Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 11, November 2019, pp. 3008-3015
DOI: 10.23940/ijpe.19.11.p20.30083015

Survivability of Distributed Fault Detection Systems

Lijun Zhou*, Haiyan Lv, Kai Liu, and Jie Zhang

Naval Aviation University, Yantai, 264001, China

Abstract

In the design of distributed fault detection systems, an important basis is to monitor the entities in the distributed network computing system in order to achieve the full coverage of the fault detection function. Such distributed computing systems often involve many nodes, large geographical span, unstable communication delays, and loose management. It is very difficult to cover such systems functionally. Aiming at this problem, based on the idea of self-organizing networks and the realization of coverage monitoring of system nodes, this paper studies the survivability of monitoring functions caused by highly dynamic nodes and proposes a set of detection and repair methods for the system cut vertexes, which reduces the impact of highly dynamic nodes on system monitoring.

Keywords: survivability; distributed system; fault detection

(Submitted on September 12, 2019; Revised on October 16, 2019; Accepted on November 25, 2019)

1. Introduction

With the continuous development of information technology, software systems have become more large-scale and distributed. In the large-scale distributed environment, the previous fault detection methods have been inadequate. The main reason is that these methods are mostly aimed at centralized computing environments or known fault sources to study fault recognition mechanisms based on a single computing object.

In a large distributed system, the importance of each node is different for system reliability. For example, Saroiu et al. observed that in a Gnutella network with 1,771 nodes, 30% of the nodes were randomly removed, and 1,106 of the remaining 1,300 nodes remained connected, and they were still exchanging and sharing information and data. However, if only 4% of the nodes satisfying certain conditions are selected and removed from the system, the whole system will be scattered into hundreds of independent "information islands". Some scholars also noticed the existence of key nodes and links in P2P networks and sensor networks, but they did not solve this problem well [1-3]. Other researchers proposed solutions to detect and restore key nodes and links in P2P and AdHoc networks based on flood message dissemination [4-5].

2. Design of Distributed Fault Detection System

2.1. Survivability of Distributed Fault Detection System

The survivability of distributed fault detection systems means that under the premise that the system components are unreliable and the overall reliability of the system is required, the system can still guarantee the integrity of the remaining components to be able to detect faults when the system is faced with random failure of components. At the same time, the reliability of the detection function can be restored in a certain time.

In the distributed fault detection system designed in this paper, detectors are distributed everywhere in the network system, and a self-organizing detection network is formed among them. The detection function covers the whole distributed system. Figure 1(a) is part of a detection coverage network. The dotted arrow in the figure indicates the detection of the message propagation flow. Obviously, from the global point of view, the failure of any white node in the graph will not

* Corresponding author.
E-mail address: jungle730@163.com

cause the system to be split into many parts, and the detection message can still reach all the nodes in the graph. However, if the grey node P fails, the original connected detection coverage network will be divided into two parts. Here, node P is called the key node, or cut vertex.

Figure 1. Illustration of a cut vertex in a detection overlay network

2.2. Framework of Distributed Fault Detection System

Distributed fault detection, as an abstraction of network computing entity, runs on a specific network computing service entity, collects the state of the entity providing service, and disseminates the collected state information to other detectors to provide real-time identification of the entity's fault state and fault-tolerant strategy support for trusted application services. Between the entity of the business and the decision-making of the trusted application, it is a basic service. It is shown in Figure 2.

