MPFL: A Distributed Fault Localization Framework for High-Performance Computing Systems

Jian Gao, Kang Yu, Peng Qing, Hongmei Wei

Jiangnan Institute of Computing Technology
Wuxi Jiangsu 214083, China
gaojian627@126.com

Abstract. Fault localization has become an increasingly challenging issue in high-performance computing (HPC) systems. Various techniques have been used for HPC systems. However, as the HPC systems scale out, resulting in the rapid deterioration of the existing techniques. In this context, we propose a message-passing based fault localization framework, namely MPFL, which provides a light-weight distributed service using tree-based fault detection (TFD) and fault analysis (TFA) algorithms. In essence, MPFL serves as a fault localization engine within message-passing libraries by enabling several system middleware such as job scheduler to provide abnormal information. We present details of the MPFL framework, including the implementation of TFD and TFA. Further, we develop the fault localization engine prototype within MVAPICH2. The experimental evaluation is performed on a typical HPC cluster with 10 computing nodes, which demonstrate the capability of MPFL and show that the MPFL service does not affect the performance of an application in practice.

1 Introduction

High-performance computing (HPC) systems are widely used in important fields such as national defense, scientific research and finance. However, as these systems scale out, their mean time between failures (MTBF) rapidly deteriorates, even has reduced from days to a couple of hours. According to the discussion of reliability theory [1], if a *fault* is activated, the internal state of the system will be abnormal, also known as *error*. The error will continue to propagate and lead to the system *failure*. That is, system failure is a gradual process caused by the fault.

Fault localization is a process of deducing the exact source of system failure, acting as an important foundation for maintaining the reliability of HPC systems. It mainly includes two stages: (1) *Fault detection*, responsible for timely collecting all abnormal behavior in system, which is also known as *symptoms*. (2) *Fault analysis*, to quickly and accurately perform fault reasoning according to the collected symptoms. Obviously, an efficient fault localization scheme can help the systems to take the appropriate treatment strategy before failure and avoid the fault propagation.

The rest of the paper is organized as follows: in Section 2, we present the research results previously obtained in fault localization. Section 3 describes the MPFL framework in detail, including the design and implementation of fault detection and

analysis. Experimental evaluation is presented in Section 4, which demonstrate the capability, and evaluate the performance impact of MPFL. We conclude in Section 5.

2 Related Work

At present, from the different computer fields derived fault localization approaches are mainly divided into two categories: *event correlation* [2] and *active probing* [3].

2.1 Event correlation

Event correlation is an important fault localization method in network management systems. *Event*, is similar to symptom, defined as an exceptional situation occurring in the system. By correlating event messages, a system can provide a more concise view of events such that operators can accurately identify the underlying faults [2].

Steinder et al. [4] conducted a more comprehensive review of event correlation techniques, including analysis methods based on rules, models and cases. Event correlation based on the fault propagation model (FPM) is a common approach for the HPC systems. The FPM describes which symptoms may be observed if a specific fault occurs [5]. The FPM includes the representation of all faults and symptoms that occur in the system. Observed symptoms are mapped into FPM. The fault analysis algorithms study the FPM to identify the best explanation of the observed symptoms. However, there are many disadvantages to this approach:

(1) The real-time nature of this approach is not strong enough. The hysteresis of fault localization may lead to more faults in the system, or even failure.
(2) To create the FPM, an accurate knowledge of current dependencies among abstract and physical system components is required. But the complexity of HPC systems is proportional to the physical scale.
(3) The observed symptoms may be ambiguous, inconsistent, and incomplete, even some key symptoms are lost during the transmission process.

2.2 Active Probing

Rish et al. [3] proposed a fault localization approach called active probing, which performs fault detection and analysis by using various probes. *Probe* is an end-to-end test transaction which be sent from the specific *probe station* and responsible for collecting information about system components. For example, the *ping* is probably the most popular probe that can be used to detect the availability of network.

Active probing selects and sends different probes as needed in response to problems that actually occur. Natu et al. [6] indicate that the key factor affecting the active probing is the selection of probes and probe stations. Patil et al. [7] summarize and compare the different algorithms, such as greedy search, binary search algorithms. Compared with event correlation, the advantages of active probing are active, targeted and real-time. More importantly, active probing can avoid the effect of symptoms delay or loss on the accuracy. But active probing also has obvious defects:

(1) In HPC systems, the number of probes required is greater and the complexity of probes is higher. Executing a large number of complex probes will increase the overhead and consume valuable computing resources dramatically.
(2) The number of the probes with strong capability as well as the probe stations which can sent appropriate probes are limited.
(3) Selecting the required probe stations and probes perfectly has been proved to be NP-Hard [6]. The execution time of the optimal algorithm to select probes and probe stations increases exponentially with increase in system scale.

Our works develops a framework called MPFL that addresses these limitations in both the above two approaches. MPFL supports the system to obtain abnormal states of nodes at runtime by using the tree-based fault detection (TFD). And, MPFL develop the tree-based fault analysis (TFA), which properly integrates the advantages of the above two approaches. More details are described in the next section.

