IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 4, 2021, accepted June 10, 2021, date of publication June 17, 2021, date of current version June 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3090028

Stateless Node Failure Information Propagation Scheme for Stable Overlay Networks

KIMIHIRO MIZUTANI, (Member, IEEE)
Department of Informatics and Cyber Informatics Research Institute, Kindai University, Osaka 577-8502, Japan
e-mail: mizutani@info.kindai.ac.jp

This work was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant JP20K19791.

ABSTRACT A structured overlay technology has the advantages for fault tolerance and computation resource (i.e., node) discovery in distributed data storage and its computation platform, however, these strengths are only guaranteed on stable environment that node failures do not occur frequently. To deal with the environment, many advanced schemes based on the well-known node failure information propagation scheme are proposed, which stabilizes the platform by quickly handling node failures. In the existing scheme, a computation node propagates a node-failure information when the node detect its failure. However, the existing scheme needs stateful maintenance against propagation targets; in other words, it must maintain the network connections of both the propagation target nodes and the nodes held on the general overlay. The nodes then exhaust the machine resources (e.g., CPU, memory, network bandwidth) for the connection management and cannot concentrates on their own tasks, such as data analysis or its storage application. To resolve this problem, I propose a stateless node-failure information propagation scheme, which propagates a node failure at the speed of the existing scheme but without requiring maintenance of the propagation target connections. In the proposed scheme, each computational node can effectively utilize the machine resources for its own task. Instead of retaining the propagation targets, my scheme estimates the propagation targets after detecting a node failure. I analyzed the estimation accuracy of a simple propagation model, which guarantees effective propagation. The accuracy was found to depend on the overlay distance between the failed node and the propagator node. Based on this observation, my scheme adjusts the keep-alive interval to bias the detection of closer node failures. In a simulation evaluation, the detection delay of the proposed stateless propagation was similar to that of the stateful propagation scheme, but delivered superior maintenance cost and scalability.

INDEX TERMS Peer-to-peer computing, overlay networks, fault tolerance.

I. INTRODUCTION

A large scale deep learning architectures [1]–[3] and distributed key value store are famous use-cases of distributed computing technologies in current [4], [5]. In order to run their distributed computing platforms permanently, it is much important for the computation or storage nodes to keep the network connectivity among them. A structured overlay network satisfies the stability and connectivity requirements by providing an effective routing function for the distributed computing platforms. On such a network, the look up of a target node requires only $O(\log N)$ messages [6]–[10]. To realize effective routing, each node maintains a routing table containing only $O(\log N)$ nodes ID and addresses (i.e., pointer), along with the states (i.e., alive or dead). However, in a high-churn environment that many node insertions and failures, the maintenance costs of continuously updating the node states are very large. During a node insertion, the inserted node notifies its presence to the existing nodes, so the insertion process is safely completed. Conversely, a failed node does not notify its failure to the existing nodes, so the existing nodes must detect any failure through a keep-alive messaging process [11]. In this process, a source node sends a *SYN* message to a target node contained in the routing table and confirms the status of the node (alive or dead) by checking its response to an *ACK* message. If the target node does not reply to the *ACK* message within a specified time-out period, the source node detects that the

The associate editor coordinating the review of this manuscript and approving it for publication was Daniel Grosu.

target node has failed. Note that the time of detecting a node-A failure by all nodes is intractably large. In this paper, I refer to this delay as the detection delay.

Methods for improving the detection delay of node failure can be classified into two categories; detection schemes based on node behavior [12]–[15], and propagation schemes of node failure [19]–[24]. In the first category, a node estimates the failure pattern of other nodes based on a failure-pattern model, and adjusts the keep-alive interval based on the estimation results. These actions reduce the detection delay and the number of waste keep-alive messages in node-failure detection. However, as the failure model is not dynamically changed, the contribution of these schemes is limited.

Node-failure information propagation also effectively reduces the detection delay of node failure. In propagation schemes, a node detecting a node failure propagates the failure information to the other nodes containing the failed node state in their routing tables. Figure 1 shows the basic assumptions of a propagation scheme. Each node in the structured overlay maintains its routing table as a pointer list to shortcut nodes. In detail, each entry in the routing table contains the distance to the shortcut node and its pointer; accordingly, each routing table is sized $O(\log N)$. The shortcut node is referred among $O\log(N)$ nodes by a back pointer. When a node fails, all nodes containing its entry in their routing table must detect the failed node and replace it with a live one. For example, suppose that node k fails in Figure 1. Nodes b, e, and h are back-pointed from node k and contain the pointer to node k in their routing tables; accordingly, all three nodes must handle the failure of node k. In the propagation scheme, when node h detects the node-k failure, it propagates its failure information to the other nodes with back pointers from node k (i.e., nodes b and e). Because this propagation scheme does not depend on the node-failure pattern, its detection time is reduced from that of the abovementioned detection scheme. However, the propagation scheme adopts a stateful mechanism that forces a node to permanently maintain/update not only its routing table but also its propagation targets (i.e., back pointers). The increased number of holding nodes degrades the computational performance of this scheme on a computing platform.