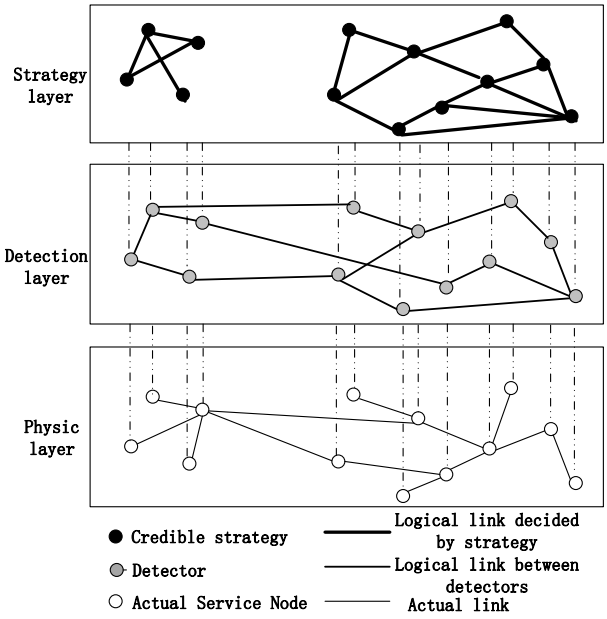


Figure 2. Layers of the distributed fault detection system

As can be seen from Figure 2, the logical relationships among computing service entities, detectors, and trustworthy application decisions are independent of each other. Detectors establish a detection network covering all network entities by communicating with each other. The network is independent of the entity network and is an application layer overlay network. Detectors can provide state information of underlying entities for upper applications, provide policy support for improving the credibility of application services, and shield other irrelevant details of network entities.

In order to achieve this goal, a detector is attached to each detected object in the fault detection system. The detector maintains that the state information of the detected object is collected and preprocessed first. Each detector maintains a monitoring list for establishing communication between detectors and sending it to other nodes in the monitoring list. The collected state data, or the entity state information sent by other detectors, can be received and forwarded to obtain the entity state information of the whole network. By identifying the state of the detected object, the credible state of the detected object is provided to the upper application program, so that it can follow a certain strategy to deal with the fault, as shown in Figure 3.

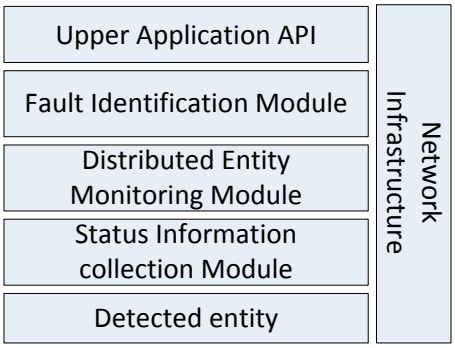


Figure 3. Architecture of fault detector

3. Entity Monitoring Algorithm for Distributed Systems

Self-organizing network is a virtual network topology composed of nodes and logical links based on existing physical networks. The method of constructing a self-organizing detection domain is based on the distance of the detector. The system can form a monitoring domain covering all nodes of the system by selectively executing the algorithms below.

3.1. Key Node Detection and Repair Algorithms

Algorithm 1: Distance-based neighborhood construction for self-organizing monitoring

Define the *t* distance: In a fully connected network, the time delay of sending back messages from node N_i to node N_j is *t*, and *t* is called the *t* distance between node N_i and node N_j. Suppose that the size of the self-organizing monitoring neighborhood is $K = k + l$, where the initial threshold of each monitoring neighborhood is *k*, but there can be a monitoring neighborhood elastic space value l, which is mainly to accommodate the newly added nodes in the redundant space. The algorithm steps are as follows:

(1) The broadcasting neighborhood construction request information is sent to all nodes in the system by the selected proxy node.

(2) Each surviving node without any monitoring neighborhood responds to the request information, and the proxy node retrieves the response message and sorts the response messages according to the time.

(3) If the number of response messages received is greater than *k*, proceed to step 4; if not, proceed to step 6.

(4) According to the time of arrival of the response message, the first *k* nodes are selected as a monitoring neighborhood centered on the proxy node. The upper limit of the r distance of the neighborhood is two times the delay of the latest response node. At the same time, the proxy node sends a neighborhood construction confirmation message containing the upper limit of the *t* distance of the neighborhood to the *k* nodes to complete a monitoring neighborhood construction.

(5) The proxy node sends the list of the remaining nodes that are not joined in the monitoring neighborhood to a node with a longer response delay. The selection principle is that the node whose neighborhood *t* distance is not more than two times the upper limit is selected, and it is requested to act as the next monitoring neighborhood proxy node. The selected node is transferred to step 1 and the local proxy node to step 6.

(6) The proxy node sends a neighborhood construction confirmation message to all the responding nodes. The upper limit of t-distance of the neighborhood is two times the r-distance of the latest response message, thus completing a monitoring neighborhood construction. The algorithm is complete.

When the system forms a monitoring domain covering all nodes of the system, the nodes in a single neighborhood implement all-to-all monitoring. They distribute the status information of the detected objects by gossip among each other and disseminate messages through proxy nodes between domains to achieve the purpose of message coverage of the whole system.

However, in large-scale distributed environment, nodes are highly dynamic; in other words, a monitoring domain is

dynamic, and nodes exit and join at any time. The withdrawal of a monitoring domain's proxy node will lead to the failure of a monitoring domain, so it is necessary to design an algorithm to join the monitoring domain to meet the dynamic requirements of the system.

