A Replica Placement Method Based on Firefly Optimization (Postprint)

Authors: Li Jun, Hou Mengshu

Date: 2018-05-18T00:00:00+00:00

Abstract

In cloud computing environments, distributed storage systems typically employ replica technology to ensure availability and reliability, with placement strategy being a critical issue in replica technology. To address the problem of high replica access overhead in existing replica placement strategies, this paper proposes a replica placement algorithm based on discrete firefly optimization. Considering the impact of replica placement on user access performance, a mathematical model is established, the fitness function of firefly positions is calculated, and movement toward the maximum luciferin value (i.e., the optimal value) is performed to obtain appropriate replica placement nodes. The proposed method is evaluated through simulation experiments and compared with replica placement strategies based on ant colony algorithms. Experimental results demonstrate that the algorithm can select appropriate replica placement nodes, exhibits good convergence, and effectively reduces replica access overhead in storage systems.

Full Text

Replica Placement Method Based on Glowworm Swarm Optimization

Li Jun, Hou Mengshu
(School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)

Abstract: Cloud storage systems often adopt replica techniques to guarantee availability and reliability, and replica placement represents a key issue in this context. To address the problem of high access overhead associated with existing replica placement strategies, this paper proposes a replica placement algorithm based on discrete glowworm swarm optimization. Considering the impact of replica placement on user access performance, we establish a mathematical model to calculate the fitness function of glowworm positions, which move toward the maximum luciferin value (i.e., the optimal value) to obtain appropriate replica placement nodes. We evaluate the proposed method through simulation experiments and compare it with a replica placement strategy based on ant colony algorithms. Experimental results demonstrate that the algorithm can select appropriate replica placement nodes, exhibits good convergence properties, and effectively reduces replica access overhead in storage systems.

Keywords: replica placement; glowworm swarm optimization; cloud computing; distributed storage

---

0 Introduction

With the development of Internet technology, the volume of data requiring processing and storage has grown explosively. In the era of massive data, traditional distributed storage solutions suffer from excessive costs, poor scalability, and difficulty in providing persistent and effective storage services. To address these challenges, distributed storage technology in cloud computing environments has emerged. Cloud-based distributed storage leverages cluster applications, grid technologies, or distributed file system capabilities to aggregate numerous heterogeneous storage devices across networks through software, enabling them to work collaboratively and provide unified data storage and business access functions. Cloud computing environments are characterized by large data volumes, heterogeneous storage devices, and varying network conditions, making data loss and errors commonplace. Replica technology offers an effective solution to this problem by preserving multiple copies of data on different storage nodes, thereby improving storage system reliability and availability. Among the key issues in replica technology, placement strategy significantly impacts user access performance, storage system load balancing, network bandwidth utilization, and consistency maintenance among replicas.

Researchers in academia and industry have conducted extensive studies on replica placement in cloud-based distributed storage systems, yielding numerous achievements. Literature [4] demonstrates that the replica placement problem is NP-hard and proposes a reduction algorithm based on graph theory to solve replica placement and reduce access response time overhead. Literature [5] presents a distributed greedy replica placement mechanism that places replicas according to data object request frequency and system capacity, reducing average data access time and theoretically proving that the method provides a 2-approximation solution. For fault and node failure scenarios, literature [6] proposes a replica reconstruction strategy to address performance degradation when nodes storing data replicas fail. Literature [7] investigates replica placement for timeout-based consistency maintenance mechanisms and presents an algorithm for obtaining optimal placement strategies. Literature [8] introduces an adaptive replica management mechanism that creates new replicas for hot data and places them on lightly loaded nodes to reduce access response time. Content delivery networks based on cloud serve as effective architectures for content distribution, improving QoS, scalability, and resource efficiency. For replica server placement problems, literature [9] proposes a greedy heuristic approach to minimize operational overhead while meeting QoS requirements. Literature [10] addresses traditional service-storage cluster VOD systems by proposing an adaptive media replica placement algorithm that demonstrates good load balancing and overall performance under varying data volumes. To solve replica placement issues in data grid environments, literature [11] organizes placement nodes into tree structures and investigates load balancing and replica quantity. Literature [12] proposes a service replica placement strategy SNPBRA based on social network characteristics for cloud environments, which replicates partial cloud services to reduce inter-regional service interactions and improve business process execution efficiency. Literature [13] studies data replica aspects in educational grid data resource sharing problems, designing a reasonable replica directory management model and proposing dynamic replica creation strategies that effectively improve replica location efficiency and adapt to dynamic changes in user requests. Additionally, some swarm intelligence algorithms, such as particle swarm optimization and ant colony algorithms, have been applied to replica problems in storage systems [14,15].

