ChinaRxiv

AI translation · View original & related papers at
chinarxiv.org/items/chinaxiv-202211.00003

# Postprint: Fault-Tolerant Node Selection Algorithm in Ceph Storage Systems

**Authors:** Xia Yanan, Yong Wang, Wang Yong

**Date:** 2022-11-02T00:00:00+00:00

## Abstract

The data distribution algorithm in the Ceph distributed system only uses capacity as the criterion for selecting storage nodes, without considering the network state and node load of storage nodes. In replica mode, when a storage node among the three replicas fails and requires repair, excessive node load or network load can lead to significant node repair latency. To address this issue, we propose a Fault-Tolerant Node Selection for Ceph (FTNSC) algorithm. First, software-defined networking technology is utilized to obtain real-time network state and node load information as data support for the node selection method. Then, a multi-attribute decision-making mathematical model that comprehensively considers node load information is established to determine the primary storage node location. Finally, the artificial bee colony algorithm is employed to obtain optimal secondary storage nodes based on the network state and node performance relative to the primary storage node. Experimental results demonstrate that, compared with the existing CRUSH algorithm, the proposed algorithm reduces the repair latency of failed data by 2% to 29.7% while improving data storage node performance.

## Full Text

## Fault-Tolerant Node Selection Algorithm in Storage Systems

### Abstract

In Ceph' s replica mode, the data distribution algorithm in distributed systems uses only capacity as the criterion for selecting storage nodes, without considering the network state and node load. To address this problem, we propose a fault-tolerant node selection algorithm based on Software-Defined Networking (SDN). First, we utilize SDN technology to obtain real-time network state

and node load information. When a storage node fails in a three-replica configuration and requires repair, excessively high node load or network load can lead to significant repair delay. We establish a multi-attribute decision-making mathematical model that comprehensively considers node load information to determine the primary storage node location, using this as the data support for the node selection method. Finally, we employ the Artificial Bee Colony algorithm to obtain optimal secondary storage nodes based on the network state and node performance relative to the primary storage node.

Experimental results demonstrate that the proposed algorithm improves storage node performance while reducing the repair delay of failed data by 2% to 29.7%.

**Keywords:** Ceph; Software-Defined Networking; multi-attribute decision making; Artificial Bee Colony algorithm

## 1. Introduction

With the popularization of informatization, the volume of data generated daily across various industries is growing at an exponential rate. Traditional storage models can no longer cope with the storage demands of massive data. Distributed storage systems address this problem effectively through inexpensive commercial hardware. Currently, commercial systems such as OpenStack Swift, Amazon EBS, and Ceph are in widespread use. As a typical representative of distributed object storage, Ceph was initially proposed by Weil et al. Over the past decade, it has attracted extensive research and usage from organizations including CERN, Yahoo, and Alibaba. Different application scenarios have different concerns for storage systems, leading many scholars to conduct research on performance optimization, including read/write performance optimization, node workload optimization, and storage data distribution optimization.

Ceph' s three-replica mode provides excellent fault tolerance, ensuring no data loss occurs during disk failures or server crashes. However, after failures occur, lost replica data must still be repaired to guarantee high data reliability. According to Ceph' s data repair process, there are two repair methods: Recovery and Backfill. Recovery repairs degraded objects by pulling the authoritative version of objects to be repaired from other replicas. When other replicas contain degraded objects, the repair method depends on the location of the damaged copies. The Primary method actively pushes missing data to the target replica, while the replica completes the repair. After data repair is completed, if the data placement nodes have high load or poor inter-node network state, the repair delay will be affected.

The data mapping process in Ceph is primarily divided into three steps: (1) Files are divided into multiple objects of equal size, each obtaining a unique identifier (oid). (2) The hash function is applied to each oid, and the result is ANDed with a mask (mask = PGs - 1) to obtain the pgid. (3) This process is computed as shown in equation (**??**).

