

SIoTFog: Byzantine-resilient IoT fog networking^{*}

Jian-wen XU¹, Kaoru OTA¹, Mian-xiong DONG^{‡1}, An-feng LIU², Qiang LI³

¹Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran 0508585, Japan

²School of Information Science and Engineering, Central South University, Changsha 410083, China

³MOE Key Laboratory of Symbol Computation and Knowledge Engineering, Jilin University, Changchun 130012, China

E-mail: {17096011, ota, mxdong}@mmm.muroran-it.ac.jp; afengliu@mail.csu.edu.cn; li_qiang@jlu.edu.cn

Received Aug. 31, 2018; Revision accepted Nov. 18, 2018; Crosschecked Dec. 17, 2018

Abstract: The current boom in the Internet of Things (IoT) is changing daily life in many ways, from wearable devices to connected vehicles and smart cities. We used to regard fog computing as an extension of cloud computing, but it is now becoming an ideal solution to transmit and process large-scale geo-distributed big data. We propose a Byzantine fault-tolerant networking method and two resource allocation strategies for IoT fog computing. We aim to build a secure fog network, called “SIoTFog,” to tolerate the Byzantine faults and improve the efficiency of transmitting and processing IoT big data. We consider two cases, with a single Byzantine fault and with multiple faults, to compare the performances when facing different degrees of risk. We choose latency, number of forwarding hops in the transmission, and device use rates as the metrics. The simulation results show that our methods help achieve an efficient and reliable fog network.

Key words: Byzantine fault tolerance; Fog computing; Resource allocation; Internet of Things (IoT)
<https://doi.org/10.1631/FITEE.1800519>

CLC number: TP393

1 Introduction

In recent years, we have witnessed the boom in the Internet of Things (IoT) and the hypergrowth of cloud computing, which again overturned our perception of information technology. By 2020, more than 20 billion IoT devices will be manufactured and put into use after a 15% annual increase (IHS Markit, 2017). Originally, as an extension of cloud computing, fog computing relied on the collaborative end-user clients or near-user edge devices to provide a substantial amount of storage capacity and communication solutions. Now, fog has already become a research hotspot, not only broadening our perspective in distributed computation, but also providing brand


new ideas to exploit the potential of “Things” besides the “Internet.”

Byzantine fault tolerance (BFT) describes the dependability of fault-tolerant computing systems, especially distributed ones. The problem of Byzantine generals or BFT was first proposed by Lamport et al. (1982). In the BFT, a group of generals is trying to reach an agreement to decide whether to attack enemies or retreat from them according to their votes in the majority. Considering the appearances of messengers or the presence of traitors who want to disrupt the whole group, the final agreement may run in an opposite direction of the loyal generals’ original intentions. A Byzantine fault is the inconsistency of different messages that the generals received from a single general, and the Byzantine failure is the system malfunction caused by a Byzantine fault.

Occurrence of Byzantine faults can be very common in distributed systems, such as fog networks. Sometimes fog nodes may fail, and there is imperfect information about whether a particular node has failed. The only way to solve this problem is to find

[‡] Corresponding author

^{*} Project supported by the JSPS KAKENHI, Japan (No. JP16K00117) and the KDDI Foundation, Japan

 ORCID: Mian-xiong DONG, <http://orcid.org/0000-0002-2788-3451>

© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2018

the failed node. However, we cannot ask a running distributed system to stop and troubleshoot all nodes. Instead, a relative compromise is a solution to a fault-tolerance mechanism. That is, what we prefer to do is to cope with BFT while introducing as little impact as possible to the network computing performance.

In this study, we focus on the issue of BFT in resource allocation of fog computing for IoT applications. Fault tolerance enables a system to continue to work when some of its components go down. Therefore, a good fault-tolerance performance can greatly tolerate the interruption of retransmissions in network communications and reduce extra energy consumption and time costs.

In fog computing, large central servers are carried out by a massive number of geo-distributed small- and medium-sized fog devices at the edge of the network structure. Thus, rather than setting dedicated standby replicas for all fog devices, we can simply make the fog devices help each other for state machine replication. Thus, a fog device can serve as the replica of its neighbor to tolerate the influence of a possible Byzantine fault. Taking the mobility of IoT devices into account, the relationship between replicas and primary devices can also change while the entire network is running. Therefore, we need a dynamic resource allocation strategy to solve the BFT in fog computing. The main contributions of our work are as follows:

1. A three-tiered heterogeneous fog network model is designed, in which the fog device routers can provide services to IoT users, such as sensors, smart devices, and vehicles.
2. A Byzantine-resilient fog networking method and a two-resource allocation strategy are proposed to tolerate the influence of Byzantine faults.
3. The cases of a single Byzantine fault and multiple faults are considered to test the performance of the methods when facing different degrees of risk.
4. Total latency, number of forwarding hops in the transmission, and the device use rates are chosen as the metrics for analysis of the simulation results.

2 Related work

In this section, we present the related work on fog computing and the BFT problem.

2.1 Fog computing

Fog computing first served as an extension of cloud computing as a way to share responsibility to data storage and process at the edge of the network structure by Cisco Systems Inc. (Bonomi et al., 2012). Vaquero and Rodero-Merino (2014) from the Hewlett-Packard Company (HP) offered a comprehensive view of fog computing and correlated it with existing technologies, such as cloud, sensor networks, peer-to-peer networks, and network virtualization function (NFV), to reach a definition of the “fog.” Satyanarayanan et al. (2009) and Satyanarayanan (2017) conducted mobility-enhanced small-scale instances of cloud datacenters and the cloudlet to mobile edge computing (MEC) in IoT. Liu et al. (2016) focused on streaming media in heterogeneous edge networks and proposed a device-to-device relay-assisted scheme to solve video frame recovery for picocell edge users. Tao M et al. (2017) integrated cloud and fog computing to build a hybrid network model for vehicle-to-grid (V2G) and the 5th generation wireless systems services (5G). Stojmenovic and Wen (2014) analyzed the real-world application scenarios of the fog, such as in smart grids, smart traffic, and software-defined networks (SDNs). In these scenarios, the man-in-the-middle attack is regarded as a typical security issue to represent new features in the fog. Yi et al. (2015) focused on the new security and privacy challenges besides those inherited from the cloud, and proposed ideas for solutions. Alrawais et al. (2017) considered the fog and the IoT as a whole and put forward a mechanism to improve the distribution of certificate revocation information to enhance the security among IoT devices in the fog. Li et al. (2018) introduced deep learning to solve problems in edge computing. Hu et al. (2017) addressed face identification and resolution technology, and implemented a prototype system to evaluate their proposed security and privacy preservation method.