Glowworm swarm optimization is a swarm intelligence-based algorithm whose fundamental concept involves simulating the luminescent behavior of natural fireflies to search for brighter companions in the vicinity, gradually moving toward optimal positions within the region and aggregating at the brightest location to achieve optimal solution searching. During the optimization process, each glowworm operates independently, enabling concurrent execution. Moreover, glowworm swarm optimization avoids the premature convergence potential of other intelligent algorithms, with its convergence speed accelerating as iteration count increases, making it suitable for clustering, classification, and combinatorial optimization problems. The algorithm features simple concepts, easy implementation and operation, high convergence speed, and optimization precision, finding widespread application across many domains with promising prospects.

---

1 Glowworm Swarm Optimization Algorithm

Glowworm swarm optimization (GSO), proposed by Indian scholars Kaipa et al. [16], represents a novel swarm intelligence optimization algorithm capable of achieving optimization in solution spaces. The algorithm draws inspiration from the natural phenomenon of firefly aggregation at night, where each firefly communicates information about food seeking and mating through its luminescent signals. The stronger the luciferin carried by a firefly, the greater its attraction, ultimately resulting in numerous fireflies gathering around the brightest individuals.

When applying GSO to problem solving, glowworms are initially scattered randomly throughout the solution space, each carrying equal luciferin values. Through multiple iterations, each glowworm' s luciferin value is updated according to the fitness function value of its position. Glowworms compare luciferin values with others within their decision domain radius and move toward brighter individuals based on a probability mechanism. Specifically, the GSO algorithm proceeds through the following steps:

a) Initialize glowworm i by randomly placing it within the search range of the objective function and set initial parameters.

b) Update luciferin values. Use equation (1) to convert the fitness function value corresponding to glowworm i' s position at iteration t into a luciferin value $l_i(t)$, where ρ represents the luciferin volatilization factor and γ represents the luciferin enhancement factor:

$$l_i(t) = (1-\rho)l_i(t-1) + \gamma J(x_i(t))$$

c) Determine neighborhood sets. At iteration t, for glowworm i, select individuals j within its dynamic decision domain radius whose luciferin values exceed i' s to form its neighborhood set $N_i(t)$, where $d_{ij}(t)$ is the distance between individuals i and j:

$$N_i(t) = \{j : d_{ij}(t) < r_d^i(t); l_i(t) < l_j(t)\}$$

d) Calculate movement probabilities. Compute the probability $p_{ij}(t)$ of glowworm i moving toward individual j in its neighborhood at iteration t using equation (3):

$$p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} (l_k(t) - l_i(t))}$$

e) Update positions. Complete the position update for glowworm i after movement using equation (4), where s is the step size:

$$x_i(t+1) = x_i(t) + s \cdot \frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}$$

f) Update dynamic decision domain radius. Update the dynamic decision domain radius using equation (5), where r_s is the maximum perception radius of glowworm individuals, β is the dynamic decision domain update rate, n_t is the threshold number of glowworms in an individual' s neighborhood set, and $|N_i(t)|$ is the number of glowworms in individual i' s neighborhood:

$$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\}$$

---

2 Replica Placement Algorithm Based on Glowworm Optimization

Through analysis of glowworm swarm optimization and the replica placement problem, this paper proposes a discrete glowworm optimization-based replica placement algorithm. By comprehensively considering factors influencing replica placement, we calculate glowworms' luciferin values through their fitness values and determine movement probabilities between individuals, ultimately moving toward the brightest glowworm to find the optimal solution and select appropriate replica placement positions.