Algorithm 2: Nodes join monitoring neighborhood

It is assumed that h independent monitoring neighborhoods cover the existing system nodes when nodes join.

(1) The node sends a message requesting to join the neighborhood to all *h* proxy nodes, and the proxy node will reply to the response message immediately after receiving the request message. The response message includes the maximum size *K* of the domain, the current size *k* of the domain, and the upper limit *T* of the distance of the domain t.

(2) The nodes put the received response messages into the list and sort them according to the response time. At this time, the *t* distance between them and all the proxy nodes is obtained.

(3) If the list is not empty, the node chooses the proxy node message with the current minimum *t* distance and proceeds to step 4; if the list is empty, it proceeds to step 7.

(4) Determine whether the t distance between the node and proxy node is less than *T*/2. If not, delete the reply message from the list and proceed to step 3. If so, proceed to step 5.

(5) Determine whether $k \geqslant K$. If so, delete the message from the list and proceed to step 3; if not, proceed to step 6.

(6) The node sends the affirmation message to the proxy node, which contains the current t-distance information between them. When the proxy node receives the affirmation message, it returns to the node a local view of the current proxy node. At the same time, it disseminates the information of the node in the domain to notify other nodes in the domain. The other nodes in the domain will selectively add the node. The algorithm is complete.

(7) The node itself becomes a proxy node in the spatial domain and sends the proxy node confirmation information to other proxy nodes. The t-distance E of the spatial domain is infinite. The algorithm is complete.

Algorithm 3: Node exit monitoring neighborhood

(1) The node decides whether it is a proxy node or not. If not, it proceeds to step 2, and if so, proceeds to step 4.

(2) The node first sends a cancellation request message to the proxy node in its humming domain and then completes the exit monitoring neighborhood.

(3) The proxy node first checks whether the node is in its local view after receiving the cancellation request message of the node. If so, it deletes the node. At the same time, it publishes a node cancellation rumor in the monitoring neighborhood. When the node in the monitoring domain receives the message, it will detect its local view. If the cancellation node is in its own view, it will delete it. If not, the message continues to be disseminated until the TTL value of the message changes to 0. The algorithm is complete.

(4) The proxy node first chooses a node with the smallest t distance in the domain and sends a message requesting it to become a proxy node. If the node generates a response, the following steps are performed:

(5) The cancellation agent node replies the confirmation message of the new agent node, which contains the domain view and the agent node view of the cancellation agent node, and the new agent node replaces its local view with this message.

(6) The cancellation agent node will send replacement messages to other agent nodes and replace their own message with the message of new agent nodes.

(7) The cancellation agent node will issue a logout rumor to the node in the domain and specify a new agent node. When the node in the domain receives the message, it will detect its local view. If the logout node is in its own view, it will be deleted and replaced by the new agent node. If not, it will continue to spread the message until the TTL value of the

message becomes zero. The algorithm is complete.

By selectively executing Algorithms 1 to 3, the system can form a monitoring domain covering all nodes of the system.

3.2. Key Node Detection and Repair Algorithms

In a distributed fault detection system, detection messages are usually propagated by hop-by-hop random dissemination mechanism after a node is generated [6-12]. In the process of message dissemination, each forwarding of a node adds its own information to the message; in other words, the message contains the path information from the starting point of the message to the current node. The design of this project makes full use of this information to determine the cut vertex by dividing the neighbors of a node into one or more blocks. When a node's neighbors belong to multiple blocks, it is a cut vertex.

3.2.1. Cut Vertex Adaptive Detection Algorithm

In Figure 4, node P is a cut vertex connecting three blocks: vertex P, and vertex R, and the bridge between them that forms a block. Removing vertex P will cause the graph to be divided into three connected segments, which are marked with dotted ellipses. In cut vertex adaptive detection, each node maintains an MsgLst list for storing the recently received message records. Each message is recorded in the following manner:

<Message ID, Node IDs of the Neighbors which the message has traversed>

Each node also maintains a block status information BlockSet data structure for storing its block information and neighborhood information. In the initial stage of cut vertex adaptive detection, MsgLst is empty and assumes that all neighbors of the node are in different blocks.