Compared with cloud computing, fog computing was originally intended to share the high load of a central architecture and to save the extra costs that occur between cloud servers and IoT devices at the edge of a network. Jalali et al. (2016) believed that fog computing could reduce the energy consumption compared with cloud computing. Tao XY et al. (2017) investigated the energy efficiency in mobile-edge computing and applied a request offloading scheme to

improve the performance of energy consumption and bandwidth capacity. Perera et al. (2017) studied the existing research and the problems in fog computing for sustainable smart cities. Castillo-Cara et al. (2018) put forward a fog-node design to deal with the energy consumption problem and network resilience provisioning in wireless sensor networks (WSNs). Zeng et al. (2018) studied how to explore energy generation diversity in a cyber physical fog system (CPFS) while considering source rate control, service replica deployment, and load balancing. Wu et al. (2018b) combined information-centric networks (ICNs) with designing content awareness filtering to increase the safety factor of fog computing.

2.2 Byzantine fault tolerance problem

Fault tolerance refers to the property that no global errors or interruptions occur in a system because of local faults. Therefore, fault-tolerant design is very common and important in fields related to an overall system structure (Khosravi and Kavian, 2016; Gao et al., 2017; Zhang et al., 2018). Since BFT was first proposed by Lamport et al. (1982), it has been studied for decades. Castro and Liskov (2002) first explored in depth the practice of BFT and implemented a generic program library and the first BFT network file system (NFS). Their experiment results showed that an NFS with BFT, i.e., a BFS, performs better than the NFS protocol without replicas. Driscoll et al. (2003, 2004) redefined the concepts of Byzantine problems, including the widely known existence of Byzantine faults and their possibility of leading to Byzantine failures. They pointed out some misunderstanding about Byzantine attack conditions, and proposed countermeasures. Kotla et al. (2010) proposed a speculative BFT protocol, the *Zyzyva*, to simplify the design of BFT state machine replication and to ensure that responses to the correct clients become stable. They compared the *Zyzyva* with existing BFT protocols, including cost, throughput, and latency, and proved that *Zyzyva* can maintain properties of safety and liveness.

BFT is now widely accepted as a basic security necessity, especially for distributed systems with system-level consensus requirements and mutual clock synchronization (Driscoll et al., 2004). Aublin et al. (2013) designed a redundant-BFT (RBFT) approach to closely monitor the performance of

instances from the primary device to replicas on different machines. Bessani et al. (2014) optimized the BFT protocols by applying an open Java-based library source to make state machine replication robust. Li et al. (2014) designed a secure SDN to tolerate Byzantine attacks on the communication links between SDN controllers and switches. Wu et al. (2018a) proposed optimization algorithms to achieve secure cluster management in SDNs. Zhang et al. (2015) focused on the cognitive radio network (CRN) and introduced the Byzantine attack and defense in cooperative spectrum sensing, which is one of the key security issues in a CRN. Miller et al. (2016) argued that the former synchronous BFT protocols critically relied on the network time assumptions and proposed an asynchronous one to extend the adaptability to asynchronous systems, such as blockchain technology.

3 Problem formulation

In this section, we design the system model and formulate the problem of BFT in fog computing.

In contrast to a traditional centralized network design, fog computing prioritizes the local distributed devices at the edge of the network to provide low-latency resource-constrained processing and storage services. In fog, there exist more complex relationships between users and service providers such as fog devices; that is, each user may not stay in contact with the same service provider all the time. Rather than a dedicated wide bandwidth, fog users prefer to flexible dynamic resource allocation, which saves extra time and energy consumption in multi-hop forwarding.

To tolerate the influence of Byzantine faults, we need to set replicas for network nodes as backups to restore and recover data when necessary. In fog, we do not need to prepare dedicated devices for BFT but can assign neighbor fog devices to serve as replicas.

$$n \geq 3f + 1, \quad (1)$$

where f is the number of Byzantine faults, and n is the number of the devices. As shown in Eq. (1), existing BFT protocols, including PBFT (Castro and Liskov, 2002), *Zyzyva* (Kotla et al., 2010), and honey badger

BFT (Miller et al., 2016), elaborate that we need at least $3f$ devices as replicas besides the primary device to tolerate f Byzantine faults, while all communications are synchronous or in bounded delays. For example, if the number of Byzantine faults reaches three, we need at least 10 fog nodes to avoid the Byzantine failure.

3.1 System outline

We formulate the mathematical model of a three-tiered heterogeneous IoT fog network (Fig. 1) (Stojmenovic and Wen, 2014; Reznik et al., 2017). We aim to reduce the impact of Byzantine faults on resource allocation in fog computing. Therefore, we consider that this three-tiered model can more intuitively show the relationship between fog nodes (service providers) and users (service receivers) than the models with more tiers.

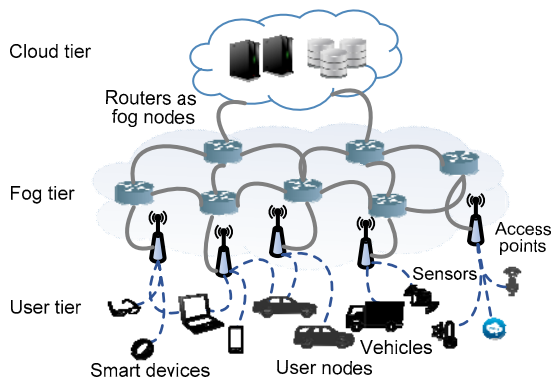


Fig. 1 A three-tiered heterogeneous IoT fog network
Solid and dotted lines stand for the Ethernet and wireless connections, respectively

User nodes (u_1, u_2, \dots, u_n) in the user tier send requests upwards to ask routers to serve as fog nodes (f_1, f_2, \dots, f_n) in the fog tier for computational resources through access points. The cloud tier serves as reliable data centers providing stable network connections. Communications between the user tier and fog tier are wireless broadcasting, those inside the fog tier are wired broadcasting, and those between the fog tier and the cloud tier are wired point-to-point.