In a cloud-based distributed storage system, assuming N nodes connected in a distributed manner, where each node can store at most one replica of the data and the number of data replicas in the storage system is R, the replica placement problem can be described as selecting an appropriate set of replica placement nodes that can handle data replica access services while minimizing overall replica access overhead.

Based on the above problem definition and the core principles of glowworm swarm optimization, we propose a glowworm optimization-based replica placement method. Before modeling and elaborating the algorithm, we make the following assumptions:

a) The node topology in the cloud-based distributed storage system is fixed. Storage system nodes are heterogeneous with different performance characteristics and certain failure probabilities.

b) To simplify the algorithm, we assume a node can access only one data replica at a time. Future work will extend this algorithm to support parallel access to multiple data replicas.

c) Network bandwidth consumption for accessing data replicas is measured by the two-dimensional distance between nodes.

2.1 Replica Placement Model

Based on the replica placement problem definition and assumptions, we model the problem and present its mathematical description along with parameter meanings. First, we define the objective function—the replica access cost function—as shown in equation (6), whose goal is to minimize overall replica access cost. Therefore, when this expression reaches its minimum value, the selected nodes constitute the optimal replica placement nodes:

$$\min \sum_{i \in N} \sum_{j \in N} D_i \cdot C_{ji} \cdot r_{ji} \cdot p_{ji}$$

Second, we present the constraints for this objective function:

$$i, j \in [1, N]$$

$$D_i = \{j \mid d_{ij} \leq \text{Distance}, i \neq j\}$$

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

$$p_{ji} \in [0, 1]$$

$$\sum_{j \in N} p_{ji} = R$$

$$\sum_{j \in N} p_{ji} = 1$$

where equation (6) represents node performance. Equation (7) defines the storage node indices, with total nodes N. Equation (8) defines, for a given storage node i, the set of nodes whose distance to node i is less than Distance. Equation (9) defines the distance between two nodes i and j. In equation (10), p_{ji} indicates whether storage node j is selected as a replica placement node (set to 1 if selected, 0 otherwise); if 1, node j provides replica access for node i, otherwise it does not. Equation (11) indicates that the number of selected replica placement nodes is R. Equation (12) ensures that only one node provides replica access for each storage node.

2.2 Discrete Glowworm Optimization-Based Replica Placement Algorithm

Based on glowworm swarm optimization principles, this section elaborates the proposed algorithm from four perspectives: solution construction and encoding, distance metric between individuals, position update method, and fitness function construction.

2.2.1 Solution Construction and Encoding The basic glowworm algorithm applies to continuous domains; we discretize it to solve the replica placement problem. This paper employs integer encoding to represent a feasible solution, such as $[2, 6, 9, 18, 26]$, where each integer represents a replica placement node ID. Each feasible solution is an R-dimensional sequence with unique node IDs.

2.2.2 Distance Metric Between Individuals Since replica placement is a discrete combinatorial optimization problem and glowworm positions are constructed through encoding with integer-valued variables, we cannot use Euclidean distance from the basic glowworm algorithm to calculate inter-individual distances. Instead, we measure distance through encoding discrepancy [17]. For two glowworms i and j, we define their distance at iteration t as:

$$d_{ij}(t) = C \cdot \delta_{ij}(t)$$

where C is a constant and $\delta_{ij}(t)$ represents the discrepancy between the encoding sequences of glowworms i and j at iteration t. The discrepancy calculation method is shown in equation (14):

$$\delta_{ij}(t) = \frac{1}{R}\sum_{k=1}^{R}\frac{|d_i^k(t) - d_j^k(t)|}{M}$$

where $|d_i^k(t) - d_j^k(t)|$ represents the absolute difference of codes at the same position in two encoding sequences, and M is the upper bound of values. When N is even, $M = \frac{2N}{R}$; when N is odd, $M = \frac{2(N-1)}{R}$. In this paper, C is set to 20.