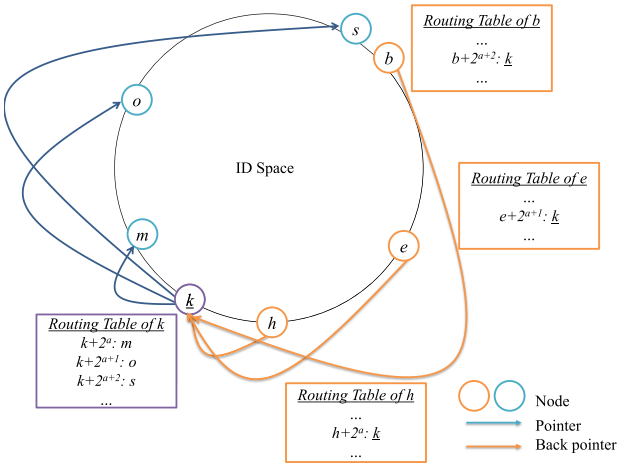


FIGURE 1. Basic assumptions of the node-failure information propagation scheme.

In this paper, I propose an effective ''stateless'' propagation scheme of node- failure information that detects an early node failure without maintaining the propagation targets. The proposed stateless propagation employs simple propagation and keep-alive interval adjustment (KIA) functions. The former function makes simple estimates of the propagation targets of a node, but the estimation error grows with increasing distance between the estimated target node and the detector; consequently, the propagation delay and number of waste propagations increase. After analyzing the propagation error, I found smaller errors when the detector propagates the failure information of nearby nodes. I thus propose a modified KIA function that adjusts the keep-alive frequency depending on the distance between the detector and a failed node. With this adjustment, a node will more likely detect a closer propagation target than farther targets. The propagation error is then reduced and the propagation is accelerated. In addition, the number of waste propagation messages caused by the estimation error is lowered, and the detection delay is improved. The detection delay of the proposed stateless scheme was compared with that of an effective stateful propagation scheme (i.e., SN+BPTR) on three structured overlays: Chord [6], Pastry [8], and Mercury [9]. The detection delay of the proposed scheme did not differ from that of SN+BPTR, but the total number of maintenance messages (e.g., for keep-alive and handling of node insertion/failure processes) was 10%–30% lower in the proposed scheme than in SN+BPTR. In the scalability evaluation, the stateless propagation was more scalable to high node numbers than stateful propagation. Therefore, my propagation scheme consumes lower computational resources than the stateful scheme, while also achieving high stability.

The remainder of this paper is organized as follows. Section II discusses related works on node-failure detection schemes. Section III describes the proposed propagation scheme and Section IV reports the experimental results. The paper concludes with Section V.

II. RELATED WORK

Many researchers have attempted to improve the delay of node failure detection and the maintenance overhead of structured overlays. As mentioned above, the existing researches are divisible into two categories: detection schemes based on node behavior, and propagation schemes of node failure.

The former scheme adjusts the keep-alive interval in response to node behavior. The authors of [12] modeled the node-failure probability as an exponential probability distribution. In their model, a node adjusts the timing of the keep-alive/refresh action of the node state. The model reduces the number of keep-alive instances and predicts node failures. The authors of [13] proposed three algorithms

(ProbKA, PredKA, and BugetProb) for reducing the number of keep-alive instances and enabling fast failure detection. ProbKA probabilistically determines the keep-alive timing, assuming a Weibull distribution of node alive intervals. PredKA determines the next keep-alive interval by considering the previous keep-alive result. This algorithm multiplicatively increments the keep-alive interval at each successful keep-alive. BugetProb reduces the keep-alive traffic into ProbKA and PredKA. The authors of [14] also proposed an adjustment scheme for the keep-alive interval, which minimizes the average detection delay of node failures. This scheme imposes a bandwidth threshold and assumes that the node insertion/failure intervals follow a bimodal distribution. Under this distribution assumption, they solved the minimization problem by the Lagrange multiplier method. The authors of [15] also modeled the node-failure interval, but assumed a Weibull distribution of intervals. However, these schemes improve the detection delay and number of waste maintenance messages only when the nodes behave as assumed in the models. To generate non-assumed model, the node behavior analysis based on neural network is proposed [16]. However, the neural network approach takes both high load computation and model database for each node, and the approach needs the high expertise adjustment of neural network parameters for realizing high prediction accuracy [17], [18].

Alternatively, node-failure propagation schemes propagate the node-failure information to other nodes. These schemes remove the dependence of the detection delay on the node-behavior model. The authors of [19] proposed SN+BPTR and another stateful propagation scheme called propagating both failure and current state information for back-pointer nodes (SNP+BPTR). The details of both propagation schemes are illustrated in Figure 2. In SNP+BPTR, suppose that nodes b, e, and h are back-pointed nodes of node k. Each of these nodes is maintained in the back-pointer list of node k. When node k fails and is detected by node h, node h refers to the back-pointer list of node k and propagates the failure to the other back-pointed nodes. In this scheme, each node holds $O(\log N)$ entries in its routing table, and each entry holds $O(\log N)$ back pointers in its back-pointer list. Therefore, there are $O(\log N^2)$ pointer/back-pointer node connections at each node. Note that a node in SNP+BPTR must maintain the keep-alive process not only of the nodes contained in its routing table, but also those of the back-pointer nodes. In addition, each node cooperatively shares its acquired keep-alive information. In contrast, although a node in SN+BPTR holds a back-pointer list (as in SNP+BPTR), it maintains no keep-alive for the back-pointed nodes. Therefore, SN+BPTR requires less memory and fewer network resources than SNP+BPTR. The propagation scheme in [20] is similar to SN(P)+BPTR, but the nodes in this scheme do not share their keep-alive information. Instead, each node actively searches the backward-pointed nodes through numerous search queries. In another node-failure information propagation scheme [21], the back-pointer list is not held by the back-pointed nodes (see Figure 3 for details). In this scheme, node k saves its back-pointer list to the closest neighbor node, and each back- pointed node from node k maintains this information (of node l) in its entry for node k. When a back-pointed node of node k detects a failure of node k, the detector finds the closest neighbor of node k in the routing table, retrieves the back-pointer list, and propagates node k's failure status to the back-pointed nodes. In other propagation schemes, the authors of [22] proposed a propagation method for specified Bamboo distributed hash table, and the authors of [23] proposed flooding propagation (i.e., broadcast). The authors of [24] presented propagation schemes for multiple structured overlays. When a node detects a node failure in a certain overlay, it shares the failure information with other overlays joined through the node.