After a period of execution of adaptive detection, node P will receive messages from three parts. For example, node A generates message 1 and broadcasts it to its neighbors. When nodes B and C receive message 1 through disjoint paths and forward it to node P, node P will find that there are two disjoint paths in the process of message 1 dissemination. Although node P does not know which node produces message 1, it can determine a closed-loop path containing nodes A, B, and C and its own (A, B, P, C), so that nodes B and C are two-connected and belong to a block. Node P merges the blocks containing B and C into one block. Similarly, after receiving message 2 from nodes C and E, P puts B, C, and E into the same block. Nodes X and Y are merged into a block after P receives message 3. Node R remains isolated in its block because no other node in the block can forward messages 4 to P. Figure 5 shows a flow chart of the adaptive cut vertex detection algorithm.

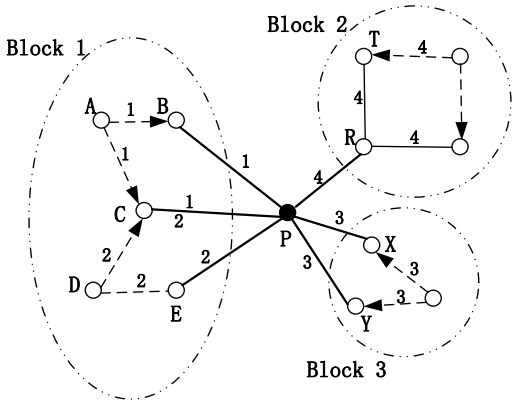


Figure 4. Example of CV adaptive detection

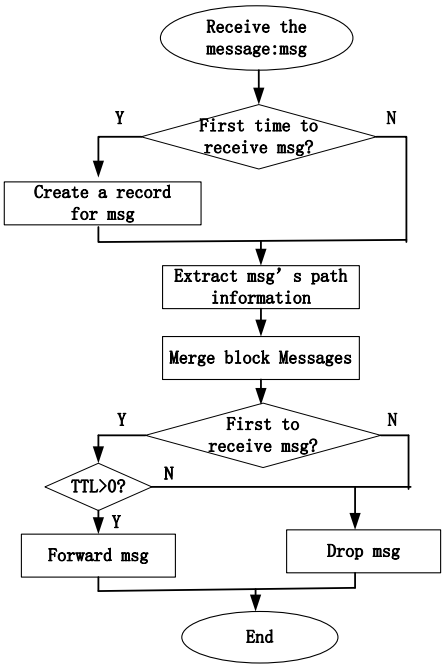


Figure 5. Flow chart of adaptive CV detection algorithm

3.2.2. Active Detection of Cut Vertex

According to adaptive detection, when all neighbor nodes of a node are in a block, it can be decided that it is not a cut vertex. If a node's BlockSet contains two or more blocks, it does not necessarily mean that the node must be a cut vertex. In the protocol, nodes disseminate messages randomly, messages have TTL thresholds, and messages may be discarded in the process of dissemination. This does not guarantee that every message can reach a node, so active detection is needed to further determine whether a node is a cut vertex. Compared with adaptive detection, active detection can achieve shorter convergence time on the premise of consuming a certain amount of network load. If the neighbor of a node is still divided into two or more blocks in the final stage of adaptive detection, the node will regard itself as a suspected cut vertex node and initiate the active detection process.

Firstly, it randomly selects a node from each block, that is, the node labeled with different block values in its LocalView and numbers in each neighbor connection. The numbering can also directly use the block values of the nodes, and then the nodes send detection information to these connections in turn. The format of the detection information is as follows:

<ID, timestamp, TTL, connection-index>

Among them, ID is the ID number of the node itself, the time stamp records the time when the detected message is generated, TTL (time to live) is a preset value of the maximum number of hops that a message can be forwarded, and connection-index indicates the connection number used by the suspected cut vertex node to send the message. Each node will save and maintain a connection list Connection-List, in which each suspected cut vertex node has a relevant record entry in the form of

<candidate's ID, timestamp, connection-index 1, connection-index 2⋯>

When a node receives a cut vertex active detection message from node N, it assists in running the cut vertex automatic detection algorithm.

3.3. Detection and Elimination of Cut Vertexes

According to the related explanations in Section 2.1, in distributed fault detection systems, two algorithms, adaptive cut vertex detection and active cut vertex detection, are used to determine whether a node is a cut vertex or not.

