suitable for distance calculation of a large number of nodes, and although the precision is not as high as Vincenty, the error between these two formulas is completely negligible at 64-bit floating point precision.

In addition to the closest node as the benchmark, we also chose the node with the fastest transfer rate. Table III shows the average transfer rates for three file objects of 100 MB, 1 GB, and 10 GB respectively. Each node took turns as a client, and after storing the inserted file, the other 9 nodes acted as servers to replicate the inserted file, running the replication command three times and taking the average and standard deviation. Table III also shows the node with the fastest transfer rate corresponding to each node, e.g., for all three file sizes, the node with the fastest transfer rate to MICH is STAR.

The purpose of Table I and III is to evaluate the GNSGA algorithm compared to the replication case without the algorithm. Also, we observed that the closest node is not necessarily the fastest node due to latency or bandwidth factors. The closest node to DALL is TACC, and the fastest node for DALL to transfer objects is STAR; similarly, the closest node to MASS is MAX and the fastest node to MASS is WASH in this experiment.

TABLE III
FASTEST NODE TRANSFER RATE (GBPS) FOR DIFFERENT OBJECT SIZES.

File Size	MICH		UTAH		TACC		WASH	
	avg	std	avg	std	avg	std	avg	std
100 MB	2.15	0.74	14.13	2.35	0.237	0	5.30	0.21
1 GB	3.86	1.04	14.10	4.16	0.277	0.58	8.60	0.10
10 GB	2.80	0.32	11.60	1.35	0.281	0.58	9.30	0.02
Fastest node	STAR		SALT		DALL		MAX	
	NCSA		DALL		MAX		MASS	
100 MB	3.04	0.25	0.94	0.12	5.30	0.21	0.15	0
1 GB	5.25	1.50	1.17	0.05	8.60	0.10	0.17	12.17
10 GB	3.45	0.34	1.22	0.01	9.30	0.02	0.16	13.01
Fastest node	STAR		STAR		WASH		WASH	
	SALT		STAR					
100 MB	14.13	2.35	3.04	0.25				
1 GB	14.10	4.16	5.25	1.50				
10 GB	11.60	1.35	3.45	0.34				
Fastest node	UTAH		NCSA					

D. Upgraded PSO

Particle Swarm Optimization (PSO) [11] is a swarm intelligence algorithm inspired by the learning of flocking of birds or fish, which randomly generates an initial population and assigns a random velocity to each particle, and dynamically adjusts the velocity and particle trajectory during flight based on their own and their peers' flight experience. The standard PSO algorithm mainly suffers from the early convergence problem, i.e., the algorithm stops prematurely at the local optimum and does not perform global exploration well. To avoid early convergence, other algorithms can be added to improve the performance of the PSO algorithm.

In the upgraded PSO, we marked the nodes in a two-dimensional coordinate system according to their actual latitude and longitude, the same as we did in GNSGA. The concept of "center" means that the central particle is located at the center of the population in each iteration [23]. We applied the center to the PSO solution space based on physical distance. The upgraded PSO algorithm not only calculates the particular position of the population center but also dynamically adjusts the weights according to the fitness of individual particles [10], which more effectively solves the problem that the PSO algorithm falls into local optimum at the late stage of diversity reduction.

TABLE IV
NODES READ/WRITE SPEED AND MONTHLY COST

Nodes	Monthly storage cost ($)	w/r speed (GB/s)
MICH	12	485
UTAH	11	380
TACC	12	483
WASH	5	256
NCSA	5	280
DALL	7	298
MAX	7	339
MASS	7	291
SALT	9	377
STAR	9	345

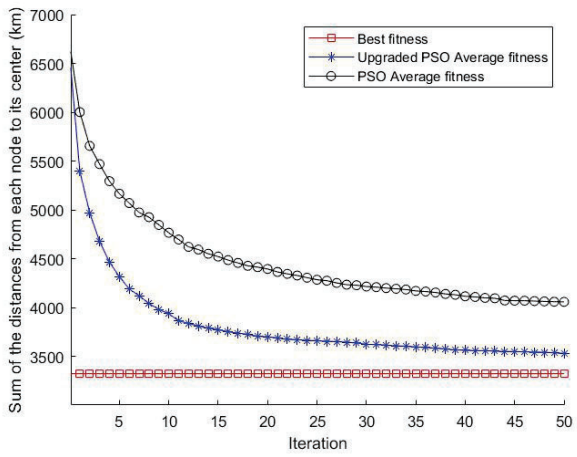

Fig. 2. Particle swarm optimization results for node selection.

