Fault-tolerant Local Recovery with Preprocessing in Multiple Shared Protection

Mengfei Zhu[1,2], Rui Kang[2], and Klaus-Tycho Foerster[3]
[1]China Mobile Group Design Institute Co. Ltd., Beijing, China
[2]Kyoto University, Kyoto, Japan
[3]TU Dortmund, Dortmund, Germany
Emails: zhumengfei@cmdi.chinamobile.com, kang.rui.u25@kyoto-u.jp, klaus-tycho.foerster@tu-dortmund.de

***Abstract*—In distributed microservice architecture, low latency and high reliability are crucial as they directly impact user experience by ensuring robust service continuity across the network. Function unavailability occurs due to various reasons, and when such failures are detected, the nodes hosting the backup resources can attempt autonomously local recovery, leveraging precomputed recovery policies, without the need for central processor intervention, thereby reducing recovery time. In scenarios involving shared protection with limited recovery capabilities, it is crucial to implement a well-defined policy for local recovery. Such a policy prevents repetitive or infeasible recovery attempts, ensuring effective recovery.**

This paper considers a local strategy to maximize the residual resilience, where individual nodes decide which function to recover. It hence works as a complement to global recovery to provide prompt recovery, as a first line of defense. This paper hence considers the preparation of priority settings with preprocessing based on the SUPPORTED model.We conduct a comprehensive analysis of the conditions for achieving feasible and optimal local recovery solutions with zero-round communication for one and two failures. Additionally, we formulate the problem as an integer linear programming problem and design a testbed to implement the local recovery algorithm incorporating preprocessing. Our evaluation reveals that the zero-round communication approach emerges as the optimal choice, drastically shortening recovery times to 10^{-6} seconds while maintaining 98% feasibility and perfect optimality for all practical solutions, on average, across two failure scenarios.

***Index Terms*—Local recovery, preprocessing, distributed algorithm, shared protection, fault tolerance, residual resilience, network virtualization.**

I. Introduction

A. Motivation

Modern network services like firewalls and load balancers are virtualized as Virtual Network Functions (VNFs) in virtual machines and containers. This virtualization decouples functions from specialized hardware, allowing multiple functions to run on the same server but introduces challenges in fault tolerance and failure recovery due to dependencies on the underlying, dynamically changing infrastructure [1]. Microservice architecture breaks an application into multiple independent functions distributed across various servers or containers. Latency is a critical factor as communication between functions introduces delays. Longer latency caused by failures are particularly intolerable.

To enhance network resilience against function unavailability, protection strategies are commonly employed. One such strategy is shared protection, which offers a cost-effective approach to provide fault-tolerance [2]. Computing resources are pooled among various backup functions hosted on the same physical host. This approach improves availability without the need for dedicated backup resources for each individual function, leading to reduced costs, minimized dependencies, and enhanced resource utilization efficiency. Through the allocation of shared protection resources across multiple critical nodes, the network achieves more efficient bandwidth utilization, enhancing its overall stability and reliability.

In such cases, functions are commonly protected by multiple nodes to guarantee the availability. Failure to do so may result in contention and failures. If multiple functions protected by the same node fail simultaneously, contention can arise if the demand for shared resources surpasses the available capacity for recovery, potentially leaving at least one of the failures unrecovered. To improve such issues, it involves #1) ensuring feasibility against the current failures, i.e., the failure should be recoverable, and #2) optimizing fault tolerance for future failures. The recovery policy should consider the tolerance for the future possible failures.

B. Background and related work

Recovery commonly follows two distinct approaches. One is from a global perspective, where all failure patterns are known, and predetermined recovery policies are applied ahead of failures [3], [4]. Alternatively, the central controller detects failures and determines the recovery policy [5], [6]. In contrast, the other approach is local (i.e., decentralized) failure recovery, with each node independently decides on the recovery procedure based on its local state and environment, without relying on global knowledge of other nodes.

Global recovery suffers from scalability limitations, delayed perception of dynamic environments, and delayed response in recovering from failures. Performing complex calculations for recoveries becomes challenging, especially considering the highly dynamic deployment of functions and their large-scale deployment. Additionally, continuous communication between the centralized controller and each node may lead to delays and increased overhead for recovery. If the centralized controller is distant from the cluster, the detection of topology status and issuing commands to each node can be time-consuming.

Local recovery enhances system responsiveness and efficiency, significantly reducing recovery times and avoiding the need to send large volumes of data to a central location, thereby reducing bandwidth consumption and network load. Additionally, this approach improves the scalability and flexibility of distributed systems, as each node's capacity to

Rui Kang and Mengfei Zhu contributed equally to this article.

independently manage faults facilitates expansion and adapts to increases in system size and complexity.