2.2.3 Position Update Method Due to the discrete nature of replica placement, equation (4) is no longer applicable for individual position updates. Assuming the distributed storage system contains N nodes and needs to select R replica placement nodes, the solution structure for the replica placement problem is $x_i = (x_i^1, x_i^2, \dots, x_i^R)$, where n is the number of glowworm individuals. Each glowworm corresponds to an R-dimensional row vector, where each dimension selects the corresponding position update formula with different probabilities to update the solution vector:

$$x_i^k(t+1) = \begin{cases} x_j^k(t), & \text{if } r_k < p_1 \\ x_i^k(t), & \text{if } p_1 \le r_k < p_2 \\ \mathrm{Int}(N \cdot \mathrm{rand}), & \text{if } r_k \ge p_2 \end{cases}$$

where r is an R-dimensional random sequence generated between 0 and 1; $x_j^k(t)$ is the position of individual j selected by individual i at iteration t; and p_1 and p_2 are update selection probability parameters with $p_1, p_2 \in [0, 1]$ and $p_1 < p_2$.

According to equation (15), glowworm i accepts the k-th dimension data $x_j^k(t)$ from $x_j(t)$ encoding with probability p_1 and updates its k-th variable.

During glowworm swarm optimization iterations, variables in some solution vectors may not be unique. To maintain population scale and diversity, after each iteration we check the number of unique variables m in the solution vector. If $m < R$, we randomly generate $R - m$ unique variables within $[1, N]$ to replace infeasible solutions with feasible ones. This ensures that solutions in the search space remain feasible during each iteration, reducing the probability of converging to suboptimal solutions due to randomness.

To address local optima problems, we randomly generate an integer i $(0 < i < R)$ in the algorithm' s later stages. For individuals with better and worse objective function values in the population, we randomly generate two segments of length i for exchange. If the new individual' s objective function value is better than the original, it replaces the original; otherwise, the original is retained.

The fitness function $J(x_i(t))$ is the reciprocal of the value obtained from equation (16) based on glowworm i' s encoding at iteration t. F is set as a large positive penalty coefficient. The fitness function considers node failure probability: if a node' s probability of failure within a certain period exceeds the predefined system threshold Tolerance, the fitness function value decreases, substantially reducing the likelihood of that node being selected as a replica placement node. The node failure probability threshold can be set according to storage system application requirements:

$$J(x_i(t)) = \frac{1}{\sum_{i\in N}\sum_{j\in N} D_i \cdot C_{ji} \cdot r_{ji} \cdot \mathrm{Disp}_j + F \cdot \mathrm{Failure}_j - \mathrm{Tolerance}}$$

2.3 Discrete Glowworm Optimization-Based Replica Placement Algorithm Steps

The specific steps of the proposed discrete glowworm optimization algorithm for solving the replica placement problem are as follows:

a) Set basic algorithm parameters.

b) Initialize the glowworm population. Set the total number of nodes N in the distributed storage system, the number of replica placement nodes R (i.e., each glowworm' s dimension), with each dimension taking random integer values in $[1, N]$, and set the iteration counter variable *iter* to 0.

c) Calculate the fitness of each glowworm individual's corresponding solution using the fitness function formula, then obtain the corresponding luciferin values.

d) Calculate distances between glowworm individuals. Each glowworm selects individuals with higher luciferin values within its dynamic decision radius to form its neighborhood.

e) Calculate the probability of glowworm i moving toward individual j through the movement probability formula, with i moving toward the individual j with maximum probability.

f) Randomly generate an R-dimensional vector r in $[0, 1]$. Update each dimension variable in the solution vector according to equation (15) based on each dimension' s value in vector r, and handle infeasible solutions accordingly.

g) Update each glowworm' s decision domain radius according to equation (5).

h) Check whether $iter > Iter_{max}$ is satisfied. If so, the algorithm terminates and outputs the global optimal value and its corresponding position; otherwise, return to step c).

2.4 Time Complexity Analysis

Let n be the number of glowworms, R the number of replica placement nodes, and $Iter_{max}$ the maximum iteration count. The time complexity for initialization is $O(n)$. Calculating luciferin values for n individuals costs $O(n)$. Calculating inter-individual distances costs $O(n^2)$. During one iteration, glowworm position update costs $O(nR)$. Therefore, the overall time complexity within the iteration count is $O(Iter_{max} \cdot nR)$.