To analyze the performance of the proposed GNSGA algorithm for solving the optimal node problem, we conducted tests on Fabric (see Section V-B2 for more details) and compared the results with the upgraded PSO. Table IV shows the different read/write speeds and relative monthly costs. The purpose of using these parameters is to select nodes with a lower cost while maintaining the replication speed.

In ten nodes, we used the standard and upgraded PSO algorithms. After 50 iterations, the average fitness and the best fitness of the population change as shown in Figure 2. The parameters are as follows: population size p = 1000 , numbers of center o = 3, inertia weights w_{max} = 0.2, w_{min} = 0.1, learning factors $c1$ = 0.4, $c2$ = 1.6, and number of iterations $gmax$ = 50. Compared with the standard PSO, the upgraded PSO has significantly better fitness, because the upgraded PSO can dynamically adjust the values of inertia weights and learning factors in the algorithm, which is conducive to balancing the global search and local search abilities of the algorithm and thus improving the performance. However, we note that GNSGA works even better than the upgraded PSO when tested on FABRIC, see Section V-B2 for details.

V. SIMULATION AND EXPERIMENTS

Given that the research in this paper only addresses the issue of replica placement in distributed systems, we used random node selection as the baseline for our experiments, and GNSGA was originally designed to calculate the Favor value in the distributed system, which refers to the suitability of a node to host one or more files. An overview of distributed storage framework can be found in Section II-B.

A. Simulation

1) Simulation setup: To understand how GNSGA behaves, we first simulate our algorithm in MATLAB. We first run periodic measurements on the NSF FABRIC platform [3] and record RTT, bandwidth, storage, and other parameters. We also estimate the prices of hardware and operational costs. Note that these are not actual numbers but our best guesses from vendor quotes and public information. A partial survey shows that the market price of storage used in the testbed ranges from $20,375.81 to $30,260.08. We chose a price range of $15,000-$35,000, which costs $250-$583 per month based on an average storage lifetime of 5 years, corresponding to an average read/write speed range of 5 GB/s - 12 GB/s, and then generated random numbers and read/write speeds for the one-month price range, which were assigned to each of the 10 nodes. We will reiterate that the numbers are only to demonstrate the utility of our algorithm. The exact values do not matter in our comparison.

This experiment is divided into two groups. In the first group, the node closest to the client node is selected as the initial comparison for this experiment, with the node selected by the GNSGA algorithm to clarify whether the algorithm proposed in this paper has the expected performance; in the second group, we compare the execution results of the node with the lowest cost (the cheapest node) and the node selected by GNSGA.

As shown in (8) below, we chose the average file replication time as an evaluation criterion in order to compare the differences between those algorithms.

$$\overline{T} = \frac{\sum_{n}^{i=1} T_i}{n} \tag{8}$$

Where, T_i is the transfer time of the i-th file, and n is the total number of files to be replicated. The average file transfer time is the ratio of the total time to replicate all files to the total number of files to be replicated. For users in a distributed system, the average file transfer time is an important metric for evaluating the merits of the replica placement algorithm. The shorter the average transfer time, the higher the Favor, which means that the selected node outperforms other nodes. According to Table I, we use the node closest to the client node to calculate the average transfer time and compare it with the node selected by GNSGA.

In addition to reducing the replication time, reducing the storage cost is also one of the goals of the GNSGA algorithm. We let the client node chooses the lowest cost node based on different storage prices, and then compare the average file transfer time between the lowest cost node and the node chosen by the GNSGA algorithm.