Fig. 2 shows the BFT threat model in our work. When users choose some fog nodes as service providers, they also need to accept some permission for authority. The situation is similar to a pop-up window that appears before installation or the first time one

opens an App on a smart device. For example, when a user chooses f_1 to finish a task on a mobile phone, for account certification, the user allows f_1 to use the camera when a service is provided. In normal cases, f_1 will send a message to make the user turn off the camera after certification. However, when f_1 is controlled by someone who wants to obtain additional personal privacy information, the message can be modified and remain open. To guarantee the operation of the fog network, we cannot extensively interrupt service to troubleshoot some individual malicious nodes. It is better to draw support from a suitable fault-tolerant strategy to tolerate possible system failures.

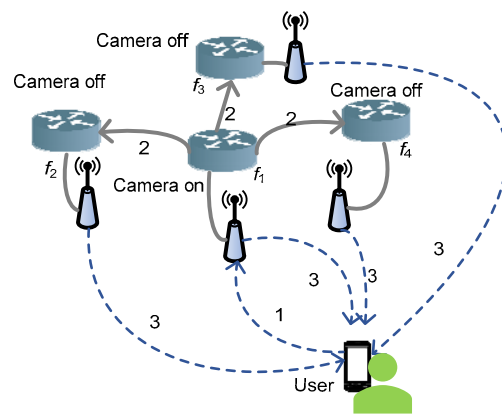


Fig. 2 BFT threat model in the fog service

To implement Byzantine fault tolerance in this three-tiered fog network, we need geo-distributed routers to work as fog nodes to help each other when facing Byzantine faults. We choose f_1 as a service provider and set replicas $f_2, f_3,$ and f_4 to ensure state machine replication when necessary. Take the case of a single Byzantine fault into account, in which three replicas are required by one primary fog node. The entire process of Byzantine-resilient communication in the design of the fog network is as follows:

1. A mobile user in the user tier requests computational resources from the fog tier.
2. A fog node in the fog tier within the suitable distance to the user accepts the request and forwards it to the other three fog nodes as replicas.
3. Both the primary fog node and the replicas execute the task and send back responses to the original user.

Therefore, the original user can check the

responses even if f_1 wants to keep the camera on, and he/she can still tolerate a Byzantine failure from the single fault by checking responses from f_2 to f_4 .

3.2 Performance metrics

To analyze the performance of the proposed BFT resource allocation strategy, we choose the total latency and the number of forwarding hops in the transmission as the two main metrics.

To achieve the BFT and tolerate the Byzantine failures, we operate at the expense of decreasing computing performance in the fog network. That is to say, in the process of multiple fog nodes working together to complete a user request, an additional information exchange is implemented to eliminate the possible impact of the failed nodes. In practice, latency is a basic metric which is widely used for performance evaluation. We use a latency to prove that our methods can achieve a BFT with as little time cost as possible:

$$\begin{aligned} L_{\text{all}} &= L_{\text{trans}} + L_{\text{prop}} + L_{\text{proc}} \\ &= n_{\text{hop}} \sum_i \frac{s_{\text{pkt}}(i)}{r_{\text{bit}}} + \frac{l_{\text{e2e}}}{v_{\text{prop}}} + \sum_i \frac{s_{\text{pkt}}(i)}{r_{\text{MTR}}}. \end{aligned} \quad (2)$$

As shown in Eq. (2), the total end-to-end latency L_{all} includes mainly three parts: transmission latency L_{trans} , propagation latency L_{prop} , and processing procedure latency L_{proc} . L_{trans} represents the time to push all bits of packets into the transmission medium, such as the wires and the air, and L_{trans} is independent of the distance between any two nodes and relates only to the total size of the packets. In contrast, L_{prop} depends on the travel distance between the sender and the receiver, and on the property of the transmission medium. n_{pkt} is the number of packets, and s_{pkt} the size of the packets. r_{bit} is the bandwidth or bit rate of the transmission link. For wireless communication, v_{prop} is equal to the speed of light c ; for wired communication, it ranges from $0.59c$ to $0.77c$. l_{e2e} is the end-to-end length added up by all distances between any two nodes that take part in the current communication. To calculate L_{prop} , we need to know the maximum transfer rate r_{MTR} of the fog devices and the total packet size s_{pkt} :

$$L_{\text{user}}^{\text{all}} = L_1^{\text{trans}} + L_2^{\text{prop}} + L_{2+3}. \quad (3)$$

To obtain the latency in practice (Fig. 2), we need to calculate each part of the three steps, as shown in Eq. (3). For step 1, we sum up L_{trans} and L_{prop} , since there is only one connection between the user and the primary fog node f_1 .

$$L_{2+3} = \max \{L_{2+3}^{f_i} \mid i = 1, 2, 3, 4\}. \quad (4)$$

However, for steps 2 and 3, since the replicas may differ from each other in their positions from the user, for the primary fog node and the processing capacity, we need to figure out the practical latency of the primary device and each replica, and pick the maximum one as shown in Eq. (4). Then, we can obtain

$$L_{2+3}^{f_2} = L_{f_1, f_2}^{\text{trans+prop}} + L_{f_2, \text{user}}^{\text{trans+prop}} + L_{f_2}^{\text{proc}}. \quad (5)$$

Take f_2 as an example. Latencies in steps 2 and 3 include two steps for L_{trans} , L_{prop} , and L_{proc} .

Moreover, we use the number of forwarding hops in the transmission, which can reflect the quantity of work in our fog network and show the practical efficiency.

$$n_{\text{hop}}^{\text{all}} = 2n_{\text{user, pri}}^{\text{hop}} + \sum_{i=2}^4 (n_{f_1, f_i}^{\text{hop}} + n_{f_i, \text{user}}^{\text{hop}}). \quad (6)$$

As shown in Eq. (6), in contrast to the total latency, to calculate the total number of forwarding hops in the transmission, we need to consider all the connections in the three steps in Fig. 2.

Besides the total latency and the number of forwarding hops, to obtain a thorough understanding of the network structure, we use the fog nodes' use rates and the percentages of workload capacity occupied by the primary device or replicas as two auxiliary metrics to analyze the simulation results.

The use rates stand for the overall resource occupancy of all fog nodes. We use the use rate to study the actual working conditions of the entire IoT fog network and the possible changes brought by resource allocation strategies. In the simulation, we consider the cases of both workload capacities occupied as primary service providers and replicas. Moreover, we separately treat the two cases to calculate the percentages.

Table 1 summarizes the main symbols used in this study.