In local recovery however, each node can only observe the failures of the functions that are hosted by the node. Hence, each node cannot know the survival of the functions protected by the other nodes, which makes it difficult to make optimal decisions without time-consuming communication. The symmetry arising from multiple nodes concurrently attempting to recover the same function could lead to inconsistent recovery; hence, we seek to address symmetry breaking, preferably by establishing a certain order or priority, importantly, to achieve break symmetry for free [7] by leveraging inherent information without introducing additional communication overhead.

To this end, Schmid and Suomela [8] introduced SUPPORTED models to enhance distributed algorithms with preprocessing. It can reduce communication overhead by performing analysis on the input topology before distributing it to nodes, thereby reducing the amount of data that needs to be transmitted and communication rounds. Their model was applied to subgraph detection by Korhonen and Rybicki [9], stronger lower bounds and the computation needed for preprocessing in distributed computing by Foerster et al. [7], [10], labelings and independent sets [11]. However, the algorithms therein still need at least a constant non-zero number of rounds of communication to achieve their goals after preprocessing, and hence cannot be applied to immediate protection.

C. Research question and main contributions

As thus a question arises: is it possible to develop a local recovery policy for function recovery with preprocessing based on the SUPPORTED model, which allows each node to make a local recovery decision with *zero-round* communication, ensuring feasibility against current failures and improving future fault-tolerance? As a complement to a global and centralized recovery strategy, if the communications among nodes can be avoided to obtain the recovery policies in a distributed manner, the recovery time can be reduced in some cases. This paper formulates the problem and analyzes the conditions under which distributed failure recovery can be utilized with zero-round communication, ensuring that the failure recovery can be accelerated while maintaining the resilience optimality of the prioritized settings. In addition, this paper introduces and evaluates a testbed including the designed algorithms to realize the proposed strategy. Compared to other related work, we investigate the capabilities of the SUPPORTED model in the complex and practical scenario of shared protection.

II. Problem formulation

A. Problem definition

Given sets of functions and nodes, the backup resource allocation between functions and nodes, and the maximum available capacity of each node for recovery, the goal is to develop a local recovery policy allowing each node to independently recover failed functions and propose a preprocessing strategy that simplifies recovery, minimizes communication, and enhances fault tolerance.

B. Optimization problem modeling for fault-tolerance

Let N be the set of available nodes for hosting functions. Let F denote the functions protected by N. A backup resource allocation is given, which is expressed by a given binary value $b_{fn}, f \in F, n \in N$. If $b_{fn} = 1$, function $f \in F$ is protected by node $n \in N$; 0, otherwise. Let F_u denote the set of unavailable (failed) functions; F_a denotes the set of available (serviceable) functions; $F_u \cup F_a = F$. The current available capacity of node $n \in N$ for recovery is given by c_n, which is a non-negative integer and expresses the number of functions that can be recovered by it. Since the shared protection allows one available resource to protect against potentially multiple failures simultaneously, the available residual capacity c_n of a node may be smaller than the number of protected functions, which indicates that not all protected nodes can be recovered with insufficient capacity.

The recovery against failures should be determined without repetition and omission, i.e., multiple nodes cannot recover the same function and failed functions must be recovered by a node. We use a binary decision variable $r_{fn}, f \in F_u, n \in N$ to denote the recovery; it is set to 1 if node $n \in N$ is chosen for recovering the failed function $f \in F_u$; otherwise, 0.

The objective of the considered problem is to maximize fault-tolerance for potential future failures, i.e., the number of functions that are protected by the remaining capacities of nodes after the recovery from failures. $\rho_{fn}, f \in F_a, n \in N$ is a binary auxiliary variable for calculating the objective value, which describes one possible way of protection based on the optimal solution. If $\rho_{fn} = 1$, function $f \in F_a$ can be recovered by node $n \in N$; 0, otherwise. The problem is formulated by:

$$\max \sum_{f \in F_a} \sum_{n \in N} \rho_{fn} \tag{1}$$

$$\text{s.t.} \sum_{f \in F_a} \rho_{fn} \leq c_n - \sum_{f \in F_u} r_{fn}, \forall n \in N, \tag{2}$$

$$\rho_{fn} \leq b_{fn}, \forall f \in F_a, n \in N, \tag{3}$$

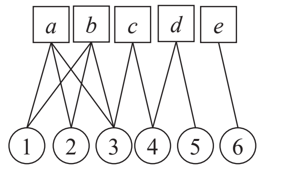
$$r_{fn} \leq b_{fn}, \forall f \in F_u, n \in N, \tag{4}$$

$$\sum_{n \in N} \rho_{fn} \leq 1, \forall f \in F_a, \tag{5}$$

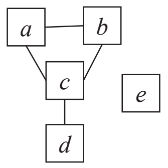
$$\sum_{n \in N} r_{fn} = 1, \forall f \in F_u. \tag{6}$$

The model is formulated as an integer linear programming (ILP) problem. Equation (2) ensures the capacity limitation after recovery. Equations (3) and (4) ensure that only the nodes hosting the backups of a function can recover the failure of the function. Equation (5) ensures that each recovery strategy can recover at most one function once. Equation (6) ensures that every failed function must be recovered.