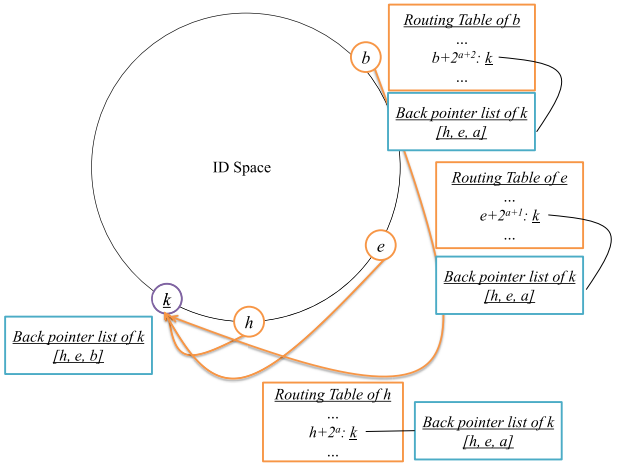


FIGURE 2. Model of node-failure information propagation scheme in [19].

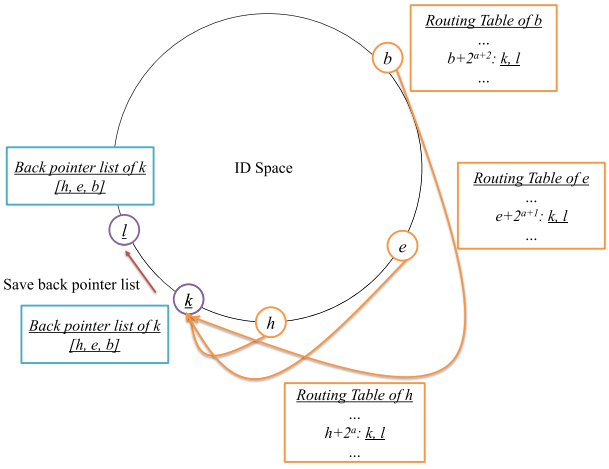


FIGURE 3. Model of node-failure information propagation scheme in [21].

The node-behavior schemes and node-failure propagation schemes have their own advantages and disadvantages.

The former scheme adjusts the keep-alive interval based on the node behaviors, which improves the detection delay and number of waste maintenance messages only when the node behaviors accord with the assumed model behaviors. Therefore, these schemes lack versatility. In contrast, the propagation schemes (with the exception of the models in [11] and [24]) remove the dependence of detection delay on the node-behavior model, but require stateful node management. Therefore, each node must maintain the back-pointed nodes of the other nodes (that is, the back-pointer lists of the other back-pointed nodes in [19], and the back-pointer list of the closest neighbor node in [21]).

As the maintenance cost (e.g., numbers of states and keep-alive messages) of the other back-pointer lists grows with number of nodes, the stateful propagation schemes lack scalability. For example, on a computing platform with several thousand nodes (the largest current case of distributed neural-network management [25]), each node must hold several tens to hundreds of node connections for maintaining the structured overlay in real-time. Such massive connections management discriminates the node's resources in general cases of high load computing platforms [26].

In this paper, I propose a stateless propagation scheme without back-pointer list management (Figure 4). Applying this scheme to the abovementioned case, each node maintains only dozens of nodes connections while realizing similar detection delay with simple, low-maintenance logic. Although its detection delay is similar to that of stateful failure propagation, it incurs lower maintenance cost and is more scalable to large node numbers than stateful propagation.

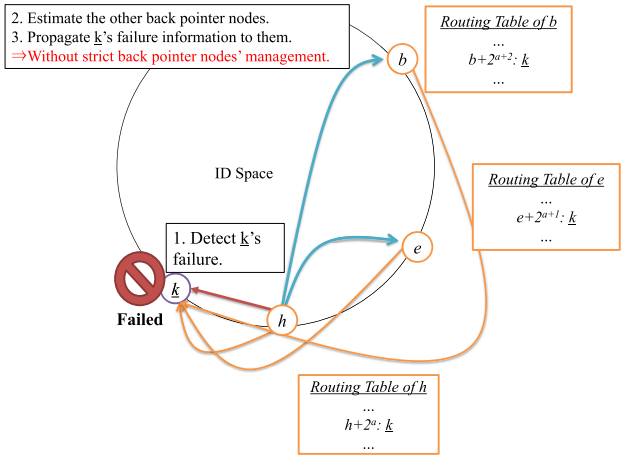


FIGURE 4. Model of node-failure information propagation scheme in proposal.