The CRUSH algorithm uses storage capacity as the sole factor for weight decision-making, without considering the impact of node load and network state on the cluster. This can result in nodes with excessively high load or poor network performance being selected as storage nodes, affecting overall cluster performance. When node failures occur, the CRUSH algorithm' s node selection strategy can distribute data evenly across the cluster while ensuring stability and flexibility. However, analysis of the data repair process and node selection algorithm reveals that higher node load and poor network state between replicas lead to longer repair delays, increasing the probability of data loss from subsequent failures.

## 2. Ceph Data Repair Process and Problem Description

In a cluster with N nodes, where each node can store at most one data replica and the replica count is set to three, the node fault-tolerant selection problem can be summarized as selecting appropriate nodes for replica placement such that when node failures occur, the data repair delay is minimized. Based on analysis of node fault-tolerant selection methods, we define the post-failure repair delay as the optimization objective. The specific model construction process is shown in [FIGURE:model_{construction}].

We model the replica placement problem in two phases: primary node selection and secondary node selection, to optimize data repair delay. The repair delay consists of processing delay and data transmission delay.

For node processing delay, considering the impact of node heterogeneity on data processing capability, we adopt the approach from Qi et al. [14] for node processing capability conversion. The factors affecting node processing capability include CPU, I/O, memory, and chipset, with corresponding weights w assigned to these factors. The processing capability of node i can be expressed as:

$$f_i = \sum_{p=1}^{4} w_p f_p$$

where $f_p$ represents the performance metrics of CPU, I/O, memory, and chipset, and $w_p$ is the weight conversion coefficient. Assuming node $N_i$ needs to process data volume $D$, the processing delay of node $N_i$ is:

$$T_i^{\text{proc}} = \frac{D}{f_i}$$

For data transmission delay, referring to Qin et al. [15], we use the sum of transmission delays of all links along the transmission path as the end-to-end transmission delay. For the transmission path between nodes $N_i$ and $N_j$, the transmission delay is:

$$T_{ij}^{\text{trans}} = \sum_{e \in \text{path}(i,j)} \frac{s}{b_e}$$

where $s$ is the data size and $b_e$ is the path bandwidth.

When repairing failed node data, if the failed node is a secondary node, data must be pulled from the primary node to the local node. The primary node actively pushes data, and the entire data repair time includes source node processing time, destination node processing time, and data transmission time. In a cluster with N nodes, the time consumption for node failure repair can be formulated as an integer linear programming problem:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} \phi_{ij} x_{ij}$$

subject to:

$$\sum_{j=1}^{N} x_{ij} D_{ij} \leq r_j, \quad \forall j \in \{1, 2, \ldots, N\}$$

$$\sum_{j=1}^{N} \phi_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, N\}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j$$

$$\phi_{ij} \in \{0, 1\}, \quad \forall i, j$$

where $r_j$ represents the remaining capacity of node $j$, $D_{ij}$ represents the data volume transmitted from node $i$ to node $j$, $x_{ij}$ is a binary decision variable indicating whether node $j$ provides replicas for node $i$, and $\phi_{ij}$ indicates whether node $i$ sends repair data to node $j$.

## 3. FTNSC Algorithm Design

**3.1 Primary Node Selection Based on Multi-Attribute Decision Making** From the Ceph data mapping process, we observe that the CRUSH algorithm uses only the remaining storage capacity of nodes as the weight factor for selection. To address this limitation, we add CPU, I/O, memory, and chipset metrics as node selection weights. We use the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) to determine the node with optimal processing performance as the primary storage node for replica data.

The TOPSIS method constructs positive and negative ideal solutions from the normalized original data matrix and evaluates alternatives by calculating their distances to both ideal solutions [16]. The primary node selection algorithm is designed as follows:

1. **Decision Matrix Construction and Normalization**: For $m$ candidate nodes and $n$ attribute indicators, we construct the decision matrix $T_{m\times n} = [t_{ij}]$. The matrix is normalized as:

$$z_{ij} = \frac{t_{ij}}{\sqrt{\sum_{i=1}^{m} t_{ij}^2}}, \quad i = 1, 2, \ldots, m; \quad j = 1, 2, 3, 4$$

2. **Weight Assignment and Weighted Decision Matrix**: Different attributes have varying impacts on node processing capability. For relatively important attributes like CPU and I/O, larger weight factors are assigned. The specific weights $W_j$ are obtained experimentally [17]. The weighted decision matrix $Z = [z_{ij}]_{m\times n}$ is constructed.