The protection relationship between functions $f \in F$ and nodes $n \in N$ in the model can be depicted as a bipartite graph. We use N, F, and b_{fn} to build a protection graph $G = (F \cup N, B)$, which represents the protection relationship between N and F. N and F are two disjoint sets in the graph. B is the set of edges which connect N and F and represent the



(a) Protection graph for given backup resource allocation.



(b) Corresponding neighborhood graph.

Note: ○ represents a function; □ represents a node.

Fig. 1. Examples of protection graph and neighborhood graph and transformation between them.

protection relationships between nodes and functions. $b_{fn}, n \in N, f \in F$, is an element of B. If $b_{fn} = 1$, i.e., function f is protected by node n, there is an edge connecting $n \in N$ and $f \in F$. The number of functions protected by a node can be expressed by the degree of the node in the protection graph. An example is shown in Fig. 1(a).

III. Local recovery in the supported model

In this section, we analyze the conditions under which a feasible and optimal recovery solution can be obtained with zero-round and one-round communication so that we can reduce the average recovery time by local recovery based on preprocessing against failures. By conducting these analyses, we can perform graph decomposition to reduce the problem size. We classify the problems in terms of the number of simultaneous failures and explore the feasibility and optimality conditions of local recovery with communication. We also clarify which failure conditions allow for local recovery and which failure conditions require a trade-off between inter-node communication and communication with the central processor. This approach enables us to optimize recovery strategies and enhance overall system recovery efficiency by tailoring the response to different failure scenarios.

The feasibility considers node capacity limitation and successful recoveries avoids repetition and omission in (2)-(6); the optimality focuses on resilience with considering the effect of the local recovery to the fault-tolerance after failure recovery. It is defined as the number of functions that are protected by the node with remaining capacities after the recovery from current failures, i.e., the objective function of the proposed strategy described in (1). For the following positive and impossibility results, we only consider instances where a global algorithm can find feasible and optimal solutions - the difficulty then lies in the question if we can achieve the same goal by the aid of preprocessing.

A. Feasibility and optimality against single failure with zero-round communication

We consider local recovery with preprocessing based on the SUPPORTED model. Each node monitors the failures of the corresponding primary instance (the instance which is currently being served) of the functions protected by it; the states of the other functions cannot be detected. Each node decides locally whether to recover each of the protected functions or not. Without loss of generality, we assume that the recovery capacity of each node is larger than zero in the following theorems and proofs.

Firstly, precomputed optimal recovery policies for local recovery with preprocessing based on the SUPPORTED model ensure zero-round complexity due to initial backup resource allocation and non-preemptive node capacities.

We next briefly show that preprocessing is a crucial even for a single failure. Else, a failure-adjacent node cannot determine (without communication) whether to recover the failed function. One way to reason about this setting is with so-called orphans:

Definition 1. *If there exists a function $f \in F$ satisfying $\sum_{n \in N, c_n > 0} b_{fn} = 1$, we call f an orphan function. Next, we call the node that protects the orphan function protector. An orphan function has only one protector.*

Without preprocessing, each node which hosts a backup resource of the failed function needs at least one-round to determine if the function is an orphan function and communicate with the neighborhoods by exchanging the values of $c_n - d_n$, which is the difference of the remaining capacity and the degree of node n in the protection graph, for reaching a consensus to avoid repeat recovery, i.e., satisfy (6). As an example of the protection graph shown in Fig. 1((a)), if function 1 fails and both nodes a and b observe the failure. Without preprocessing, both nodes may locally make the same decision to recover or not to recover function 1 and lead to contention and infeasible recovery. We cast our insight into the following Theorem 1:

Theorem 1. *Without preprocessing, to achieve feasibility of the considered problem, one needs at least one round of communication, even for a single failure.*

B. Feasibility and optimality against two failures with zero-round communication

Different from the single failure case, multiple failed functions may cause the contention for recovery due to node insufficient recovery capacity. As the example shown in Fig. 1(a), if functions 1 and 2 fail and nodes a and b should make local decision to recover one function. Without a well-designed preprocessing, both of the nodes may recover the same function and lead to contention and infeasible recovery. Compared with recovery against a single failure, more rounds of communication for obtaining a solution to avoid contentions may be required in multiple failure cases. We analyze the conditions that feasible and optimal local recovery solutions can and cannot be obtained with zero-round communication for two failures as well as the feasibility guarantees and effectiveness of one-round communication.