III. ALGORITHM OF THE STATELESS PROPAGATION SCHEME

This section describes the stateless propagation of a node failure. When a node detects a node failure, the detector estimates the IDs of the back-pointed nodes and propagates the failure information to the successor nodes responsible for the estimated IDs.

Here, I define the back-pointed nodes of node i's as B^i. The k-th back- pointed node in B^i is denoted by B^i_k, where the back-pointed nodes in B^i are sorted in order of their distance from node i. When a node j detects a failure of node i, it estimates the back-pointed nodes of node i as follows:

$$B^{i\,\prime}_k = suc(B^j_k + d(i,j)), \tag{1}$$

where $B^{j\,\prime}_k$, $suc(*)$ and $d(i,j)$ denote the estimated node (i.e., the propagation target), the successor of ID $*$, and the ID distance between nodes i and j, respectively. Figure 5 demonstrates the estimation of a failed node's back pointers, and propagation of the failed node information. When node h detects a failure of node k, it estimates the back-pointed nodes of k (excluding k itself). Then, node h can estimate the ID of the k-th back-pointed node from k by calculating $B^h_k + d(h,k)$. The ID's successor is then looked-up by general overlay routing. Overlay routing is the fundamental function of structured overlay and requires only $O(\log N)$ queries [6], [7], [10]. For example, in Figure 4, the look-up query for $B^h_1 + d(h,k)$ arrives at node e and that of $B^h_2 + d(h,k)$ arrives at node b. In this best-case estimation, all the failure information of node k arrives at all back-pointed nodes of node k. However, propagating this information to the correct back-pointed nodes is difficult in general, because invisible nodes from detector node h may exist there. For example, suppose that in Figure 5, nodes exist between node B^h_1 and node e. In this case, the successor of $B^h_1 + d(h,k)$ does not corresponded to node e.

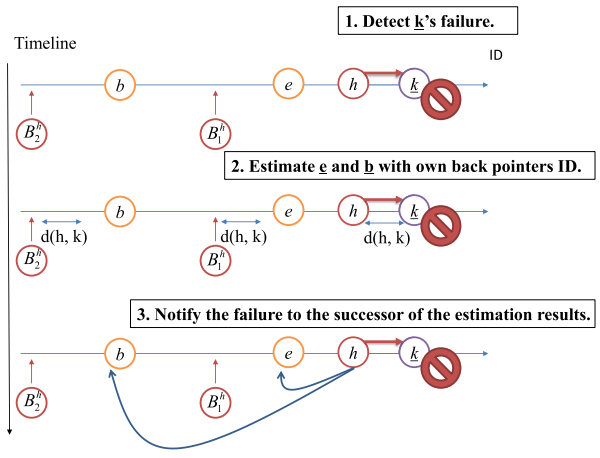


FIGURE 5. Basic estimation of the back pointers of a failed node, and propagation of the failure information to the estimated nodes.

Here, I investigated the estimation accuracy in a brief simulation of Chord, a fundamental structured overlay used in the previous Windows Azure architecture. The number of nodes was varied as 1,000 and 10,000. Failure operations were executed on each node set, and the back-pointed nodes of the failed node were estimated by Equation (1). The result

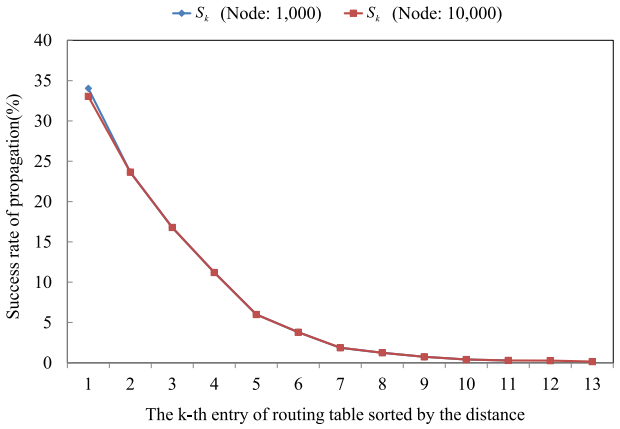


FIGURE 6. The success rate of stateless propagation for each entry of the routing table in Chord sustained by 1,000 and 10,000 nodes. In both node sets, the success probability S_k decreased as the propagation target moved farther from the failure.

is shown in Figure 6. The accuracy depended on the distance between the detector and the failed node; specifically, it improved as the detector moved closer to k. This result indicates that for successful propagation, a node should detect failures of closer nodes rather than those of farther nodes. To improve the estimation accuracy and propagation success, I proposed a keep-alive interval adjustment scheme in which closer node failures are more easily detected than further node failures.

A. KEEP-ALIVE INTERVAL ADJUSTMENT: KIA

In general, a node maintains a keep-alive process for the nodes registered in its routing table (i.e., pointed nodes), which is executed constantly in constant intervals H. The KIA is installed in each node, and its interval is adjusted for each entry in the routing table. Here, RT_i denotes the all entries in node i's routing table, and S_k denotes the estimation accuracy of the k-th ($k \leq |RT_i|$) entry of the routing table, as determined in Figure 6. With the abovementioned definitions, the adjustment policy of node i's KIA solves the following optimization problem:

$$\begin{cases} \min & \sum_{k\leq|RT_i|} H_k \cdot S_k \\ \text{s.t.} & \sum_{k\leq|RT_i|} \dfrac{1}{H_k} = \dfrac{|RT_i|}{H}, \end{cases} \tag{2}$$

where H_k denotes the calculated keep-alive interval of the k-th routing table entry, obtained by solving this optimization problem. The found keep-alive interval minimizes the expected values of the success probability. More concretely, KIA minimizes the interval in the entry because the failure information of an entry node propagates more successfully when k is lower. Under the constraint condition, the total number of keep-alive processes per second in KIA equals the number of keep-alive processes in the constant keep-alive interval. Specifically, if $\frac{1}{H_k}$ is the number of keep-alive instances per second, the total number of keep-alive instances per second at node i is $\frac{|RT_i|}{H_k}$.