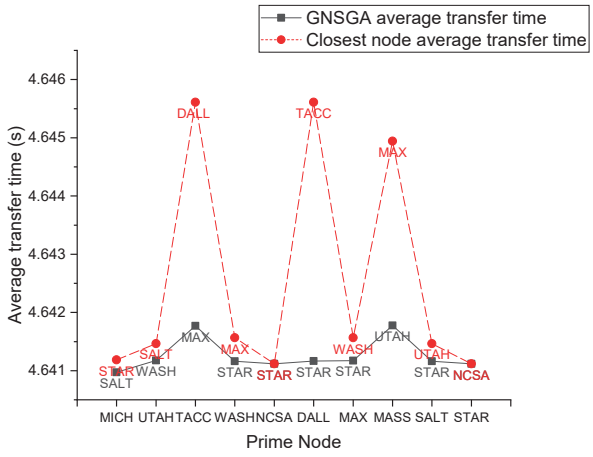

Fig. 3. Average transfer time of closest node and GNSGA selected node

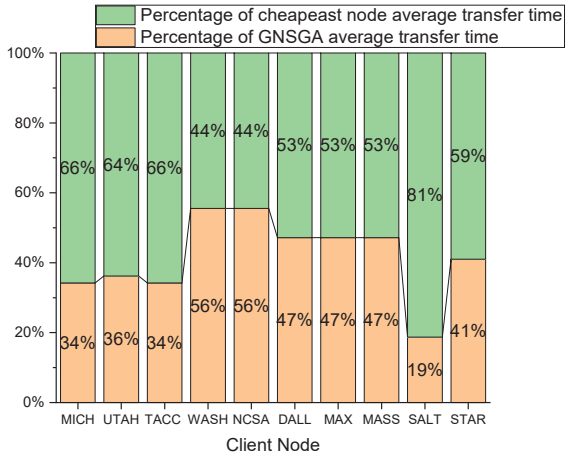

Fig. 4. Average transfer time percentage of GNSGA selected node and cheapest node

2) Simulation results analysis: With a bandwidth of 100 Gbps and a read/write speed of 6.25 GB/s, we transferred 10 files with a total file size of 14.85 GB. As shown in Figure 3, the nodes selected by the GNSGA algorithm (solid line) are slightly better than the nodes closest to the client (dashed line) in terms of average file transfer time. Even with the same bandwidth and the same read/write speed, the incurred delays are different, such as slow command execution, network delays, node delays, etc. Especially in the case of limited node capacity, the problem becomes more complicated. It does not matter if the data flow is jammed once or twice, but if it is frequently jammed, the link bandwidth will not be utilized at all, so it is necessary for GNSGA to consider the delay problem.

Figure 4 uses 100% stacked bars, splitting each bar to show the transfer time of each selected node as a percentage in the case of transferring the same file. As shown in Figure 4, the nodes selected by the GNSGA algorithm show considerable performance at 100 Gbps bandwidth and 14.85 GB object size of 10 files, with an average transfer time reduction of about 80% compared to the cheapest node. Since GNSGA considers the balance of multiple parameters such as node distance, monthly cost, bandwidth, and read/write speed, the time spent in WASH and NCSA is slightly slower than simply comparing monthly costs. And when the node encounters high latency, such as SALT, GNSGA's replication time is significantly less than the cheapest node.

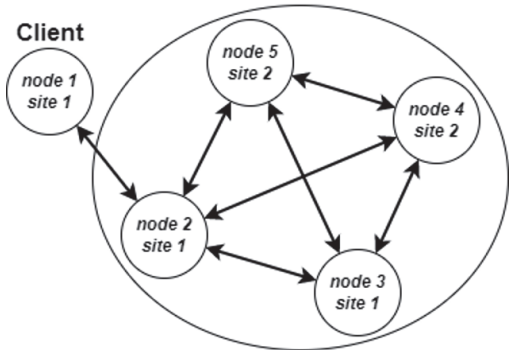

Fig. 5. FABRIC topology overview of the client and storage nodes.

B. Experiments on FABRIC

1) Experimental setup: To understand how the GNSGA behaves in realistic scenarios, we first implemented the design ideas of the GNSGA in MATLAB. Then we ran periodic measurements [3] on NSF FABRIC platform and recorded RTT, bandwidth, storage and other parameters. Then we set the same parameters as FABRIC on MATLAB and ran to get three replication nodes, which were then applied to FABRIC for replication and used iperf3 to get the replication time. We requested a slice in FABRIC containing five nodes, MICH, UTAH, TACC, WASH and NCSA. The five nodes took turns acting as the client node to transfer files. Figure 5 shows an example of node 1 as a client node and nodes 2-5 as storage nodes.