Definition 2. *For function $f \in F$, we define $V_f = \{n | b_{fn} = 1, \forall n \in N\}$ as the set of nodes that protect function f. For a node $n \in N$, we define $P_n = \{f | b_{fn} = 1, \forall f \in F\}$ as set of functions protected by node n.*

As an example shown in Fig. 1(a), if functions 4 and 5 fail, node c can observe only one failure, and node d can observe two. In such cases, a priority should be given in the preprocessing. We next discuss some conditions and situations in which feasible and optimal recovery can be guaranteed.

Theorem 2. *For two failed function f_1 and f_2, if $|V_{f_1} \cap V_{f_2}| \geq 2$ and the protection graph consisting of f_1, f_2, and $V_{f_1} \cup V_{f_2}$ is a complete bipartite subgraph, a feasible and optimal recovery can be achieved with zero-round communication.*

Proof. All nodes that observe a failure also observe both failures and due to the theorem formulation, we can assume no further failures appeared elsewhere in the graph. Hence, as f_1 and f_2 are protected by the same set of nodes, all nodes in this set know that each node in the set sees both (i.e., all) failures. Hence, for each possible such failure appearance, we can precompute which nodes optimally recover the functions in question. □

We note that this idea can also be extended to more than two failures. On the other hand, without precomputation, the nodes would need at least some communication to agree on which nodes recover the functions in question.

Next, if we can assume (e.g, by topological constraints) that the occurred failures appear in different parts of the graph, then we can recover again two failures:

Theorem 3. *When both failed functions are not shared protected, i.e., $V_{f_1} \cap V_{f_2} = \emptyset$, local recovery can obtain feasible and optimal local recovery with zero-round communication.*

Proof. All nodes that observe a failure only observe a single failure and each node that observes a failure is aware of this fact. Hence, the nodes observing a failure can make use of Theorem 1. □

Again, this idea can be extended to multiple failures, as long as the theorem assumption holds. It is even possible to extend it to multiple (arbitrary) failures in general for some graph classes, e.g., if the protection graph is a ring, we can precompute a global orientation, w.l.o.g., clockwise, and each node protects its clockwise function neighbor.

Notwithstanding, we cannot recover two failures in general with zero rounds of communication:

Theorem 4. *It is not possible to always provide a protection strategy that protects against any two failures with zero round of communication.*

Proof. We give the proof by giving a counter-example. To this end, we construct a protection graph with three nodes n_a, n_b, n_c (each having a protection capacity of one) and six functions, one orphan f^o_a, f^o_b, f^o_c for each of n_a, n_b, n_c, and three functions f_{ab}, f_{bc}, f_{ca} that are connected to the nodes corresponding to their indices, respectively. As such, each node n_a, n_b, n_c has a degree of three in the protection graph. Next, we consider the two-failure patterns where one orphan fails and one function from f_{ab}, f_{bc}, f_{ca} fails. A protection strategy must protect against all of these $3 \times 3 = 9$ different failure patterns. In this protection graph, if function f_{ca} fails, then either the node n_c or the node n_a must protect it, but it can only protect it if its orphan does not fail. Assume that, w.l.o.g., the node n_a protects f_{ca} if the orphan f^o_a does not fail. Hence, in this case, n_c must not protect f_{ca} to not violate the problem formulation constraints. However, from the viewpoint of n_c, the two failure cases (1) f_{ca},f^o_b and (2) f_{ca},f^o_a are then indistinguishable without communication. In case (1), n_c *must not* protect f_{ca}, as a protects its by the earlier assumption (and n_b would protect f^o_b), but in case (2), n_c *must* protect f_{ca}, as n_a is the only node that can protect the orphan f^o_a. Hence, there is a contradiction, which finishes the proof. □

Next, while we cannot protect against any two failures with zero rounds of communication, we can extend the protection by using one round of communication.