This problem can be solved by Lagrange multipliers. Here, I define the Lagrange multiplier as λ and the $\sum_{k\leq|RT_i|} H_k \cdot S_k$ as f. Writing $F = f - \lambda \sum_{k\leq|RT_i|} \frac{1}{H_k}$, the gradient vectors of F are given by

$$\begin{pmatrix} \dfrac{\partial F}{\partial H_1} \\ \dfrac{\partial F}{\partial H_2} \\ \cdot \\ \cdot \\ \dfrac{\partial F}{\partial H_{|RT_i|}} \end{pmatrix} = \begin{pmatrix} S_1 + \dfrac{\lambda}{H_1^2} \\ S_2 + \dfrac{\lambda}{H_2^2} \\ \cdot \\ \cdot \\ S_{RT_i} + \dfrac{\lambda}{H_{|RT_i|}^2} \end{pmatrix} = 0. \tag{3}$$

Using this matrix under the constraint condition 2, the optimal keep-alive interval of each H_k can be expressed by

$$H_k = \frac{H \cdot \sum_{j\leq|RT_i|} \sqrt{S_j}}{|RT_i| \cdot \sqrt{S_k}}. \tag{4}$$

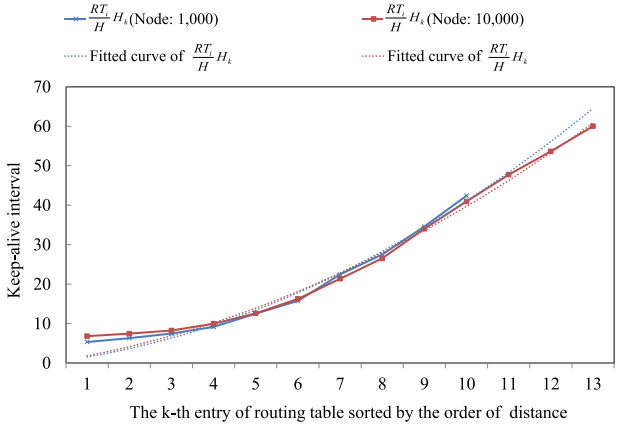


FIGURE 7. Keep-alive interval H_k in the 1,000- and 10,000-node Chord networks. In the networks of both sizes, the normalized H_k based on $\frac{|RT_i|}{H}$ is well fitted by a quadratic function.

Figure 7 plots the H_k (normalized $\frac{|RT_i|}{H}$) values determined from the simulation results of S_k (Figure 6). I found that the interval H_k in the k-th routing table depends on the squared logarithmic distance between a detector and its pointer node (Fitted curve of $\frac{|RT_i|}{H}H_k$ in Figure 7). Hence, I can conclude the fitting curve is calculated with following terms,

$$H_k \propto \log^2 d(k,i) \quad k \in RT_i. \tag{5}$$

Therefore, we can set the keep-alive interval based on this approximated H_k. With this approximation, when KIA sets the keep-alive interval H_k for the k-th entry in the routing table, it executes more frequent keep-alive processes of closer nodes than of farther nodes.

Algorithm 1 KIA Function of Node i

Require: RT_i, H
  for $k \in RT_i$ **do**
    $H_k \leftarrow \frac{H}{|RT_i|} \cdot \log^2 d(i,k)$
    setKeepAliveInterval(H_k, k)
  end for

Algorithm 2 Node $j's (j \in RT_i)$ Failure Information Propagation Function of Node i

Require: B^i, j
  for $k \leq |B^i|$ **do**
    target $\leftarrow B^i_k + d(j,i)$
    suc = findSuccessor(target)
    notifyFailure(suc, j)
  end for

The entire propagation process are summarized as algorithm 1 and 2. The algorithm 1 is executed when node i sets the keep-alive interval of each routing table entry. The H_k is normalized at each second with a constant keep-alive interval H set by the system administrator. The algorithm 2 is executed when node i detects the node j's failure ($j \in RT_i$). The algorithm estimates the all back pointers of the failure node, finds their successors, and finally notify the failure information for them.

When a node detects a node failure, the computational effort of determining the propagation targets by Equation (1) is $O(\log N)$, because the number of propagation targets is $O(\log N)$ in general structured overlays. Therefore, the computational effort negligibly affects the maintenance effort of the structured overlay network. In addition, the computational effort of KIA is only $O(RT_i)$, and the keep-alive interval is easily computed.

On the computing platform with the 1,000,000-node structured overlay (assumed as the largest computing platform in current use), the number of the routing table entries and back-pointer list entries are several tens to hundreds in the conventional schemes. In these schemes, the node continuously executes keep-alive processes and maintains their network connections. In my proposed scheme, which requires no back-pointer list, the number of network connections is around 10, yet the propagation speed of the node-failure information matches those of the conventional schemes. The proposed scheme therefore provides high stability at low computational cost (e.g., maintenance cost) on the largest existing computing platform.