As the number of messages received by node P increases, it also constantly merges and updates the value of the node's BlockSet in its LocalView. In the final stage of adaptive detection, all neighbor nodes of node P will be merged into several blocks. If there is only one block in the BlockSet, that is, all nodes have the same BlockSet, then node P is not a cut vertex. On the contrary, the active detection process will be triggered for further detection.

According to the detection message received from the cut-point suspect node, a node makes corresponding operations according to the content stored in the local Connection-List. There are four operations.

Through the continuous implementation of adaptive detection and active detection, it is possible to determine whether the node itself is a cut vertex in the global view of the system with a larger probability. Because the large ship model distributed network computing system is a system with many participating nodes, which are widely distributed and highly dynamic nodes, the acquisition of global knowledge tends to converge very slowly. At the same time, when global knowledge is formed, the system topology may have changed. Therefore, the current method of determining a node as a cut vertex in a relatively short period of time can satisfy the system's requirement for survivability.

4. Experimental Verification

The project uses the network monitoring tool developed on the Linux system based on Socket network programming API to simulate D-Gossip message dissemination in a wide area network environment and verify the effectiveness of algorithms designed in this paper. UDP is used to disseminate messages in neighborhoods, and TCP is used to disseminate messages between neighborhoods. The experiment of large-scale nodes adopts the method of mathematical simulation.

In order to accelerate the convergence speed of cut vertex detection and verify the influence of message spread and message TTL on Cut Vertex detection, Algorithm 1 is used to construct the monitoring neighborhood by setting the size of

each monitoring area to 8, 16, and 32. The related settings are shown in Table 1 .

Table 1. Configuration of cut vertex detection and verification

Round	Total number of system nodes	Number of messages disseminated	Number of proxy nodes	*TTL*
1	800	2	100	5
2	1600	4	100	5
3	3200	4	100	10

Firstly, cut vertexes are determined by running Algorithms 1 and 2 in nodes. The experimental results are shown in Figure 6. They indicate that the number of messages disseminated has a greater impact on the detection of key nodes, while a larger TTL does not play a significant role in the latter half of the detection.

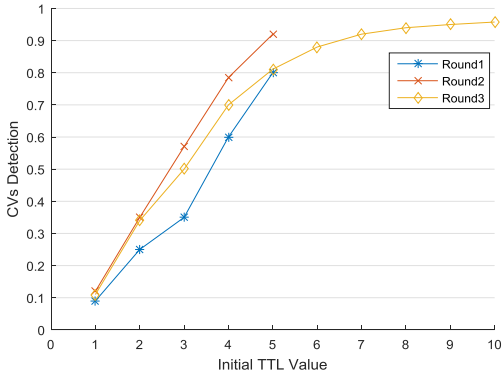
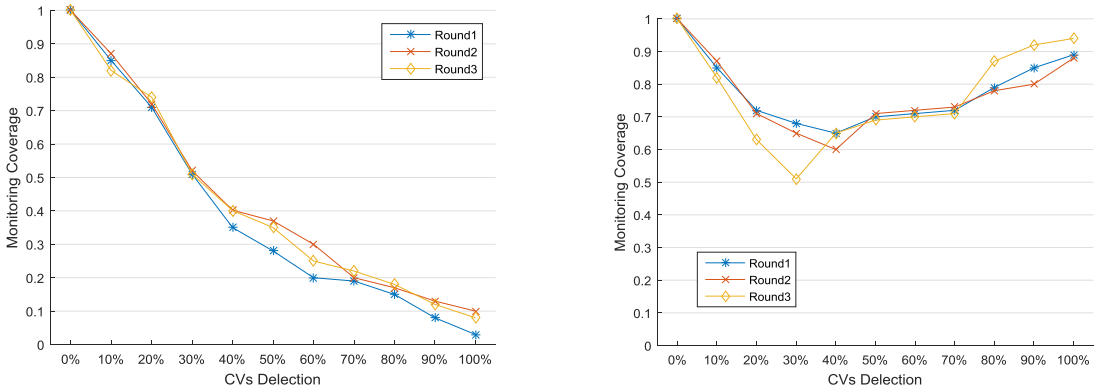


Figure 6. Accuracy of CVs detection

At the same time, random deletion of key nodes is performed in the experiment to verify the impact of key nodes on the survivability of the detection system. In this experiment, there is no improved neighborhood construction algorithm, that is, there is no shadow proxy node in the neighborhood, so each neighborhood proxy node is a cut vertex. Figure 7 is an experimental comparison of the algorithms without running cut vertex detection, repair, and operation.