Theorem 5. *If two failed functions are shared protected and $V_{f_1} \neq V_{f_2}$, in other words, at least one node can observe two failures and one node can only observe one failure, the local recovery can obtain* ***optimal*** *recovery policies within* ***one-round*** *communication.*

Proof. Each node v that sees only one failure has at least one neighbor w that sees two failures in the neighborhood graph. Hence, each such node v node that sees one failure, can be informed about the second failure within one round of communication by its neighbor w. As such, all nodes that locally observe at least one failure know about both failures, and we can precompute an optimal recovery policy that the nodes can apply after one round of communication, depending on which two failures appear. □

IV. Testbed

Whereas the previous section presented theoretical algorithmic and impossibility results, we next propose a testbed to translate these ideas into a practical setting.

A. Overview

The testbed is designed to evaluate the recovery mechanism introduced in the paper. The mechanism utilizes the local recovery solution executed by worker nodes as a supplement to the global recovery solution executed by a central controller of the cluster. The overall structure is shown in Fig. 2. In this testbed, we assume that the capacities of nodes are one.

The central controller plays the role of initializing function deployments, monitoring the statuses of all functions running in the cluster, and executing global recovery. *Initializer* reads the topology and protection relationships and creates primary and backup functions in the corresponding worker nodes in the beginning. *Daemon* communicates with worker nodes and monitors the statuses of functions by heartbeat packets from worker nodes. *Preprocessor* executes the pre-step of the local recovery, which processes the information from the *initializer* and sends the results to worker nodes via *daemon* for local recovery. *Global optimizer* calculates the recovery policies globally after detecting function failures and sends the new recovery policy to replace the results obtained by the local recovery solution in worker nodes if necessary.

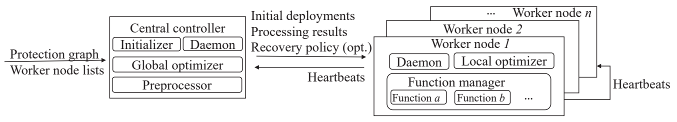


Fig. 2. Overall structure of testbed.

Worker nodes play the role of executing instructions from the central controller, lifecycle management of functions, monitoring the statuses of locally deployed functions, sending heartbeat packets, and executing local recovery. *Daemon* in a worker node monitors the statuses of local functions and communicates with the other elements in the testbed including sending and receiving heartbeat packets, receiving instructions and preprocessed information from the central controller. If any function which is protected by a worker node is not alive, the *local optimizer* calculates the local recovery policy and requests *function manager* to execute the recovery. *Function manager* executes the requirements from the other components and maintains the lifecycle of locally deployed functions by connecting to the platforms, e.g., Docker [12].

B. Local recovery algorithm

Algorithm 1 processes the input protection graph in the central controller and sends the results to corresponding worker nodes for Algorithm 2 for local recovery in worker nodes.

Algorithm 1 Preprocessing

```
Input: Protection graph $G(F \cup N, B)$, priorities for up to $K$ function failures.
Output: Processed information for node $n \in N$ including: $r^k_{nf}$ optimal recovery priority of function $f \in P_n$ in $k$-failure cases, $k \in [1, \cdots, K]$; neighbor nodes of $n$, $N'$, with $f \in P_{n'}, \forall n' \in N'$ and $r^k_{n'f}, \forall f \in P_{n'}, n' \in N', k \in [1, \cdots, K]$.
1: for $k \in [1, \cdots, K]$ do
2:     for $f \in F$ do
3:         Calculate the optimal solution for failed function $f$ under $k$-failures. $r^k_{nf} = 1$, if $n \in N$ is the optimal recovery policy of $f$ under $k$-failures; 0, otherwise.
4:     end for
5: end for
6: Assemble the processed information for each node referring the neighborhood relationships converted from the protection graph.
```

Algorithm 1 prepares two parts of information for the local decisions in Algorithm 2. Recovery priorities are used to avoid recovery contentions if multiple worker nodes protect the same function. When communication is prohibited, k is set to 1. In this scenario, each node does not receive information about failures observed by its neighbors. Each node can only know the failures of functions which it protected directly. As a result, when different priorities are assigned based on the failures observed by the node itself, it can lead to repetition or omission of recovery, and inefficient utilization of computational time during preprocessing. The neighbor information is used for communication if a node observes two failures whose priorities are the same if communication is allowed.

After failures are detected by node $n \in N$, Algorithm 2 is executed to find out the recovery policies of the failed functions directly connected to n.