IV. PERFORMANCE EVALUATION

This section compares the maintenance costs, detection delays of four schemes: the general scheme, SN+BPTR (i.e., stateful propagation), stateless propagation without KIA, and stateless propagation with KIA. Evaluations were performed on three key value store computational platforms based on the Chord, Mercury, and Pastry overlays. All evaluations were executed by the following steps on an original simulator.

1) The simulator generates 10,000 nodes with a set keep-alive interval.
2) The simulator inserts the nodes into the Chord, Mercury, and Pastry computational platforms.
3) In each node insert/failure interval, the simulator executes the corresponding action on the nodes.

In my proposed scheme (named stateless with KIA in the evaluation figures), the interval in the routing table entries was changed by KAI. In all other schemes, the all keep-alive interval was set to 1,000 seconds by the simulator. The average interval of node insertion/failure was varied as 100, 500, 1,000, 5,000, and 10,000 seconds. Node insertions and failures occurred simultaneously, ensuring a constant total number of nodes in the computation platform [11], [28]. The ID of each node in key value store computation platform was determined by the uniform or Mandelbrot–Zipf [27] distribution with $\alpha = 0.62$ and $q = 5$. Here, the Mandelbrot–Zipf parameters were selected to fit the measured distributions of real distributed platform's traffic flows. The round trip time (RTT) through all nodes was randomly selected from a uniform distribution in the interval (0, 2] seconds, meaning that a keep-alive process takes 1 RTT. The node placement and node insertion/failure rates on the simulated computation platform were those given in previous studies [11], [27], [28]. In this environment, Step (3) was iterated for 24 hours, and the average detection delay and maintenance cost of each scheme were evaluated in each overlay on each computational platform.

A. DETECTION DELAY

Figure 8 shows the average detection delay in each propagation scheme on the different overlays with different distributions of nodes. Although the propagation speed was faster in SN+BPTR than in the stateless propagation schemes, stateless propagation with KIA was only a few seconds

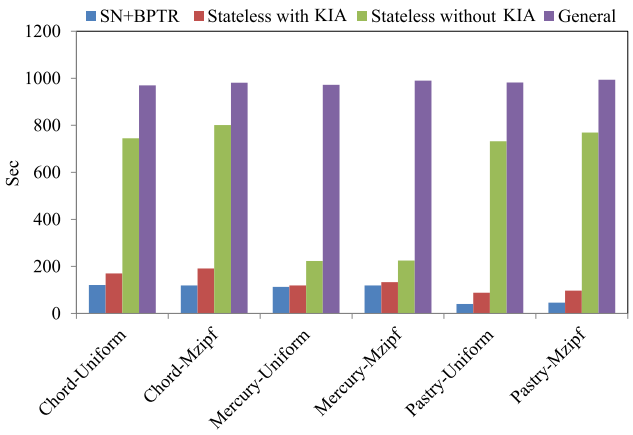
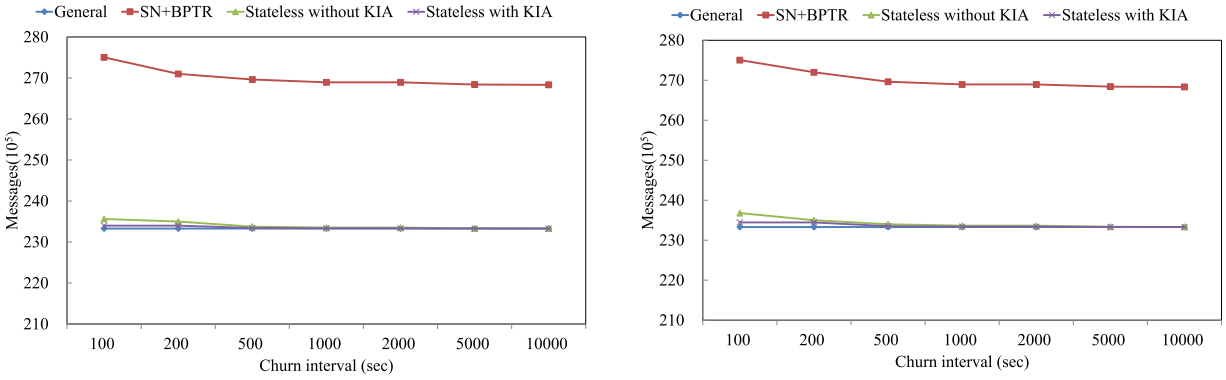


FIGURE 8. Average detection delays of both detecting node failure and its failure information propagation.



(a) *Chord in uniformed distribution*

(b) *Chord in Mandelbrot-Zipf*

(c) *Mercury in uniformed distribution*

(d) *Mercury in Mandelbrot-Zipf*

(e) *Pastry in uniformed distribution*

(f) *Pastry in Mandelbrot-Zipf*

FIGURE 9. Total number of maintenance messages versus time (monitored over 24 hours).

slower than SN+BPTR. The stateless scheme without KIA was slower because its propagation accuracy was lower than in the stateless scheme with KIA. According to these results, the node-failure information was propagated with a few seconds delay in stateless propagation. This delay was smaller than in the general detection scheme and the stateless scheme without KIA.