---

3 Experimental Evaluation

To verify the proposed discrete glowworm optimization-based replica placement algorithm, we first use MATLAB to validate the algorithm' s effectiveness and convergence. We extend CloudSim by modifying the Host and Datacenter classes and adding a new replica placement strategy component to complete replica placement work, then compare and analyze the replica access overhead of different algorithms. The experimental environment consists of: Windows 7 operating system, MATLAB R2014a, CloudSim 3.0.3, i5-3470 @ 3.2 GHz processor, and 8 GB memory. Algorithm parameters used in this experiment are shown in Table 1 .

Table 1 Experimental Parameter Values - Luciferin volatilization and enhancement factors: $\rho = 0.4$, $\gamma = 0.6$ - Dynamic decision domain update rate: 0.5 - Initial luciferin value: 5 - Initial dynamic decision domain radius: 3 - Individual maximum perception radius: 20 - Neighborhood set threshold: 5 - Position update formula selection parameters: $p_1 = 0.2$, $p_2 = 0.8$

This experiment employs normalized geographic coordinates of Chinese provincial capitals and municipalities to represent corresponding positions of different nodes in the storage system. Failure probabilities within $[0.01, 0.05]$ are randomly assigned to each node, and the system node failure probability threshold

Tolerance is empirically set to 0.03. The replica quantity follows the three-replica principle.

3.1 Replica Placement Node Selection

Figure 1 [Figure 1: see original paper] illustrates the distribution of nodes in the distributed storage system and the replica placement nodes selected by the proposed method. Our approach constructs the objective function based on node performance and failure probability, selecting nodes that minimize the objective function as replica placement positions. During replica placement selection, the goal is minimizing overall replica access overhead. Once replica placement nodes are determined, the total overhead is minimized, enabling appropriate node selection for replica placement, reducing replica access overhead, improving user access performance, and balancing system load.

3.2 Algorithm Convergence

Figure 2 [Figure 2: see original paper] depicts the convergence curve of the glowworm optimization-based replica placement algorithm. The figure shows that the algorithm quickly converges to the optimal value, can select appropriate nodes for replica placement, and minimizes user access overhead to replicas. The proposed glowworm optimization-based replica placement algorithm can efficiently select suitable replica placement nodes and demonstrates good convergence properties.

3.3 Parameter Selection

Figure 3 [Figure 3: see original paper] shows the algorithm' s convergence curves under different population sizes and iteration counts. The population size n affects the algorithm' s objective function value, achieving good convergence when $n > 80$. Figures 4 [Figure 4: see original paper] and 5 [Figure 5: see original paper] illustrate the impact of glowworm population size and maximum iteration count on the proportion of optimal objective function values obtained. We conduct 50 experiments under certain population sizes and iteration counts, statistically analyzing the proportion achieving optimal objective function values. Figure 4 indicates that increasing glowworm population size initially improves algorithm performance, but the proportion of optimal values stabilizes as population size continues to grow. Figure 5 shows that when population size takes default values, iteration count does not significantly impact algorithm performance, and increasing iterations does not substantially improve the proportion of optimal objective function values. Population size and iteration count are two key factors affecting algorithm time complexity. Appropriate population size and iteration count should be selected, potentially reducing iteration count while increasing population size to lower time complexity while maintaining high algorithm performance.

3.4 Access Time

Figure 6 [Figure 6: see original paper] presents simulation experiments on CloudSim comparing the glowworm optimization-based replica placement strategy with the ant colony algorithm-based strategy [18], analyzing average access time under different user access patterns as a metric for access overhead. The figure demonstrates that under sequential, uniform, Gaussian, and Zipf distribution access patterns, the average access time of the proposed glowworm optimization-based replica placement algorithm is superior to the ant colony algorithm-based strategy. Since our method minimizes overall replica access overhead during replica placement selection, the glowworm optimization-based algorithm can select more appropriate nodes for replica placement, effectively reducing replica access time and demonstrating good performance.