Table 1 Notations used in the design of the Byzantine-resilient fog network

Symbol	Meaning
U and u_i	A set of user nodes and one element in it, respectively
F and f_i	A set of fog nodes and one element in it, respectively
L_{trans} , L_{prop} , and L_{proc}	Transmission latency, propagation latency, and processing latency, respectively
n_{hop}	Number of forwarding hops in the transmission
n_{pkt} and s_{pkt}	The number and size of packets, respectively
r_{bit}	Bit rate of the transmission link
l_{e2e}	End-to-end length of the network connection
v_{prop}	Wave propagation speed of the transmission medium
r_{MTR}	Maximum transfer rate of the device
$C_{i,j}$	The distance or number of forwarding hops between f_i and f_j
P_i	Position coordinates of f_i
$path(P_i, P_j)$	Summation of all connections between f_i and f_j
w_{this}	Needed workload capacity of the current request from the user
C_{pri} and C_{rep}	Workload capacities of the fog nodes occupied as primary and replicas, respectively

4 Byzantine fault-tolerant resource allocation strategy

In this section, we propose resource allocation strategies for a fog network, aimed at tolerating the influence of Byzantine faults.

4.1 BFT fog networking

Before choosing the primary nodes and replicas for users in the need of fog service, we build a BFT fog network considering all neighbor relationships among the routers as fog nodes. Therefore, we aim to fulfill the requirements of the BFT protocol, called ‘‘Zyzyva’’ (Kotla et al., 2010).

Algorithm 1 is based on a non-recursive breadth-first search (BFS) method to implement BFT fast networking. To obtain the connection situation $c_{i,j}$ between any two fog nodes (f_1, f_2, \dots, f_n), including geographical distances and number of forwarding hops in routing, we need a two-dimensional position P_i and neighbor lists recording all adjacent nodes

$f_i.adj$. Two first-in first-out (FIFO) queues Q_{fog} and Q_{save} , variables layer, and leaves are used to build the tree maps formulated using the BFS method. Some key points are as follows:

1. We use Q_{fog} as the main data structure to consider the whole situation. The cyclic condition in step 9 could not be broken after traversing all other nodes, unless no existing path is found between f_i and f_j .

2. Q_{save} is an instrumental variable to save all non-repetitive nodes, which means no path will be tried twice. In step 19, we drop the neighbors of f_{this} , which are already covered by Q_{save} before pushing the rest into two queues.

$$path(P_i, P_j) = \sum_{n=1}^{n_{i,j}-1} length(P_n, P_{n+1}). \quad (7)$$

Algorithm 1 BFT fog networking

Input: $F=\{f_1, f_2, \dots, f_n\}$,
// n fog nodes are in the network structure
 P_i , // coordinates of all the fog nodes
 $f_i.adj$, // the list of all adjacent nodes to f_i
 Q_{fog} and Q_{save} , // FIFO queues to keep fog nodes layer and leaves(layer). // layers and leaves in tree map

Output: $\{C_{i,j}|i, j \in \{1, 2, \dots, n\}, i \neq j\}$.
// connections between any f_i and f_j

```

1  for  $i \leftarrow 2$  to  $n$  do
2  for  $j \leftarrow 1$  to  $i-1$  do
3   $Q_{fog} \leftarrow \emptyset$  and  $Q_{save} \leftarrow \emptyset$ ;
4  if find( $f_i.adj=j$ ) then
5   $c_{i,j} \cdot n_{hop} \leftarrow 1$ ,  $c_{i,j} \cdot l_{e2e} \leftarrow path(P_i, P_j)$ ;
6  end if
7  push all  $f_i.adj$  into  $Q_{fog}$  and  $Q_{save}$ ;
8  layer  $\leftarrow 1$  and leaves(layer)  $\leftarrow$  size of  $f_i.adj$ ;
9  while  $Q_{fog} \neq \emptyset$  do
10  if leaves(layer)  $= 0$  then
11  layer  $\leftarrow$  layer+1;
12  end if
13  this  $\leftarrow Q_{fog}.pop()$ ;
14  leaves(layer)  $\leftarrow$  leaves(layer)-1;
15  if find( $f_{this}.adj=j$ ) then
16   $c_{i,j} \cdot n_{hop} \leftarrow$  layer+1 and  $c_{i,j} \cdot l_{e2e} \leftarrow path(P_i, P_j)$ ;
17  break;
18  end if
19  drop any  $f_{this}.adj$  in  $Q_{save}$  and push them into
    $Q_{fog}$  and  $Q_{save}$  one by one;
20  leaves(layer)  $\leftarrow$  leaves(layer)+size of  $f_{this}.adj$ ;
21  end while
22  end for
23  end for
```

3. Path(P_i, P_j) in steps 5 and 16 are the summation of all connections between any two of the nodes in the full path from f_i to f_j . We can obtain $\text{path}(\cdot, \cdot)$ in Eq. (7), and $n_{i,j}$ is the number of nodes in the path between f_i and f_j .

4. A tree map is obtained using the BFS method, and we use the layer and leaves(layer) to record the current layer and how many nodes are within this layer.

To set the primary fog node and replicas for the user in a request, we need to ensure protocol communications between any two different nodes; that is to say, although our fog network is not a real full connected network, we can still make sure that f_i can exchange messages with f_j at any time after a limited number of forwarding hops. The time complexity of Algorithm 1 is $O(n_2(1+n))=O(n_3+n_2)=O(n_3)$.

4.2 BFT resource allocation strategy

To tolerate Byzantine faults in our fog network, we set the nearby fog nodes as replicas to achieve state machine replication. Thus, each replica needs to repeat what the primary fog node is doing and send back the processing result to the user.

One phase minimum distance (OPMD) gives an entire procedure to set one primary fog node and three replicas for workload w_{this} requested by the current user. C_{pri} and C_{rep} are the resource allocation results, where a part of the workload capacity is set as the primary fog node or replicas. We sort all the fog nodes by their distances away from the position of the current user, and judge whether the remaining workload capacity C_{left} is full, and whether fog node is being requested twice. Some key points are as follows:

1. We use f_{need} as a set of temporary choices of fog nodes during the $3f+1$ cycles, and regard the first choice as the primary fog node.

2. $\text{!find}(f_{\text{need}}(1 \text{ to } i-1)=f_{\text{need}}(i))$ in step 3 is aimed to make sure that the currently chosen $f_{\text{need}}(i)$ is not included in the former one.