B. MAINTENANCE COST

Figure 9(a) - 9(f) show the total numbers of messages required for the keep-alive processes and failure propagations in each propagation scheme on the different overlays with the uniform and Mandelbrot–Zipf distributions. In the keep-alive operation, the sender's SYN and the receiver's ACK messages are transmitted via a pair of nodes, so each

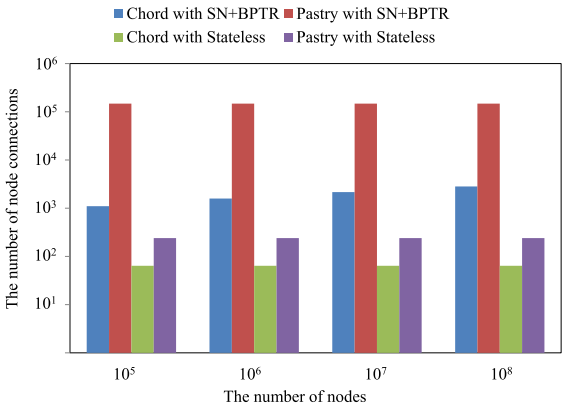


FIGURE 10. Number of network connections versus number of nodes in the compared methods on different overlays.

keep-alive operation generates two messages. During propagation, a detector node propagates $O(\log N)$ back-pointed nodes. Note that the propagation is recursively completed for all back-pointed nodes. In the Chord and Mercury overlays with the uniform distribution, the number of messages was $10 - -15\%$ lower in both stateless schemes than in SN+BPTR, and was $1-2\%$ in the Mandelbrot–Zipf distribution than in the uniform distribution. The latter is attributable to the larger estimation error in the Mandelbrot–Zipf distribution than in the uniform distribution results. This is because the node ID's density is concentrated on a certain area in overlays, therefore, a number of nodes existed between a detector and an estimated node. On the Pastry overlay, the number of messages in both stateless propagations was approximately 30% lower than in SN+BPTR, regardless of node distribution. This result can be explained by the larger size of the routing table on Pastry than on the other overlays [8], which must be maintained by the nodes during stateful failure propagation. More specifically, the routing table size is approximately $\log L$ on Chord, but $\frac{L(2^b-1)}{b}$ $(b > 0)$ on Pastry. Therefore, a node maintains a large number of states on a computational platform with the Pastry overlay, and must perform many keep-alive messages and insertion/failure processes.

These results confirm the lower maintenance costs in stateless propagation than in SN+BPTR and the general detection scheme (in which the costs were quite similar). Therefore, stateless propagation incurs fewer overheads than general detection schemes, and places a non-critical cost burden on a platform's computational processes.

C. SCALABILITY

Finally, I evaluated the scalability of each detection scheme by increasing the number of nodes in the network. Figure 10 plots the number of network connections (e.g., in transmission control protocol sessions) versus number of nodes in each overlays. The SN+BPTR scheme formed over 1,000 connections in the Chord overlay, and over 100,000 nodes in the Pastry overlay. The number of connections depended on the features of the overlays, and increased when the routing tables enlarged. The Chord overlay required fewer network connection than Pastry. In addition, SN+BPTR also takes much number of connections in the overlays: epichord [29] and d1ht [30]. Unlike stateful propagation, the stateless propagation scheme required no additional network connections because they do not maintain the back-pointer list of each routing table entries (as explained for Figure 2). Therefore, the stateless propagation scheme is scalable to large node networks without compromising its fast failure detection.

TABLE 1. Quality comparison of the detection schemes (circles: quality achieved; crosses: quality not achieved).

	Detection delay	Messages	Scalability
General	×	◯	◯
Stateful	◯	×	×
Stateless without KIA	×	◯	◯
Stateless with KIA	◯	◯	◯

D. SUMMARY OF THE EXPERIMENTS

The above evaluations are summarized as Table 1. The stateful propagation scheme (SN+BPTR) must handle many messages in networks with large-size routing tables. In addition, it requires a large number of network connections to realize fast propagation. On the other hand, the stateless propagation schemes (i.e., stateless with/without KIA) require fewer messages and network connections than the stateful scheme. However, stateless propagation without KIA takes estimation error in the propagation target, which increases the detection delay of notifying a node failure to all back-pointed nodes. Meanwhile, the stateless propagation scheme with KIA improved the detection delay and shortened the delay detection to that of stateless propagation. This is because it refrains keep-alive messaging for high estimation error targets. We conclude that stateless propagation with KIA achieved efficient node-failure detection with low overheads; moreover, its performance met the requirements of large-scale computing platforms (namely, low memory consumption and small network overhead).

V. CONCLUSION

I proposed a stateless propagation scheme for transferring node-failure information. Unlike the stateful propagation schemes, the overlay nodes in the proposed scheme do not maintain the statuses of the propagation target nodes. In addition, the stateless propagation scheme calculates the optimal propagation policy that improves the propagation delay and the waste propagation. In the performance evaluation, the stateless propagation schemes required 10%–30% fewer overlay maintenance messages than stateful propagation. Although the detection delay of stateless propagation was several seconds longer than that of stateful propagation,

the scalability evaluation confirmed that (unlike stateful propagation) stateless propagation was upscalable to huge networks (such as the 100,000-node computational platform). I conclude that the stateless propagation improves the scalability and maintenance costs over those of the stateless propagation scheme, and is useful for constructing efficient large-scale computing platforms.

REFERENCES

[1] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement.*, 2014, pp. 583–598.