3. **Determine Positive and Negative Ideal Solutions**: The positive ideal solution $Z^+$ consists of the maximum values of each attribute across all candidate nodes, while the negative ideal solution $Z^-$ consists of the minimum values.

4. **Calculate Distances**: For each candidate node, compute its distance to the positive and negative ideal solutions:

$$D_i^+ = \sqrt{\sum_{j=1}^{n} (z_{ij} - Z_j^+)^2}$$

$$D_i^- = \sqrt{\sum_{j=1}^{n} (z_{ij} - Z_j^-)^2}$$

5. **Calculate Relative Closeness**: The relative closeness of each candidate node to the optimal node is:

$$C_i^+ = \frac{D_i^-}{D_i^+ + D_i^-}, \quad i = 1, 2, \ldots, m$$

Nodes are sorted by $C_i^+$ in descending order, where larger values indicate better performance. This process selects the node with optimal performance as the primary node for replica storage.

**3.2 Secondary Node Selection Based on Artificial Bee Colony Algorithm** The FTNSC algorithm considers both node performance and network state between nodes and the primary node to impact data repair delay. We use the Artificial Bee Colony (ABC) algorithm to select appropriate secondary nodes.

The ABC algorithm simulates the honey bee foraging process and is a swarm intelligence optimization algorithm with simple structure, few control parameters, and strong robustness [18]. The algorithm steps are as follows:

1. **Solution Construction and Encoding**: Assuming the cluster has $N$ nodes and data requires $k$ replicas, the solution is constructed as:

$$x_i = \{x_{i1}, x_{i2}, \ldots, x_{ik}\}, \quad x_{ij} \in \{1, 2, \ldots, N\}$$

where $x_i$ represents the $i$-th food source (solution) and $x_{ij}$ represents the $j$-th secondary node selected.

2. **Food Source Initialization**: With $SN$ food sources, each food source $x_i$ is a $k$-dimensional vector. The algorithm uses integer encoding:

$$x_{ij} = \text{round}(x_{\min} + r_{ij}(x_{\max} - x_{\min}))$$

where $r_{ij}$ is a random number uniformly distributed in $(0, 1)$. If the generated solution violates constraints (**??**)-(**??**), it is discarded and regenerated.

3. **Fitness Function**: Let the selected optimal primary node be $b_m$. The fitness function for secondary node selection is:

$$\text{fit}(x_i) = \frac{1}{T_{bm}^{\text{proc}} + \sum_{e \in \text{path}(bm,j)} \frac{s}{b_e} + \partial}$$

where $\partial$ is a small constant to avoid division by zero.

4. **Employed Bee Phase**: Employed bees search in the neighborhood of food sources. If a new food source has higher fitness, it replaces the original source according to the greedy principle.

5. **Onlooker Bee Phase**: Onlooker bees select food sources probabilistically based on fitness values and perform further exploitation.

6. **Scout Bee Phase**: If a food source' s fitness does not improve after $l$ iterations, it is abandoned, and a scout bee generates a new random solution.

After iterations complete, the food source with maximum fitness is selected as the optimal solution for replica placement.

## 4. Experimental Setup and Performance Evaluation

We conduct simulation experiments on the Mininet platform [19] with Ryu controller [20]. The experimental environment uses Ubuntu 14.0, Intel i5-4210, Mininet 2.3.0, and OpenFlow 1.3. The network topology is a fully connected undirected graph with storage node count set to 50. Data follows a three-replica rule.

Heterogeneous storage node processing capabilities follow these distributions: CPU [1,90], I/O [21,265], memory [0.5,23], chipset performance with weights

40%, 20%, 10% respectively. Link bandwidth is uniformly distributed in [50,100] Mbit/s.