At last, we conducted experiments from 7 am to 9 am CST and 7 pm to 9 pm CST. For example, we first uploaded the file to node MICH, node MICH as the client, replicated the file to the node selected by GNSGA and random selection, entered the fabric interface on MICH, and ran the `iperf -s` command. Next, we enter the interface from the selected node and run `iperf -c x.x.x.x -F filename`, this command connects to the MICH with the IP address `x.x.x.x` and replicates the file for the selected node.

The replication times of randomly selected nodes and the GNSGA selected nodes were tested using two objects, 48.2 MB and 881.6 MB files. FABRIC has two bandwidths available for testing, 100 Gbps and 25 Gbps. We replicated each of the two objects three times at different time periods to make the experimental data more valid.

2) Experiment results analysis: As shown in Figure 6, we first performed replication at 100 Gbps bandwidth. When

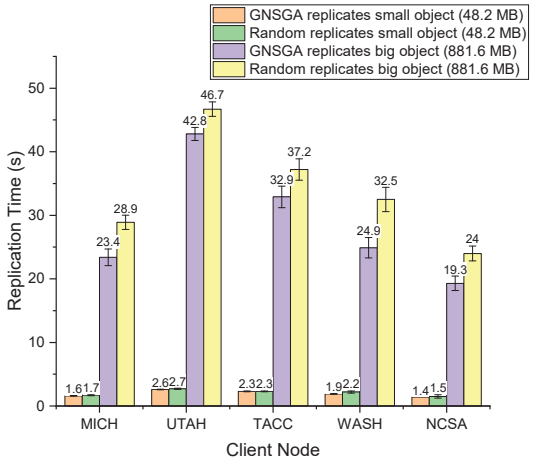

Fig. 6. Standard Deviation of the replication time.

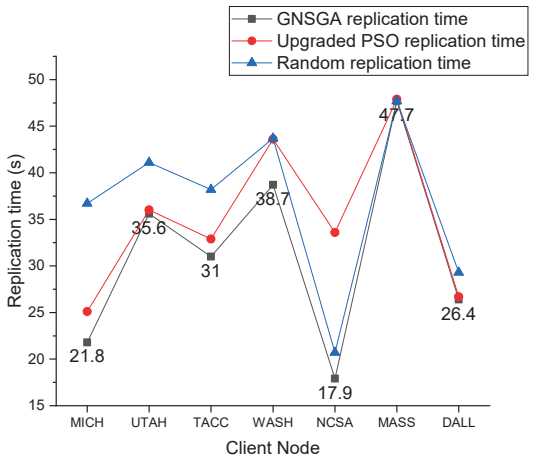

Fig. 7. Comparison of GNSGA, Upgraded PSO, and random selection

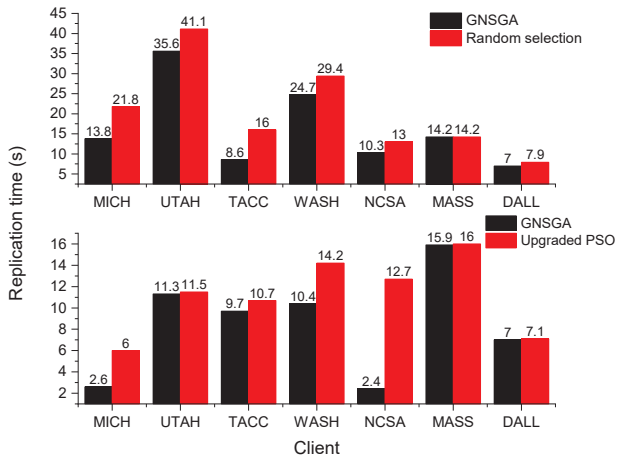

Fig. 8. Replication time after excluding same nodes.

replicating smaller objects, the difference in replication time between the GNSGA algorithm and random selection is not significant, and the performance is comparable; however, as the object size increases, the GNSGA algorithm significantly outperforms random selection. When the bandwidth is reduced to 25 Gbps, the experiment results still show that the overall replication time of the storage nodes with GNSGA is better than without this algorithm. By analyzing the data of the experiment results, it can be concluded that the replication time is reduced by approximately 5.8%-14.3% when using the nodes selected by GNSGA.