---

4 Conclusion

This paper proposes a discrete glowworm optimization-based replica placement algorithm for replica placement problems in cloud-based distributed storage, selecting optimal replica placement positions. We comprehensively consider the impact of replica placement nodes on user access performance, establish a mathematical model, design a reasonable encoding scheme, employ encoding discrepancy to measure distances between glowworm individuals, and introduce glowworm position update and infeasible solution handling methods. Simulation experiments verify the proposed method' s effectiveness, with results showing that it reduces overall user access overhead to replicas and exhibits good convergence. Future work will implement this algorithm in actual distributed storage environments.

References

[1] Mell P M, Grance T. The NIST definition of cloud computing. National Institute of Standards and Technology [R]. Gaithersburg: National Institute of Standards and Technology, 2011.

[2] Sun Dawei, Chang Guiran, Gao Shang, et al. Modeling a dynamic data replication strategy to increase system availability in cloud computing environments [J]. Journal of Computer Science and Technology, 2012, 27(2): 256–272.

[3] Ye Zhen, Hu Weijian. An adaptive replica selection algorithm for quorum based distributed storage system [J]. International Journal of Grid and Distributed Computing, 2016, 9(5): 55-68.

[4] Bai Xiaohu, Jin Hai, Liao Xiaofei, et al. RTRM: a response time-based replica management strategy for cloud storage system [C]// Proc of International Conference on Grid and Pervasive Computing. Heidelberg: Springer, 2013: 124-133.

[5] Zaman S, Grosu D. A distributed algorithm for the replica placement problem [J]. IEEE Trans on Parallel and Distributed Systems, 2011, 22(9): 1455-1468.

[6] Higai A, Takefusa A, Nakada H, et al. A study of effective replica reconstruction schemes for the hadoop distributed file system [J]. IEICE Trans on Information and Systems, 2015, E98.D(4): 872-882.

[7] Tang Xueyan, Chanson S T. Analysis of replica placement under expiration-based consistency management [J]. IEEE Trans on Parallel and Distributed Systems, 2006, 17(11): 1253-1263.

[8] Chen Lingfeng, Hoang D B. Adaptive data replicas management based on active data-centric framework in cloud environment [C]// Proc of IEEE International Conference on High Performance Computing and Communications and IEEE International Conference on Embedded and Ubiquitous Computing. New York: IEEE Press, 2013: 101-108.

[9] Sahoo J, Glitho R. Greedy heuristic for replica server placement in cloud based content delivery networks [C]// Proc of IEEE Symposium on Computers and Communication. New York: IEEE Press, 2016: 302-309.

[10] Zhao Jun, Jin Hai. Adaptive cluster streaming media file replica placement strategy [J]. Computer Applications Research, 2008, 25(2): 594-596.

[11] Wu Janjan, Lin Yingfang, Liu Pangfeng. Optimal replica placement in hierarchical data grids with locality assurance [J]. Journal of Parallel and Distributed Computing, 2008, 68(12): 1517-1538.

[12] Luo Haoyu, Chen Wanghu. Cloud service replica placement strategy based on social network characteristics [J]. Computer Applications, 2013, 33(8): 2143-2146.

[13] Gao Tian, Liu Fang' ai. A dynamic data replication technology in educational resource grid [J]. Computer Applications Research, 2008, 25(3): 869-871.

[14] Wang Lijuan, Luo Junzhou, Shen Jun, et al. Cost and time aware ant colony algorithm for data replica in alpha magnetic spectrometer experiment [C]// Proc of IEEE International Congress on Big Data. New York: IEEE Press, 2013: 247-254.

[15] Lim S P, Haron H. Performance comparison of genetic algorithm, differential evolution and particle swarm optimization towards benchmark functions [C]// Proc of IEEE Conference on Open Systems (ICOS). New York: IEEE Press, 2013: 41-46.

[16] Kaipa K N, Ghose D. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics [C]// Proc of IEEE Swarm Intelligence Symposium. New York: IEEE Press, 2005: 84-91.

[17] Singh A, Thapar S, Bhatia A, et al. Disk scheduling using a customized discrete firefly algorithm [J]. Cogent Engineering, 2015, 2(1): 1011929.

[18] Shen Wei, Liu Fang'ai. Data replica placement strategy based on ant colony algorithm [J]. Computer Applications Research, 2007, 24(6): 82-84.

Note: Figure translations are in progress. See original paper for figures.