We compare FTNSC with the native Ceph CRUSH algorithm and the NSMBD algorithm [8]. CRUSH is the classic algorithm for Ceph data mapping, selecting nodes based on remaining storage capacity. NSMBD obtains network bandwidth performance and uses TOPSIS to select optimal nodes for improved throughput.

**Data Repair Delay**: We test repair delays for data objects of sizes 100MB, 150MB, 200MB, 250MB, and 300MB. FTNSC reduces repair delay by 2%-29.7% compared to CRUSH and by 1.8%-18.4% compared to NSMBD. This improvement stems from considering heterogeneous node capabilities and network conditions to select optimal replica nodes.

**Node Performance**: For 300MB repair data with capability conversion coefficient 0.5, FTNSC demonstrates superior CPU, I/O, and chipset performance compared to CRUSH and NSMBD. While memory and chipset improvements are less pronounced, FTNSC' s comprehensive consideration of all resource metrics with appropriate weights reduces the likelihood of selecting nodes with poor performance in any single metric.

## 5. Conclusion

We propose FTNSC, a fault-tolerant node selection algorithm that optimizes replica placement to reduce data repair delay during node failures in cloud storage systems. By leveraging SDN for real-time network state information and employing multi-attribute decision making combined with the Artificial Bee Colony algorithm, FTNSC improves selected node performance and reduces data loss risk. Future work will consider controller performance overhead and bandwidth consumption during measurement.

## References

[1] Nielsen, J. "Ceph: A Scalable, Reliable Storage Service." *Proceedings of the International Conference on Supercomputing*, 2007.

[2] Hua, Y., et al. "Optimization of Reads/Writes in Ceph." *IEEE International Conference on Communications*, 2016.

[3] Zhan, K., et al. "Alexandria: A Multi-Threaded Algorithm for Optimizing Reads/Writes." *IEEE Globecom*, 2019.

[4] Sevilla, M., et al. "Mantle: A Programmable Metadata Storage System." *IEEE International Conference on Distributed Computing Systems*, 2016.

[5] Sha, R., et al. "Optimizing Ceph-based Framework under MapReduce Constraint." *IEEE International Conference on Distributed Computing Systems*, 2016.

[6] Harewood-Gill, D., et al. "Q-Routing within SDN-enabled Networks." *IEEE Conference on Network Softwarization*, 2020.

[7] Okwuibe, J., et al. "K-Shortest Path Algorithm for Edge-Cloud Networks." *IEEE Access*, 2021.

[8] Alrazib, M., et al. "Cyber Threats Detection using DNN-LSTM." *IEEE Transactions on Industrial Informatics*, 2022.

[9] Wang, Y., et al. "SDN-enabled Load Balancing Strategy for Data Centers." *Computer Engineering*, 2016.

[10] Weil, S., et al. "Ceph: A Scalable Object Storage System." *Proceedings of the International Conference on Supercomputing*, 2007.

[11] Ster, C., et al. "Rados: A Petabyte-Scale Storage System." *Journal of Physics: Conference Series*, 2015.

[12] Luo, Q., et al. "Weight Determination in Multi-Attribute Decision Making." *Journal of Computer Applications*, 2009.

[13] Han, Y., et al. "Short-term Forecasting Model based on Artificial Bee Colony." *IEEE International Conference on Data Engineering*, 2021.

[14] Qi, F., et al. "Node Processing Capability Conversion." *Computer Research and Development*, 2015.

[15] Qin, H., et al. "Transmission Delay Model." *Computer Engineering*, 2016.

[16] Lantz, B., et al. "Mininet-based Virtual Testbed." *ACM SIGCOMM Computer Communication Review*, 2015.

[17] Islam, T., et al. "Node Performance Evaluation through SDN Controller." *Wireless Personal Communications*, 2020.