By analyzing the performance at both bandwidths, we clearly show that the GNSGA algorithm performs better than randomly selected nodes. Based on this, We requested another slice containing seven nodes, MICH, UTAH, TACC, WASH, NCSA, MASS, and DALL.

Comparative analysis was again performed at 100 Gbps bandwidth. When a node sends a replication request, the other six nodes are available for the upgraded PSO to replicate the object. As shown in Table IV, each node has a different read/write speed and relative storage price. With equal bandwidth, three fast and affordable nodes were selected. In Figure 7, as the number of nodes increases, GNSGA reduced the total replication time by about 14.8% compared to random selection and 10.9% compared to upgraded PSO. After excluding the same nodes selected by GNSGA and random selection, GNSGA and upgraded PSO, we note that in some cases, such as when MASS acted as a client in Figure 7, all three algorithms selected the same node for replication, yielding the same replication time, which occurred only once in the seven-node test, and we can deduce that when there are more nodes, the less often the three algorithms select the same node. Figure 8 indicates that although the reduction tends to slow down, the overall algorithm still outperforms random selection and upgraded PSO.

Finally, we conducted replication experiments for three different sizes of objects, 100 MB, 250 MB, and 500 MB. Figure 9 shows that when the size of the objects is relatively small ($\leq$ 100 MB), it can be observed that the replication times of the three algorithms do not differ much, with GNSGA performing slightly better, on average 0.7 seconds shorter than the random selection and 0.4 seconds shorter than the upgraded PSO. As the file size increases to 500 MB, the GNSGA algorithm takes on average 2.8 seconds and 1.9 seconds shorter than the random selection and upgraded PSO, respectively.

VI. Conclusion

In this work, we propose two novel ideas.First, we introduce *Favor* - a numerical value that indicates a node's preference for replication. While we currently utilize a single Favor value in this work, we plan to extend it to a per-namespace Favor value in the future. We also introduced GNSGA, a novel replication mechanism based on a combination of Greedy and NSGA-II to take advantage of both algorithms. GNSGA quickly finds the best file replication order using the greedy algorithm and

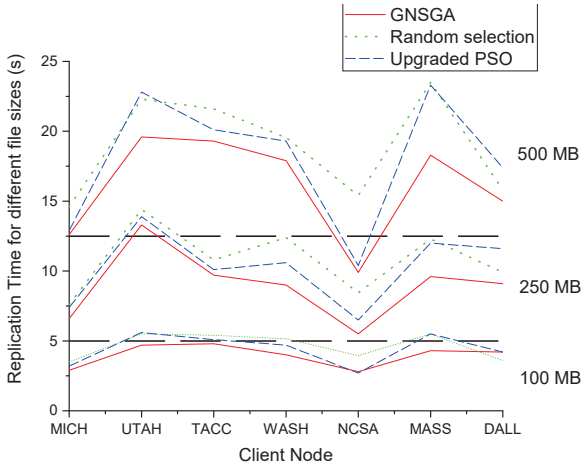

Fig. 9. Replication Time for different sizes

then finds the optimal set of non-dominated solutions using NSGA-II. GNSGA considers multiple conflicting goals, such as cost, latency, and bandwidth, and can be used in a federation of nodes with different amounts of resources. Through simulations and an actual deployment on FABRIC, we demonstrate that GNSGA can optimize multiple objectives and reduce replication time and cost. GNSGA outperforms other strategies such as replicating randomly, to the closest node, the node with the lowest latency, or PSO-based algorithms.

We are working on several improvements to this algorithm. First, we aim to investigate how the parallel transfer of files affects our algorithm. We also plan to incorporate this algorithm with Hydra and evaluate it in the context of real-world data replication. In addition, We aim to investigate how the use of network-layer multicast/broadcast affects the performance of our algorithm. Finally, we plan to utilize NSGA-III to improve the capabilities of GNSGA.

ACKNOWLEDGMENTS

This work has been supported by National Science Foundation Awards OAC-2019163, OAC-2126148, DGE-2043324, and OAC-2019012.