OPMD focuses on shortening the communication distances between users and fog nodes, which may extensively cut down on the L_{prop} in Eq. (2). Algorithm 2 makes full use of the advantages of fast networking in the BFS method and can find all required $3f+1$ fog nodes in a simple and straightforward

way. The time complexity of Algorithm 2 is $O((3f+1)(n+1))=O(3f_n+3f+n+1)=O(f_n)$.

However, OPMD may also place an extra burden on communications among the primary fog nodes and replicas providing service for the same users to some extent. Therefore, we put forward a two-phase algorithm to optimize this issue among the primary fog nodes and replicas.

Algorithm 2 One phase minimum distance

Input: w_{this} , // workload needed by the user

C_{pri} and C_{rep} , // capacities of fog nodes used as primary devices and replicas, respectively

$c_{\text{this},j}$, // connections between the user and fog node f_j

f_{need} , // $3f+1$ fog nodes as primary devices and replicas

C_{left} , // resource capacity of fog nodes

Output: resource allocation results for all users

```

1  for  $i \leftarrow 2$  to  $3f+1$  do
2  find  $f_j$  with minimum  $c_{\text{this},j} \cdot l_{e2e}$  and set it as  $f_{\text{need}}(i)$ 
3  if !find( $f_{\text{need}}(1 \text{ to } i-1)=f_{\text{need}}(i)$ ) &&
    $f_{\text{need}}(i) \cdot C_{\text{left}} \geq w_{\text{this}}$  then
4  if  $i=1$  then
5   $f_{\text{need}} \cdot C_{\text{pri}} \leftarrow f_{\text{need}} \cdot C_{\text{pri}} + w_{\text{this}}$ ;
6  else
7   $f_{\text{need}} \cdot C_{\text{rep}} \leftarrow f_{\text{need}} \cdot C_{\text{rep}} + w_{\text{this}}$ ;
8  end if
9   $f_{\text{need}} \cdot C_{\text{left}} \leftarrow f_{\text{need}} \cdot C_{\text{left}} - w_{\text{this}}$ ;
10 else
11 continue;
12 end if
13 end for

```

Compared with OPMD, two-phase shortest path (TPSP) adopts a two-phase design which uses the optimal fog node as the primary one, and lets the primary one look for its $3f$ replicas. Thus, after using one of the fog nodes as f_j , subsequent sorting and other work will be carried out around it, instead of the current user who requests w_{this} . Some key points are as follows:

1. The sum of f_{pri} and f_{rep} is equal to f_{need} in TPSP.

2. Step 5 shows two selections when choosing suitable neighbor fog nodes as replicas, showing different performances, such as the majority of total latency as shown in Eq. (2). L_{trans} pays more attention to the number of forwarding hops, and L_{prop} relies on the transmission distance.

3. The time complexity of Algorithm 3 is $O(1+3f(n+1))=O(3f_n+3f+1)=O(f_n)$.

Algorithm 3 Two-phase shortest path

Input: w_{this} , // workload needed by the user
 C_{pri} and C_{rep} , // capacities of fog nodes used as the
// primary device and replicas, respectively
 f_{pri} and f_{rep} , // fog nodes set as the primary device and
// replicas, respectively
 $c_{\text{this},j}$, // connections between the user and fog node f_j
 f_{needs} , // $3f+1$ fog nodes as primary device and replicas
 C_{left} , // resource capacity of fog nodes
Output: resource allocation results for all users.

- 1 find f_j with minimum $c_{\text{this},j} \cdot l_{e2e}$ && $f_j \cdot C_{\text{left}} \geq w_{\text{this}}$ and set it as $f_{\text{pri}}(i)$;
- 2 $f_{\text{pri}} \cdot C_{\text{pri}} \leftarrow f_{\text{pri}} \cdot C_{\text{pri}} + w_{\text{this}}$;
- 3 $f_{\text{pri}} \cdot C_{\text{left}} \leftarrow f_{\text{pri}} \cdot C_{\text{left}} - w_{\text{this}}$;
- 4 **for** $i \leftarrow 1$ to $3f$ **do**
- 5 find f_j with least $c_{\text{pri},j} \cdot n_{\text{hop}}$ or minimum $c_{\text{pri},j} \cdot l_{e2e}$ and set it as $f_{\text{rep}}(i)$;
- 6 **if** $f_{\text{rep}} \neq f_{\text{pri}}$ && $! \text{find}(f_{\text{rep}}(1 \text{ to } i-1) = f_{\text{rep}}(i))$ && $f_{\text{rep}}(i) \cdot C_{\text{left}} \geq w_{\text{this}}$ **then**
- 7 $f_{\text{rep}} \cdot C_{\text{rep}} \leftarrow f_{\text{rep}} \cdot C_{\text{rep}} + w_{\text{this}}$;
- 8 $f_{\text{rep}} \cdot C_{\text{left}} \leftarrow f_{\text{rep}} \cdot C_{\text{left}} - w_{\text{this}}$;
- 9 **else**
- 10 **continue**;
- 11 **end if**
- 12 **end for**

5 Simulations and analysis

In this section, we carry out simulations to evaluate the performance of the resource allocation strategies designed for a BFT fog network in two cases: with a single Byzantine fault and with multiple Byzantine faults. The simulation scenario is a 10 km² square open area in which we set up 100 routers with access points as fog nodes. There are 50–500 mobile IoT users requesting for fog service from nearby fog nodes.

As shown in Table 2, we consider the conditions of both wireless and Ethernet connections with their respective transmission bit rate and wave propagation speed. The workload capacity of a single fog node would be 32, 64, 128, 256, or 512 MB, according to a single Byzantine fault or multiple faults. We use multiple time slots to collect requests, and then process them and store the results. We repeat each set of simulations 10 times with different numbers of IoT users.

5.1 A single Byzantine fault

We consider first the case of a single Byzantine fault ($f=1$), which means that we aim to tolerate the

influence of a single fault in the procedure of answering a request from an IoT user. Therefore, we need to choose four fog nodes for one user in each time slot as the primary device and replicas. The workload capacity range set of the fog nodes is {32, 64, 128, 256}.