(a) Before repair (b) After repair

Figure 7. Effect on survivability without and with CVs neutralization

As shown in the figure above, abscissa is the cut vertex deletion ratio and ordinate is the monitoring coverage, that is, the coverage of the detection system to the nodes. From Figure 7(a), we can see that the coverage of system monitoring decreases linearly with the increasing percentage of cut vertexes deleted. Finally, when the known cut vvertexes are deleted completely, the coverage of system monitoring is less than 10%. This is because after the key nodes are deleted in the previous stage, the scattered nodes aggregate to form a new monitoring area, which can be found. In Round 3, by deleting less than 4% of the nodes, the monitoring domain will lose 90% of the node coverage. From Figure 7(b), it can be seen that the survivability of the system has been significantly improved after running cut vertex detection and repair functions. However, deleting cut vertexes in the pre-existing period will result in a decrease in coverage, because active detection and repair of cut vertexes in Round 3 need to consume a certain number of message rounds. Generally speaking, cut vertex detection and repair function

can significantly improve the survivability of the monitoring system.

5. Conclusions

Because of the reciprocity between nodes and the self-organization of monitoring domains in distributed detection systems, there are key nodes in monitoring domains. Once the key nodes fail to exit the system, a large number of nodes will be unable to be monitored, leading to partial failure of distributed detection system functions and reducing the survivability of fault detection functions. This problem is distributed in a high dynamic state. This is particularly evident in a stylized environment. For this reason, this paper designs a set of methods including adaptive detection, active detection, and repair for key nodes, which effectively solves the survivability problem of distributed fault detection systems.

With the continuous development of large-scale distributed network computing systems represented by cloud computing, virtualization is now the developmental direction of computing services. The acquisition of resource status and fault identification in virtual environments will be a research direction and application hotspot in the future.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China (No. 61032001).

References

1. F. Stann, J. Heidemann, R. Shroff, and M. Z. Murtaza, "RBP: Robust Broadcast Propagation in Wireless Networks," in *Proceedings of ACM SenSys*, pp. 85-98, Boulder, Colorado, USA, 2006
2. C. Hsu and C. Lin, "A Comparison of Methods for Multi-Class Support Vector Machines," *IEEE Transactions on Neural Networks*, Vol. 13, No. 2, pp. 415-425, 2002
3. X. Ren and S. Lv, "Analysis of ARP Protocol Deception Principle and Resistance Method," *Computer Engineering*, Vol. 29, No. 3, pp. 127-129, 2003
4. G. Chang and S. Chen, "An Efficient and Scalable Self-Organizing Neighborhood Fault Detection Protocol," *Journal of Electronic and Information Science*, Vol. 32, No. 9, pp. 2145-2150, 2010
5. I. Stojmenovic, D. Simplot-Ryl, and A. Nayak, "Towards Scalable Cut and Link Detection with Applications in Wireless Ad Hoc Net-Works," *IEEE Transactions on Networks*, Vol. 25, No. 1, pp. 44-48, 2011
6. H. Peng, D. Zhao, Y. J. Yu, Z. D. Wu, and S. Y. Wu, "Trust Model based on Trusted Computing for Distributed Heterogeneous Networks," *Computer Scie*nce, pp. 66-69, October 2016
7. P. Nageshwar, I. Nagaraju, and M. A. Kumar, "The Trusted Computing Model for Providing Security in Cloud Computing," *International Journal of Mathematics and Computer Research*, Vol. 3, No. 6, pp. 1018-1024, 2015
8. G. Proudler and L. Chen, "Trusted Platform Architecture," *Springer International Publishing*, Vol. 2, pp. 109-111, 2014
9. L. Huawei, "Research on Dependable Computing Oriented Distributed Fault Detection System," Ph.D. Thesis. College of Computer Science of Chongqing University, pp. 54-60, April 2012
10. C. Lin and X. Peng, "Research on Trustworthy Networks," *Chinese Journal of Computers*, pp. 751-757, May 2005
11. A. Tajeddine, A. Kayssi, A. Chehab, and H. Artail, "A Comprehensive Reputation-based Trust Model for Distributed Systems," in *Proceedings of Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks*, pp. 116-125, IEEE, 2005
12. L. Bhasker, "Genetically Derived Secure Cluster-based Data Aggregation in Wireless Sensor Network," *IET Information Security*, Vol. 8, No. 1, pp. 1-7, 2014