REFERENCES

[1] Github - fabric-testbed/fabrictestbed-extensions: Extensions for the fabric apui/cli. https://github.com/fabric-testbed/fabrictestbed-extensions. Accessed: 2023-01-30.

[2] Osama Al-Haj Hassan, Lakshmish Ramaswamy, John Miller, Khaled Rasheed, and E Rodney Canfield. Replication in overlay networks: A multi-objective optimization approach. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 512–528. Springer, 2008.

[3] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. Fabric: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing*, 23(6):38–47, 2019.

[4] Luca Cinquini, Daniel Crichton, Chris Mattmann, John Harney, Galen Shipman, Feiyi Wang, Rachana Ananthakrishnan, Neill Miller, Sebastian Denvil, Mark Morgan, Zed Pobre, Gavin M. Bell, Bob Drach, Dean Williams, Philip Kershaw, Stephen Pascoe, Estanislao Gonzalez, Sandro Fiore, and Roland Schweitzer. The earth system grid federation: An open infrastructure for access to distributed geospatial data. In *2012 IEEE 8th International Conference on E-Science*, pages 1–10, 2012.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[6] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[7] Alvise Dorigo, Peter Elmer, Fabrizio Furano, and Andrew Hanushevsky. Xrootd-a highly scalable architecture for data access. *WSEAS Transactions on Computers*, 1(4.3):348–353, 2005.

[8] Yalda Ebadi and Nima Jafari Navimipour. An energy-aware method for data replication in the cloud environments using a tabu search and particle swarm optimization algorithm. *Concurrency and Computation: Practice and Experience*, 31(1):e4757, 2019.

[9] Carlos Guerrero, Isaac Lera, and Carlos Juiz. Migration-aware genetic optimization for mapreduce scheduling and replica placement in hadoop. *Journal of Grid Computing*, 16(2):265–284, 2018.

[10] Bin Jiao, Zhigang Lian, and Xingsheng Gu. A dynamic inertia weight particle swarm optimization algorithm. *Chaos, Solitons & Fractals*, 37(3):698–705, 2008.

[11] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[12] Wenhao Li, Yun Yang, and Dong Yuan. A novel cost-effective dynamic data replication strategy for reliability in cloud data centres. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 496–502, 2011.

[13] Sai-Qin Long, Yue-Long Zhao, and Wei Chen. Morm: A multi-objective optimized replication management strategy for cloud storage cluster. 60(2):234–244, feb 2014.

[14] Amjad Mahmood. Replicating web contents using a hybrid particle swarm optimization. *Information processing & management*, 46(2):170–179, 2010.

[15] Sujaudeen Nannai John and TT Mirnalinee. A novel dynamic data replication strategy to improve access efficiency of cloud storage. *Information Systems and e-Business Management*, 18(3):405–426, 2020.

[16] Justin Presley, Xi Wang, Tym Brandel, Xusheng Ai, Proyash Podder, Tianyuan Yu, Varun Patil, Lixia Zhang, Alex Afanasyev, F. Alex Feltus, and Susmit Shannigrahi. Hydra – a federated data repository over ndn, 2022.

[17] Rashed Salem, Mustafa Abdul Salam, Hatem Abdelkader, and Ahmed Awad Mohamed. An artificial bee colony algorithm for data replication optimization in cloud environments. *IEEE Access*, 8:51841–51852, 2020.

[18] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[19] Véronique Simon, Sébastien Monnet, Matthieu Feuillet, Philippe Robert, and Pierre Sens. *SPLAD: scattering and placing data replicas to enhance long-term durability*. PhD thesis, inria, 2014.

[20] Ankur Sinha, Kalyanmoy Deb, Pekka Korhonen, and Jyrki Wallenius. Progressively interactive evolutionary multi-objective optimization method using generalized polynomial value functions. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[21] Wikipedia contributors. Haversine formula — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=1090892412, 2022.

[22] Wikipedia contributors. Vincenty's formulae — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Vincenty%27s_formulae&oldid=1081502579, 2022.

[23] Yang Xiaojing, Jiao Qingju, and Liu Xinke. Center particle swarm optimization algorithm. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 2084–2087, 2019.

[24] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.