Table 2 Experimental setup

Parameter	Value
Bit rate of transmission	
Wireless (802.11ad)	6.8 Gb/s
Ethernet	10 Gb/s
Maximum transfer unit (802.11)	2304 bytes
Wave propagation speed of transmission	
Wireless (air)	c (speed of light)
Ethernet (thick coax)	$0.77c$
Maximum transfer rate (SATA3)	750 MB/s
Workload capacity of the fog node	32–512 MB

As shown in Fig. 3, we calculate the total latency and the number of forwarding hops in the transmission for different numbers of users. In Fig. 3a, all the four methods show a linear increase from 50 to 500 IoT users requesting for fog services from 100 routers. Although the differences among the four methods are not large when there are only a few users, the gap between the random and the other three methods appears with the growth of the number of users. The performance of OPMD is relatively poor compared with the TPSP-dist and TPSP-hop methods, which matches our expectation. Compared with TPSP which applies two phases in fog node selections, the same treatments for the primary fog device and replicas do generate some impacts on the total latency. That is, the position of the primary fog node is more than an issue not only when receiving the request but also when distributing it to all replicas. Therefore, the overlong distance or redundant forwarding hops between the primary fog node and replicas may cost extra time in data transmission.

The number of total forwarding hops is the second metric that we use to compare and analyze the simulation results of BFT resource allocation in the three-tiered heterogeneous IoT fog network. In Fig. 3b, we can see that TPSP-hop still holds the lead in terms of practical efficiency, whereby more transmission hops mean extra energy consumption in the transmissions between the IoT users and the fog nodes. In particular, when there are a large number of

users, TPSP-hop behaves better in dealing with situations where demand exceeds supply. Thus, service capacities could be insufficient relative to the user's needs, and sometimes the user has to choose a service node with a relatively high cost in terms of time and energy consumption.

5.2 Multiple Byzantine faults

Because it is not sure whether only one Byzantine fault would occur in the BFT communication procedure (Fig. 2), we should take multiple Byzantine faults into account. In this simulation, f relates to the size of the requested workload capacity, which means that the possibility of multiple Byzantine faults is proportional to the number of resources allocated to users. To fulfill the urgent need of available resources, we adjusted the workload capacity range set of the number of fog nodes to $\{64, 128, 256, 512\}$.

Compared with the case of a single Byzantine fault in Fig. 3a, the case of multiple Byzantine faults

in Fig. 4a does not show considerable performance degradation in the total latency. For the numbers of forwarding hops in Figs. 3b and 4b, as more replicas are set to ensure the state machine replication when necessary, the numbers of total transmission hops in the three methods increase by more than 50%. Therefore, our approach is not limited to a single fault. It can maintain performance when the multiple ones occur. Another point is that TPSP-dist fluctuates and loses the advantage over OPMD when number of users exceeds 450. The reason may be that, as the relationship among fog nodes becomes more complex, making a decision in the second phase of TPSP-dist fails to find a better choice of suitable replicas.

5.3 Device use rates and percentages of the primary devices and replicas

To figure out the composition and the actual working conditions of the IoT fog network, we add the fog nodes' use rates as well as the percentages of

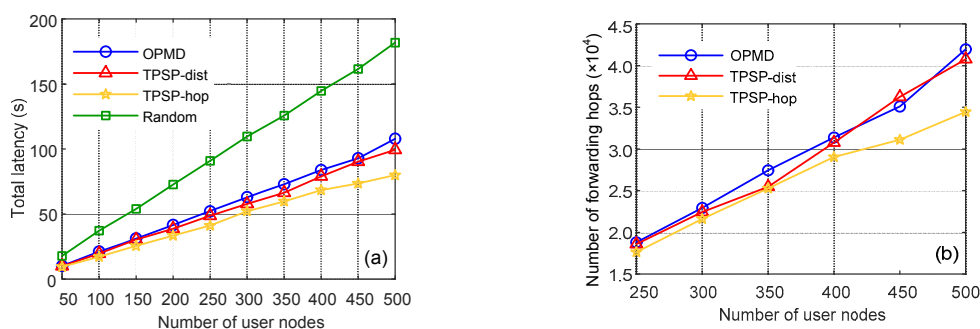


Fig. 3 Simulation results in the case of a single Byzantine fault: (a) total latency; (b) number of forwarding hops in transmission

The red and yellow lines represent the simulation results of different standards when choosing suitable neighbor fog nodes, as shown in step 5 of Algorithm 3. The yellow line considering the number of forwarding hops shows less total latency than the red one, which illustrates that the time cost of L_{trans} takes up a larger proportion than that of L_{prop} . References to color refer to the online version of this figure

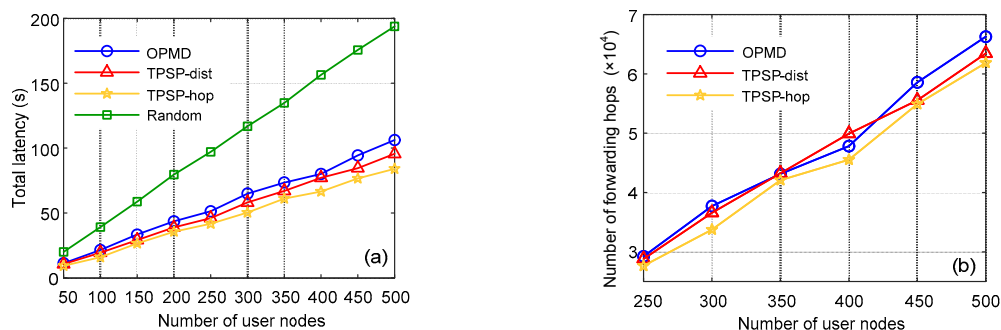


Fig. 4 Simulation results in the case of multiple Byzantine faults: (a) total latency; (b) number of total forwarding hops in transmission

References to color refer to the online version of this figure

workload capacity occupied by the primary devices or replicas as auxiliary metrics to provide more details.

Figs. 5 and 6 show the use rates and primary devices and replicas percentages of three resource allocation methods in cases of a single fault and multiple faults. First, in the comparisons between the two cases of the same method, the occupied workload proportions of the number of replicas all increase when more replicas are needed, and a single fog node is set as a replica in multiple requests. Second,

the use rates of TPSP-hop are always lower than those of the other two methods ranging from 5% to 10% (Fig. 5), which can be the superiority in terms of efficiency. That is to say, TPSP-hop can complete the same amount of work using fewer computational resources. Third, compared to the second point above, in Fig. 6, the gap between TPSP-hop and other two methods in terms of device use rates is narrowed when more than one Byzantine fault occurs in a single BFT communication procedure.