Algorithm 2 Local recovery

```
Input: Current node $n$, processed information for node $n \in N$ from Algorithm 1 including: $r^k_{nf}$ optimal recovery priority of function $f \in P_n$ in $k$-failure cases, $k \in [1, \cdots, K]$; neighbor nodes of $n$, $N'$, with $f \in P_{n'}, \forall n' \in N'$ and $r^k_{n'f}, \forall f \in P_{n'}, n' \in N', k \in [1, \cdots, k]$; set of failed functions protected by $n$, $F'_n$.
Output: Recovery choices for failed functions $F'_n$.
1: if $|F'_n| \geq K + 1$ then
2:     Choose and recover $f \in F'_n$ whose first recovery priority is current node according to $r^1_{nf}$. If there are multiple functions need to be recovered, randomly choose one.
3: else
4:     If one-round communication is allowed, $n$ sends observed failures to its neighbors and waits until messages are received from all the neighbors which shared protects at least one failed function with $n$ and save the list of failures to $L$; otherwise, $L \leftarrow F'_n$.
5:     if $|L| \geq K + 1$ or $|L| = 1$ then
6:         Choose and recover $f \in F'_n$ whose first recovery priority is current node according to $r^1_{nf}$. If there are multiple functions need to be recovered, randomly choose one.
7:     else
8:         Choose and recover $f \in F'_n$ whose first recovery priority is current node according to $r^{|L|}_{nf}$.
9:     end if
10: end if
11: return obtained recovery policy.
```

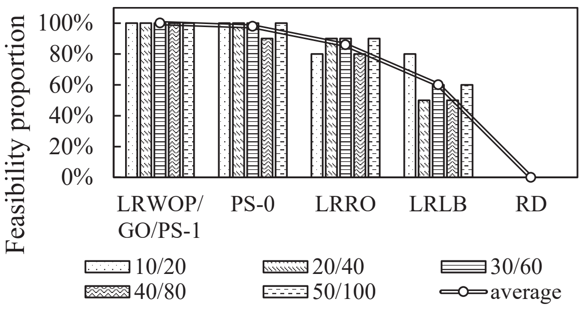
If there are more than K failures observed by n, the preprocessed information cannot ensure an optimal solution. To avoid unsuccessfully recovery caused by priority conflicts for different numbers of failures and time for one-round communication, the local algorithm attempts to recover the failures by using the priority r_{nf} for single-failure cases directly. If the number of failures whose priorities are all n, n chooses one to recover in order not to exceed its capacity. If there are no more than K failures protected by n, n collects the failure information from its neighbors. If n and its neighbors observe one failure or more than K failures, n recovers the failure if the priority to recover the failed function f is n according to the priority r^1_{nf} for single-failure cases obtained by Algorithm 1. If n and its neighbors observe less than K failures, n recovers the failure if the priority to recover the failed function f is n according to the priority r^k_{nf} for k-failure cases obtained by Algorithm 1. If n and its neighbors observe more than two failures, local algorithms attempt recovery using the priority r_{nf} for single-failure cases. If one-round communication is not allowed, the definition of L equals to F'_n.

V. Evaluation

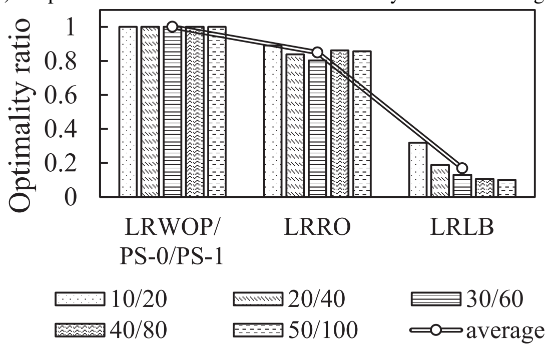
We evaluate the proposed strategy from different aspects. The evaluation is programmed in Python 3.9, running on an Intel Core i9-11900KB CPU with 64 GB memory.

A. Baseline strategies and settings

To evaluate the performances of the local recovery solution proposed in this paper, we introduce the following five common and state-of-the-art solutions as baseline models.



(a) Proportion of feasible solutions obtained by different strategies.



(b) Ratio of objective values of different strategies to optimal values obtained by GO.

Fig. 3. Comparison on different strategies in protection graphs of different **sizes**. x/y: there are x nodes and y functions in the protection graph.

- Random decision (RD): simulates how local recovery works without a policy. Each node detects the failures individually and decides whether to recover.
- Local recovery without preprocessing (LRWOP): nodes exchange information with neighbors to obtain a global view, using a global optimal algorithm to calculate recovery policies.
- Local recovery with random order (LRRO): based on the SUPPORTED model with a random recovery priority.
- Local recovery considering load-balancing (LRLB): We consider two baselines based on the SUPPORTED model for LRLB according to [13] and [3]. The priorities are set to minimize the utilization ratio against failures.
- Global optimization solution (GO)

The proposed strategy with zero-round and one-round communication are represented by PS-0 and PS-1, respectively. The proposed strategy maximizes the potential fault-tolerance after failure recovery. The objective values of LRWOP and GO are the number of recoverable functions in the remaining functions by following the global optimization algorithms. The constraints are the same with (2)-(6) in the proposed strategy. We compare the solutions obtained by the baselines and PS-0/PS-1 in terms of the objective value in (1) in randomly generated protection graphs, where the number of nodes and functions are given in advance. Each function is randomly connected to one node in order to ensure that each function is protected by at least one node. The other edges are created by following Erdős–Rényi model [14].