3 MPFL Framework

The nodes of the HPC system are interconnected by specific hardware and high-performance network such as InfiniBand [8]. These nodes are isomorphic, and their states have similarities when normally performing tasks. Although each node runs independently, they collaborate on a task by communicating with each other.

Considering that the message-passing is widely used for communication between nodes, MPFL takes full advantage of message-passing to obtain node states in order to enable detection and analysis of abnormal information when the system is running. The implementation of fault localization at runtime facilitates the system to handle faults in a timely manner and avoid failure.

3.1 Generic Architecture

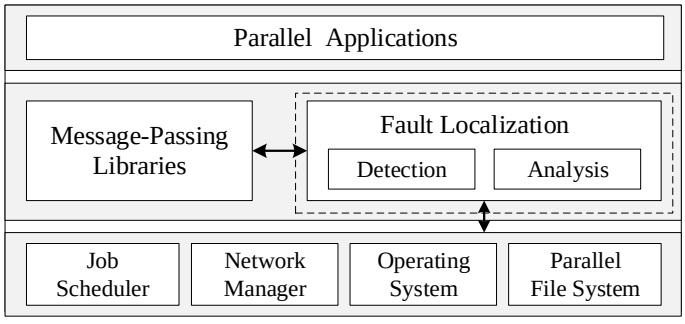

Fig. 1. Generic architecture of MPFL

Fig. 1 illustrates the contributions of this paper and shows how existing system software stack can make use of the proposed MPFL. The MPFL and message-passing

libraries are at the same layer, they collaborate closely to provide fault localization service for the parallel applications. Meanwhile, the MPFL can get support from system middleware ranging from operating system and job scheduler to network manager, parallel file system. In addition to existing middleware, third party developers can set up other MPFL-enabled middleware such as automatic scripts.

We propose the generic architecture shown in Fig. 1, mainly for the following aspects of consideration:

(1) Subject to job scheduling algorithm, node allocation strategy and other factors, many faults are often activated only in specific runtime environments. Practice has proved that these faults are difficult to reproduce and debug. The MPFL that works at runtime is conducive to locate such faults.
(2) Many computational tasks such as Finite Element Analysis [9] require a large number of nodes to participate in the computing. The faults of some nodes are more likely to be reflected in other nodes, so it is necessary to cooperate with multiple nodes for comprehensive analysis based on message-passing.
(3) Because the MPFL obtains the internal state of nodes at runtime by tight coupling with the message-passing library, the obtained symptoms are real-time and targeted. We no longer need to spend huge amounts of time and computational costs in the data mining work for massive system logs.
(4) Several system middleware provide support for fault detection by relying on external components such as system monitor to obtain abnormal information. However, they work independently, we want to conduct a comprehensive analysis by summarizing the information from different middleware.

3.2 Tree-based Fault Detection (TFD)

At present, HPC systems generally adopt the global centralized fault management. However, with the expansion of system scale, many fault indicators such as probability, complexity and correlation are also increased accordingly. The centralized approach is easy to encounter the bottleneck problem.

In order to solve the bottleneck problem, MPFL assigns the fault localization tasks to multiple computing nodes. Different nodes are responsible for fault detection and analysis in different domains by performing a light-weight fault localization process. In more detail, when the system initializes the job, all the computing nodes organize themselves into a tree-based logical topology. This topology is named a *fault localization tree (FLT)* by MPFL and each job has its own FLT. The initial topology of FLT takes place with the assistance of the external configuration which provides information that helps every computing node determine its parent and children. In the FLT, each node except the root node has a unique parent. MPFL specifies that the parent is responsible for collecting and analyzing the symptoms associated with its children. That is, the parent is the *fault localization node (FLN)* of all child nodes.

During the lifetime of a job, the structure of FLT will not change. If a node finds that its FLN is in the unacceptable state such as overload or failure, it can select its own *substitute of fault localization node (SFLN)* according to the configuration file. The nodes in a same sub-tree are more likely to have strong correlation, so a node who loses its FLT usually selects a higher-level node in the same sub-tree as SFLN.

Although each non-root node has a unique FLN, it has more than one SFLN. It should be noted that, the root node is responsible for receiving the fault localization results from non-root nodes and reporting to the user. Correspondingly, the user can manage the job's fault localization process through the specific interface of root node.

The goal of fault detection is timely collects symptoms caused by a fault. The temporal and spatial properties of symptoms are closely related to the accuracy of fault analysis. The temporal property requires the collection must be done before the system failure and as timely as possible. An excellent symptom sequence should be able to reflect the process of node state changes over time. And, the spatial property refers to whether the collected symptoms can cover all possible faults, it is necessary to summarize the information from multiple system middleware.

The symptom is defined as information about any situation that has caused or can cause excessive errors or can stop the system from working. Table 1 shows some symptoms from various MPFL-enabled middleware. MPFL can subscribe to be notified about one or more symptoms of interest from other middleware.