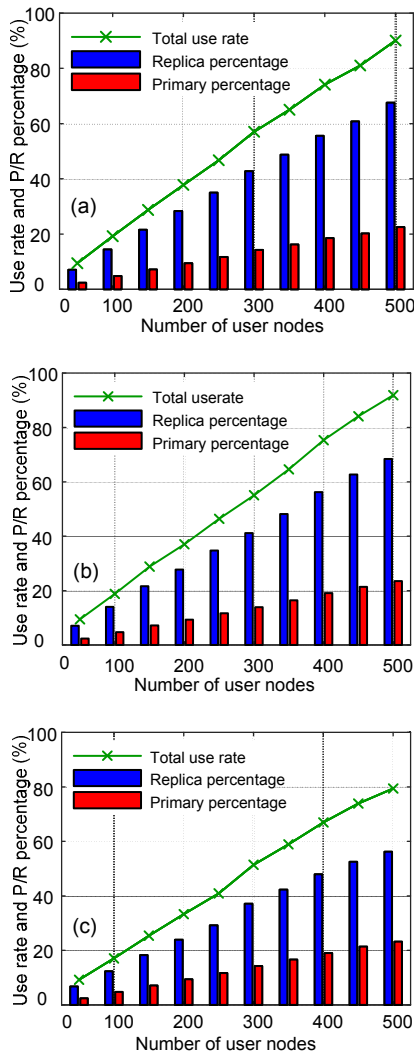


Fig. 5 Device use rates and percentage of the primary devices and replicas in the case of a single Byzantine fault: (a) OPMD; (b) TPSP-dist; (c) TPSP-hop
 Green broken line stands for the actual occupancy rates, which are the average of 10 time slots. The blue and red bars are the average percentages of workload capacity occupied by the replicas and primary devices, respectively. References to color refer to the online version of this figure

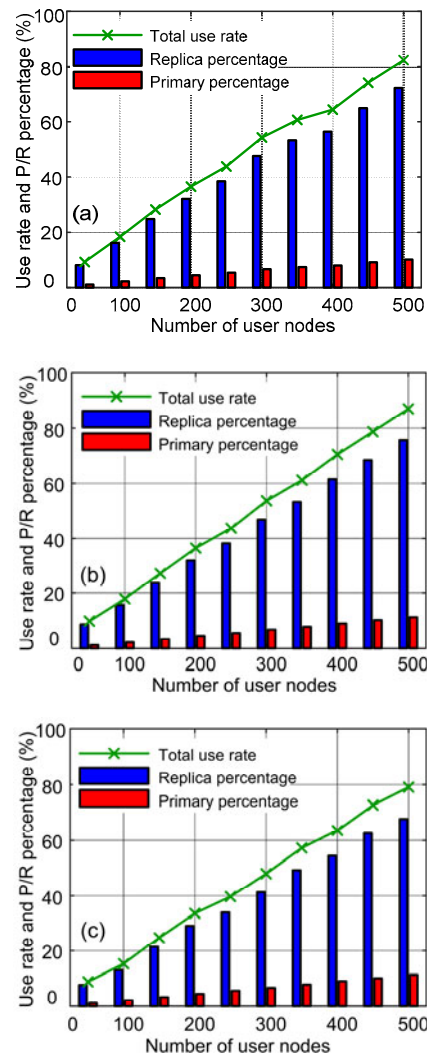


Fig. 6 Device use rates and percentage of the primary devices and replicas in the case of multiple Byzantine faults: (a) OPMD; (b) TPSP-dist; (c) TPSP-hop
 The green broken line stands for the actual occupancy rates, which are the average of 10 time slots. The blue and red bars are the average percentages of workload capacity occupied by the replicas and primary devices, respectively. References to color refer to the online version of this figure

In summary, from the simulations of the cases of a single Byzantine fault and multiple faults, TPSP with the selection standard of fewer transmission hops shows better performance in terms of total latency, number of forwarding hops, and device use rates. As a result, the BFT resource allocation strategy builds a reliable fog network structure to tolerate the influence of a single Byzantine fault or multiple faults.

6 Conclusions

In this paper, we aim to tolerate the influence of Byzantine faults and improve the transmission and processing efficiency in SIoT Fog. We have designed a three-tiered heterogeneous IoT fog network model which consists of routers as fog nodes to provide fog service to IoT users. To solve the problem of BFT in fog services, we have proposed a fog networking method based on breath-first search and two BFT resource allocation strategies to distribute workload capacities of the fog nodes to users upon request. We consider both a single Byzantine fault and multiple faults in simulations. Simulation results show that our proposed strategies can build an efficient and reliable fog network when faced with Byzantine faults.

In the future, we will focus on further improving our approach to deal with the various situations that may occur in actual network operations. There are two performance boundaries in our proposed strategies: (1) To ensure BFT in fog computing, we rely on the mutual assistance of the geo-graphically distributed fog nodes, which means that there may be significantly different performances for different node distributions; (2) On a distributed network composed of large-scale fog nodes, the fact that BFT does increase the relationships among the nodes may lead to new issues when the network topology changes.