[2] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[3] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 344–353.

[4] J. Dhanani, R. Mehta, D. Rana, and B. Tidke, "Back-propagated neural network on MapReduce frameworks: A survey," in *Smart Innovations in Communication and Computational Sciences*. Singapore: Springer, 2019, pp. 381–391.

[5] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. ACM SIGCOMM*, 2001, pp. 149–160.

[7] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Peer-to-Peer Systems* (Lecture Notes in Computer Science), vol. 2429. Springer, 2002, pp. 53–65.

[8] A. Rowstron and D. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware* (Lecture Notes in Computer Science), vol. 2218. Springer, 2001, pp. 329–350.

[9] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," in *Proc. Int. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2004, pp. 353–366.

[10] J. Aspnes and G. Shah, "Skip graphs," *ACM Trans. Algorithms*, vol. 3, no. 4, p. 37, Nov. 2007, doi: 10.1145/1290672.1290674.

[11] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. USENIX Annu. Tech. Conf.*, 2004, pp. 127–140.

[12] G. Ghinita and Y. M. Teo, "An adaptive keep-alive framework for distributed hash tables," in *Proc. Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2006, pp. 1–10.

[13] R. Price, P. Tiňo, and G. Theodoropoulos, "Still alive: Extending keep-alive intervals in P2P overlay networks," *Mobile Netw. Appl.*, vol. 17, no. 3, pp. 378–394, Jun. 2012.

[14] K. C. W. So and E. G. Sirer, "Latency and bandwidth-minimizing failure detectors," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 89–99, Jun. 2007.

[15] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. 6th ACM SIGCOMM Internet Meas. (IMC)*, 2006, pp. 189–202.

[16] R. Kaur, A. L. Sangal, and K. Kumar, "Modeling and simulation of adaptive neuro-fuzzy based intelligent system for predictive stabilization in structured overlay networks," *Eng. Sci. Technol., Int. J.*, vol. 20, no. 1, pp. 310–320, Feb. 2017.

[17] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, "Artificial neural network architectures and training processes," in *Artificial Neural Networks*. Springer, 2017, pp. 21–28.

[18] J. Fu, P. Liu, and Q. Zhang, "Rethinking generalization of neural models: A named entity recognition case study," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 7732–7739.

[19] S. Q. Zhuang, D. Geels, I. Stoica, and R. H. Katz, "On failure detection algorithms in overlay networks," in *Proc. IEEE 24th Annu. Joint Conf. IEEE Comput. Commun. Societies. (INFOCOM)*, 2005, pp. 2112–2123.

[20] I. Dedinski, A. Hofmann, and B. Sick, "Cooperative keep-alives: An efficient outage detection algorithm for P2P overlay networks," in *Proc. 7th IEEE Int. Conf. Peer-Peer Comput. (P2P)*, Sep. 2007, pp. 140–150.

[21] K. Mizutani, T. Inoue, T. Mano, O. Akashi, S. Matsuura, and K. Fujikawa, "Living will for resilient structured overlay networks," *IEICE Trans. Commun.*, vol. E99.B, no. 4, pp. 830–840, 2016.

[22] (2005). *BambooDHT*. [Online]. Available: http://bamboo-dht.org/

[23] Z. Yao, D. B. H. Cline, X. Wang, and D. Loguinov, "Unifying models of churn and resilience for unstructured P2P graphs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2475–2485, Sep. 2014.

[24] K. Mizutani, S. Matsuura, S. Doi, K. Fujikawa, and H. Sunahara, "An implementation and its evaluation of a framework for managing states of nodes among structured overlay networks," in *Proc. 6th Int. Conf. Netw. Services (ICNS)*, Mar. 2010, pp. 282–287.

[25] Google Official Report. *Using Large-Scale Brain Simulations for Machine Learning and A.I.* Accessed: 2012. [Online]. Available: https://blog.google/topics/machine-learning/using-large-scale-brain-simulations-for/

[26] Z. Zhang, Y. Gu, F. Ye, H. Yang, M. Kim, H. Lei, and Z. Liu, "A hybrid approach to high availability in stream processing systems," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Jun. 2010, pp. 138–148.

[27] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1447–1460, Dec. 2008.

[28] K. Shudo, "Churn resilience improvement techniques in an algorithm-neutral DHT," *IPSJ J. Comput. Syst.*, vol. 49, no. SIG 2(ACS 21), pp. 1–9, 2007.

[29] B. Leong, B. Liskov, and E. D. Demaine, "EpiChord: Parallelizing the chord lookup algorithm with reactive routing state management," *Comput. Commun.*, vol. 29, no. 9, pp. 1243–1259, May 2006.

[30] L. R. Monnerat and C. L. Amorim, "D1HT: A distributed one hop hash table," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2006, pp. 1–10.

KIMIHIRO MIZUTANI (Member, IEEE) received the M.S. and Ph.D. degrees from the Nara Institute of Science and Technology, in 2010 and 2015, respectively. He was a Researcher with the NTT Group (Network Innovation Laboratories and West Research and Development Center), from 2010 to 2019. He is currently a Lecturer and a Senior Assistant Professor (Principal Investigator) with the Department of Informatics, Kindai University. His research interests include future network architectures and various systems powered by deep learning. He was a recipient of the Best Student Paper Award from ICCSA and the Research Awards from IEICE, in 2010 and 2013, respectively.