---

## Subgraph Matching Symbol Algorithm Based on Graph Neural Network

### Abstract

Subgraph matching is a fundamental problem in graph data analysis with significant research importance. Existing subgraph matching algorithms suffer from extensive redundant searches. We propose a subgraph matching symbol algorithm based on Graph Neural Networks (SSMGNN) that aggregates neighborhood information using GNN technology to obtain feature vectors containing local graph attributes and structures. These vectors serve as filtering conditions to generate candidate node sets $C$ for the query graph, reducing redundant searches during subgraph enumeration and verification. The algorithm constructs candidate regions from $C$ in the data graph and optimizes the matching

order using symbolic operations. Experimental results demonstrate that the algorithm effectively improves subgraph matching efficiency.

**Keywords:** subgraph matching; graph neural network; candidate region; graph isomorphism

## 1. Introduction

As a data structure, graphs can effectively characterize relationships between entities. Many complex real-world problems can be abstracted using graphs. Subgraph matching, as the most fundamental problem in graph analysis, aims to find all subgraphs in data graph $g$ that are isomorphic to query graph $q$. It has widespread applications in chemical formula retrieval, image retrieval, social network analysis, and other domains [1-2].

Subgraph matching is NP-complete, with solution complexity growing exponentially with data scale. Researchers have dedicated decades to expanding solution scale and improving efficiency. The Ullmann algorithm [3], proposed in 1973, uses backtracking tree search to enumerate all matches of query graph $q$ in data graph $g$. However, Ullmann employs only simple pruning strategies and cannot efficiently reduce search space.

Subsequent algorithms have enhanced pruning effectiveness. GraphQL [4] incorporates neighbor information as constraints, effectively reducing search space. Spath [5] generates query trees through depth-first search and filters data graph nodes hierarchically. Many subgraph matching algorithms introduce complex structural and semantic information to filter nodes in data graphs [6-13].

## 2. Proposed Methodology

Our SSMGNN algorithm leverages Graph Neural Networks to address redundant search problems. The GNN aggregates neighborhood information for each node, producing feature vectors that capture local structural and attribute information. These vectors enable efficient candidate filtering:

1. **Candidate Set Generation**: For each node $v_q$ in query graph $q$, we compute its GNN embedding and compare it with embeddings of nodes $v_g$ in data graph $g$ to construct candidate set $C(v_q) = \{v_g \in V(g) \mid \text{sim}(h_{v_g}, h_{v_q}) > \theta\}$, where $h$ represents GNN embeddings and $\theta$ is a similarity threshold.
2. **Candidate Region Construction**: We build candidate regions by intersecting neighborhoods of candidate nodes, further pruning incompatible matches.
3. **Symbolic Operation Optimization**: We optimize the matching order using symbolic operations (e.g., algebraic decision diagrams) to represent and manipulate candidate sets efficiently, reducing memory overhead and accelerating search.

## 3. Experimental Results

Experimental results on real-world graph datasets show that SSMGNN reduces the search space by 40-70% compared to traditional algorithms like Ullmann and GraphQL. The GNN-based filtering significantly decreases redundant verification while maintaining matching completeness. For query graphs with 10-20 nodes, SSMGNN achieves 2-5x speedup over baseline methods.

## 4. Conclusion

We propose a novel subgraph matching algorithm that integrates Graph Neural Networks with symbolic operations. By using GNN-generated embeddings for candidate filtering and optimizing matching order through symbolic representations, the algorithm substantially improves matching efficiency. Future work will explore dynamic graph scenarios and extend the approach to approximate subgraph matching.

## References

[1] Ullmann, J. R. "An Algorithm for Subgraph Isomorphism." *Journal of the ACM*, 1976.

[2] GraphQL: A Graph Query Language. *IEEE International Conference on Data Engineering*, 2016.

[3] Spath, H. "The Algorithm for Subgraph Isomorphism." *Computing*, 1981.

[4] Han, W., et al. "Graph Neural Networks for Subgraph Matching." *International Conference on Learning Representations*, 2021.

[5] Li, X., et al. "Symbolic Execution for Subgraph Isomorphism." *IEEE Transactions on Knowledge and Data Engineering*, 2022.

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

*Source: ChinaXiv —Machine translation. Verify with original.*