Recovery time is another metric including the transmission delay for communication between neighbors, computational time to calculate the recovery time, and the waiting time for messages from neighbors and controllers.

B. Evaluation of impact of size of protection graphs on recovery solutions and time

We consider five types of protection graphs with different numbers of nodes and functions. The probability of connecting a pair of node and function is set to 0.5. We randomly generate 10 graphs for each type. In each graph, two failures randomly occur. The results are shown in Fig. 3 and Table I.

GO and PS-1 obtain optimal solutions in all examined cases. PS-0 does not guarantee a feasible solution if the function to be recovered has a priority conflict, which occurs in 2% of all examined cases in this evaluation; it is consistent with the proof of Theorems 3 and 4. PS-0 guarantees that all obtained feasible solutions are optimal. LRRO obtains 94% feasible solutions and reaches 85% optimality on average in the examined cases. LRLB obtains 62% feasible solutions and reaches 17% optimality on average in the examined cases. RD does not obtain any feasible solution in the examined cases. LRRO and LRLB outperform RD significantly because of their priority settings with constraints (2)-(6).

In terms of the recovery time, LRWOP takes the longest recovery time to obtain the global topology and failure information and compute the global optimal solution on each node, which is 15 and 324758 times than that of PS-1 and PS-0, respectively. LRRO, LRLB, and PS-0 do not take time on communications or calculations. They only make decisions referring to the priorities obtained in the preprocessing phase, which reduces more than 99.99% recovery time. Compared with GO and PS-1, PS-0 reduce more than 99.99% recovery time with sacrificing 2% feasibility. As the graph size increases, the increase of time of GO and PS-1 is greater compared to PS-0. This is because PS-1 needs to communicate with a larger number of nodes, and GO needs to compute more nodes. From Fig. 3(a) and Table I, we can observe that the proportion of feasible solutions and recovery time do not vary significantly with increasing graph size in PS-0. It indicates that for relatively larger graphs, PS-0 has an advantage compared with GO and PS-1 which require longer time for computation and communication.

VI. Further related works on recovery

The protections for functions in previous research can be divided into two main categories depending on the method of resource usage: dedicated and shared protections. The dedicated protection ensures that the protected functions can always be recovered at a cost of high consumption and deployment costs. In response to the problems posed by the dedicated protection, the shared protection for virtual network has been introduced in [15]. The shared protection allows different functions to share the same resources to improve utilization and reduce deployment costs and energy consumption while presenting a challenge of resource scheduling. The work in [16] designed a model to schedule delay sensitive functions

TABLE I

COMPARISON OF RECOVERY TIME [S] USING DIFFERENT STRATEGIES IN DIFFERENT **SIZES** OF PROTECTION GRAPHS

Graph sizes†	LRWOP $(\times 10^{-1})$	LRRO/LRLB $(\times 10^{-6})$	GO $(\times 10^{-3})$	PS-0 $(\times 10^{-6})$	PS-1 $(\times 10^{-3})$
10/20	1.39	3.07	7.25	3.84	3.12
20/40	3.94	2.62	17.5	2.91	13.4
30/60	7.97	2.81	35.0	2.88	40.2
40/80	14.1	6.84	74.6	2.81	53.1
50/100	22.9	11.3	94.6	3.11	219
average	10.1	3.67	45.8	3.11	65.8

†: x/y: there are x nodes and y functions in the protection graph.

allowing resource sharing and preemption. Our work addresses the issues arising from shared protection in local recovery by prioritizing planning based on a SUPPORTED model.

Previous strategies for failure recovery include immediate recovery at the time of failure to adapt to dynamic scenarios. For instance, reinforcement learning (e.g., [17]) dynamically allocates resources for recovery, while heuristic algorithms (e.g., [18]) manage stateful function placement and switch traffic. Alternatively, global recovery policies calculated preemptively promise rapid recovery but may sacrifice accuracy (e.g., [19]). Others (e.g., [3], [4]) focus on deploying and managing primary and backup resources to prevent service interruptions. In contrast, our approach combines both strategies by preprocessing topology for protection relationships and using this data dynamically. This aims to minimize recovery time and ensure effective recovery.