References

- Alrawais A, Alhothaily A, Hu CQ, et al., 2017. Fog computing for the Internet of Things: security and privacy issues. *IEEE Internet Comput*, 21(2):34-42. <https://doi.org/10.1109/MIC.2017.37>
- Aublin PL, Mokhtar SB, Quéma V, 2013. RBFT: redundant Byzantine fault tolerance. *IEEE 33rd Int Conf on Distributed Computing Systems*, p.297-306. <https://doi.org/10.1109/ICDCS.2013.53>
- Bessani A, Sousa J, Alchieri EEP, 2014. State machine replication for the masses with BFT-SMART. 44th Annual IEEE/IFIP Int Conf on Dependable Systems and Networks, p.355-362. <https://doi.org/10.1109/DSN.2014.43>
- Bonomi F, Milito R, Zhu J, et al., 2012. Fog computing and its role in the Internet of Things. *Proc 1st Edition of the MCC Workshop on Mobile Cloud Computing*, p.13-16. <https://doi.org/10.1145/2342509.2342513>
- Castillo-Cara M, Huaranga-Junco E, Quispe-Montesinos M, et al., 2018. FROG: a robust and green wireless sensor node for fog computing platforms. *J Sens*, 2018:3406858. <https://doi.org/10.1155/2018/3406858>
- Castro M, Liskov B, 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans Comput Syst*, 20(4): 398-461. <https://doi.org/10.1145/571637.571640>
- Driscoll K, Hall B, Sivencrona H, et al., 2003. Byzantine fault tolerance, from theory to reality. *LNCS*, 2788:235-248. https://doi.org/10.1007/978-3-540-39878-3_19
- Driscoll K, Hall B, Paulitsch M, et al., 2004. The real Byzantine generals. 23rd Digital Avionics Systems Conf, p.1-11. <https://doi.org/10.1109/DASC.2004.1390734>
- Gao DH, Wang QF, Lei Y, 2017. Distributed fault-tolerant strategy for electric swing system of hybrid excavators under communication errors. *Front Inform Technol Electron Eng*, 18(7):941-954. <https://doi.org/10.1631/FITEE.1601021>
- Hu PF, Ning HS, Qiu T, et al., 2017. Security and privacy preservation scheme of face identification and resolution framework using fog computing in Internet of Things. *IEEE Internet Things J*, 4(5):1143-1155. <https://doi.org/10.1109/JIOT.2017.2659783>
- IHS Markit, 2017. IoT Trend Watch 2017. <https://ihsmarkit.com/Info/0117/IoT-trend-watch-2017.html> [Accessed on Aug. 29, 2018].
- Jalali F, Hinton K, Ayre R, et al., 2016. Fog computing may help to save energy in cloud computing. *IEEE J Sel Areas Commun*, 34(5):1728-1739. <https://doi.org/10.1109/JSAC.2016.2545559>
- Khosravi A, Kavian YS, 2016. Autonomous fault-diagnosis and decision-making algorithm for determining faulty nodes in distributed wireless networks. *Front Inform Technol Electron Eng*, 17(9):885-896. <https://doi.org/10.1631/FITEE.1500176>
- Kotla R, Alvisi L, Dahlin M, et al., 2010. Zyzzyva: speculative Byzantine fault tolerance. *ACM Trans Comput Syst*, 27(4), Article 7. <https://doi.org/10.1145/1658357.1658358>
- Lamport L, Shostak R, Pease M, 1982. The Byzantine generals problem. *ACM Trans Program Lang Syst*, 4(3):382-401. <https://doi.org/10.1145/357172.357176>
- Li H, Li P, Guo S, et al., 2014. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *IEEE Trans Cloud Comput*, 2(4):436-447. <https://doi.org/10.1109/TCC.2014.2355227>
- Li H, Ota K, Dong MX, 2018. Learning IoT in edge: deep learning for the Internet of Things with edge computing. *IEEE Network*, 32(1):96-101.

- <https://doi.org/10.1109/MNET.2018.1700202>
- Liu Z, Dong MX, Zhou H, et al., 2016. Device-to-device assisted video frame recovery for picocell edge users in heterogeneous networks. *IEEE Int Conf on Communications*, p.1-6. <https://doi.org/10.1109/ICC.2016.7511342>
- Miller A, Xia Y, Croman K, et al., 2016. The honey badger of BFT protocols. *Proc ACM SIGSAC Conf on Computer and Communications Security*, p.31-42. <https://doi.org/10.1145/2976749.2978399>
- Perera C, Qin YR, Estrella JC, et al., 2017. Fog computing for sustainable smart cities: a survey. *ACM Comput Surv*, 50(3), Article 32. <https://doi.org/10.1145/3057266>
- Reznik A, Arora R, Cannon M, et al., 2017. Developing software for multi-access edge computing. ETSI White Paper 20.
- Satyanaarayanan M, 2017. The emergence of edge computing. *Computer*, 50(1):30-39. <https://doi.org/10.1109/MC.2017.9>
- Satyanaarayanan M, Bahl P, Cáceres R, et al., 2009. The case for VM-based cloudlets in mobile computing. *IEEE Perv Comput*, 8(4):14-23. <https://doi.org/10.1109/MPRV.2009.82>
- Stojmunic I, Wen S, 2014. The fog computing paradigm: scenarios and security issues. *Proc Federated Conf on Computer Science and Information Systems*, p.1-8. <https://doi.org/10.15439/2014F503>
- Tao M, Ota K, Dong M, 2017. Foud: integrating fog and cloud for 5G-enabled V2G networks. *IEEE Network*, 31(2): 8-13. <https://doi.org/10.1109/MNET.2017.1600213NM>
- Tao XY, Ota K, Dong MX, et al., 2017. Performance guaranteed computation offloading for mobile-edge cloud computing. *IEEE Wirel Commun Lett*, 6(6):774-777. <https://doi.org/10.1109/LWC.2017.2740927>
- Vaquero LM, Rodero-Merino L, 2014. Finding your way in the fog: towards a comprehensive definition of fog computing. *ACM SIGCOMM Comput Commun Rev*, 44(5): 27-32. <https://doi.org/10.1145/2677046.2677052>
- Wu J, Dong MX, Ota K, et al., 2018a. Big data analysis-based secure cluster management for optimized control plane in software-defined networks. *IEEE Trans Network Serv Manag*, 15(1):27-38. <https://doi.org/10.1109/TNSM.2018.2799000>
- Wu J, Dong MX, Ota K, et al., 2018b. FCSS: fog computing based content-aware filtering for security services in information centric social networks. *IEEE Trans Emerg Top Comput*, in press. <https://doi.org/10.1109/TETC.2017.2747158>
- Yi SH, Li C, Li Q, 2015. A survey of fog computing: concepts, applications and issues. *Proc Workshop on Mobile Big Data*, p.37-42. <https://doi.org/10.1145/2757384.2757397>
- Zeng DZ, Gu L, Yao H, 2018. Towards energy efficient service composition in green energy powered cyber-physical fog systems. *Fut Gener Comput Syst*, in press. <https://doi.org/10.1016/j.future.2018.01.060>
- Zhang LY, Ding GR, Wu QH, et al., 2015. Byzantine attack and defense in cognitive radio networks: a survey. *IEEE Commun Surv Tutor*, 17(3):1342-1363. <https://doi.org/10.1109/COMST.2015.2422735>
- Zhang WZ, Lu K, Wang XP, 2018. Versionized process based on non-volatile random-access memory for fine-grained fault tolerance. *Front Inform Technol Electron Eng*, 19(2): 192-205. <https://doi.org/10.1631/FITEE.1601477>