VII. CONCLUSION

In distributed systems, the reliability and latency of each function are extremely crucial as they affect user experience. We considered the problem of how to determine a policy for each node to recover functions distributively that maximizes residual resilience. The failures should be recovered without repetition or omission. We formulated the problem as an ILP problem. We analyzed conditions for obtaining feasible and optimal local recovery solutions with zero-round or one-round communication against one and two failures. We proposed a local recovery strategy based on preprocessing with zero-round or one-round communication executed on each worker node. We designed a testbed with related algorithms to realize the proposed strategy. We evaluated the strategy compared with five baseline strategies. PS-0 cannot guarantee 100% feasibility for the cases with more than one failures, which is 98% on average in all examined cases with two failures. However, considering the recovery time, PS-0 is the only method whose recovery time is on the scale of 10^{-6} among other baselines, which is 3.11×10^{-6} seconds on average. If a certain level of infeasibility for the cases with two failures is allowed, PS-0 is the best choice, which reduces more than 99.99% recovery time with sacrificing 2% feasibility.

REFERENCES

[1] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *USENIX Symp. on Networked Syst. Design and Implementation (NSDI)*, 2014, pp. 459–473.

[2] S. Aidi, M. F. Zhani, and Y. Elkhatib, "On improving service chains survivability through efficient backup provisioning," in *Int. Conf. on Network and Service Management (CNSM)*. IEEE, 2018, pp. 108–115.

[3] M. Zhu, F. He, and E. Oki, "Resource allocation model against multiple failures with workload-dependent failure probability," *IEEE Trans. on Network and Service Management*, vol. 19, no. 2, pp. 1098–1116, 2022.

[4] R. Kang, F. He, and E. Oki, "Fault-tolerant resource allocation model for service function chains with joint diversity and redundancy," *Computer Networks*, vol. 217, p. 109287, 2022.

[5] V. Mushunuri, A. Kattepur, H. K. Rath, and A. Simha, "Resource optimization in fog enabled iot deployments," in *Second Int. Conf. on Fog and Mobile Edge Comp. (FMEC)*. IEEE, 2017, pp. 6–13.

[6] Z. Wen, R. Qasha, Z. Li, R. Ranjan, P. Watson, and A. Romanovsky, "Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-time failures," *IEEE Trans. on Cloud Computing*, vol. 8, no. 4, pp. 1093–1107, 2020.

[7] K.-T. Foerster, J. Hirvonen, S. Schmid, and J. Suomela, "On the power of preprocessing in decentralized network optimization," in *IEEE Conf. on Computer Commu. (INFOCOM)*. IEEE, 2019, pp. 1450–1458.

[8] S. Schmid and J. Suomela, "Exploiting locality in distributed sdn control," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 121–126.

[9] J. H. Korhonen and J. Rybicki, "Deterministic subgraph detection in broadcast CONGEST," in *2017 21st International Conference on Principles of Distributed Systems, OPODIS,*, J. Aspnes, A. Bessani, P. Felber, and J. Leitão, Eds.

[10] K. Foerster, J. H. Korhonen, A. Paz, J. Rybicki, and S. Schmid, "Input-dynamic distributed algorithms for communication networks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 1, pp. 06:1–06:33, 2021. [Online]. Available: https://doi.org/10.1145/3447384

[11] K. Foerster, J. H. Korhonen, J. Rybicki, and S. Schmid, "Does preprocessing help under congestion?" in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*, P. Robinson and F. Ellen, Eds.

[12] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[13] F. He and E. Oki, "Preventive priority setting against multiple controller failures in software defined networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 8, pp. 2352–2364, 2023.

[14] P. Erdős, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[15] T. Guo, N. Wang, K. Moessner, and R. Tafazolli, "Shared backup network provision for virtual network embedding," in *IEEE International Conference on Communications (ICC)*, 2011, pp. 1–5.

[16] Y. Zhang, F. He, T. Sato, and E. Oki, "Network service scheduling with resource sharing and preemption," *IEEE Trans. on Network and Service Management*, vol. 17, no. 2, pp. 764–778, 2019.

[17] R. Kang, F. He, and E. Oki, "Resilient virtual network function allocation with diversity and fault tolerance considering dynamic requests," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022, pp. 1–9.

[18] G. Yuan, Z. Xu, B. Yang, W. Liang, W. K. Chai, D. Tuncer, A. Galis, G. Pavlou, and G. Wu, "Fault tolerant placement of stateful vnfs and dynamic fault recovery in cloud networks," *Computer Networks*, vol. 166, p. 106953, 2020.

[19] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1253–1301, 2021.