Table 1. Symptoms from various MPFL-enabled middleware

MPFL-enabled middleware	Related symptoms
Message-passing libraries	The timeout or error of messages
Network manager	The port down or link down
Job scheduler	No heartbeat has been received from a node
Parallel file system	The I/O error or performance degradation
Operating system	The abnormal state of CPU or memory

The pseudocode of TFD is shown in Algorithm 1. After initialization, each node perform normal work independently. If any of the MPFL-enabled middleware detects a symptom from the *suspected fault node* (SFN), it will notify its FLN or SFLN according to the relevant indicators such as CPU utilization. For each node, the received symptoms are stored in the symptom set, which will be further analyzed.

Algorithm 1. The TFD algorithm

Premise: After the initialization of a job, the FLT is visible for each node.
For each node:

```
1.  WHILE (Job_Finished != true) DO
2.      IF Find_Symp(SFN) THEN
          /* The MPFL-enabled middleware detects a symptom from SFN */
3.          IF Available_FLN(SFN) THEN
              /* The availability of the SFN's FLN must be determined */
4.              Send the symptom to SFN's FLN;
5.          ELSE
6.              Select the SFN's SLFN according to the structure of FLT and configuration;
7.              Send the symptom to SFN's SFLN;
8.          END IF
9.      END IF
10.     IF (Recv_Symp == true) THEN
11.         Store_Symp(Symptom);
12.     END IF
13. END WHILE
```

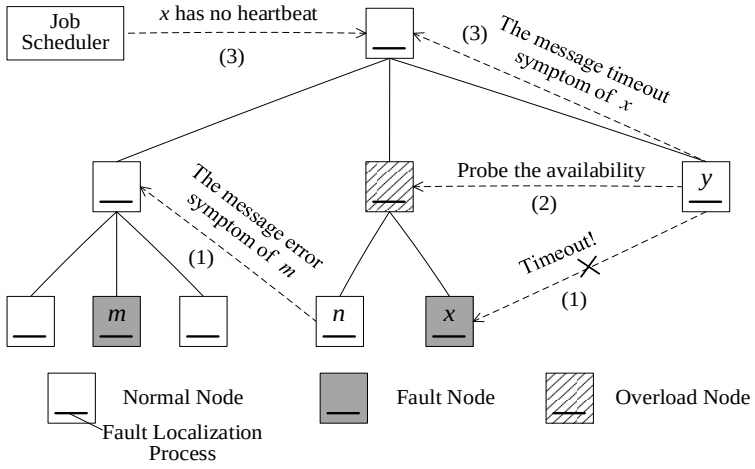

Fig. 2. A typical example of the TFD algorithm

Fig. 2 describes the work-flow of TFD running on a representative computing cluster installation with a set of computing nodes. Certainly, the topology illustrated in Fig. 2 is a logical rather than a physical structure that represents the FLT for a job of the cluster. The numbers indicate the order in which the events occur:

(1) Firstly, the node *n* sent a message error symptom of fault node *m* to *m*'s FLN. At the same time, the node *y* detected the timeout symptom after a point-to-point message was sent to the fault node *x*. This means that the fault detection and analysis of different fault nodes can be parallel.

(2) Then, when node *y* tries to send the timeout symptom to node *x*, it found that the FLN of node *x* has been overloaded so that no more symptoms are accepted. It is necessary to point out that the job scheduler has excluded the FLN of node *x* in the same way, but is not shown in the figure for simplicity.

(3) Finally, the node *y* selected the root node as the receiving node of the timeout symptom. That is, the root node is the SFLN of node *x* in this case. Simultaneously, the job scheduler told the root node that no heartbeat has been received from node *x*. This shows that different MPFL-enabled middleware can also perform fault detection in parallel.

Besides summarizing the symptoms from multiple MPFL-enabled middleware, MPFL also assigns the global fault localization tasks to all non-leaf nodes of FLT. Each non-leaf node only need to manage local domain that can be adjusted adaptively. Moreover, multiple faults can be processed in parallel, the efficiency of fault localization is greatly improved so that the bottleneck problem is clearly alleviated.

3.3 Tree-based Fault Analysis (TFA)

Fault analysis be performed when the symptom set in a node meets some sort of trigger condition, for example, the number of symptoms exceeds the threshold. The

goal of fault analysis is to quickly and accurately deduce the fault by analyzing the symptom set and taking the initiative to get more information.

However, usually the number of symptoms is large enough to greatly increase the complexity of fault analysis. Multiple symptoms may be a result of (a) fault re-occurrence, (b) the same symptoms caused by different faults, (c) the different symptoms caused by a single fault, and (d) error propagation between different components causing them to fail and generate additional symptoms. In serious cases, some specific faults may lead to symptom storm.

As previously stated, the performance of event correlation or active probing used in HPC systems is not very satisfactory. Thus, we proposed the TFA algorithm that integrates the advantages of event correlation and active probing in order to improve the efficiency of fault analysis. Fig. 3 describes the structure of TFA algorithm. The following paragraphs describe different parts of the TFA algorithm in detail.

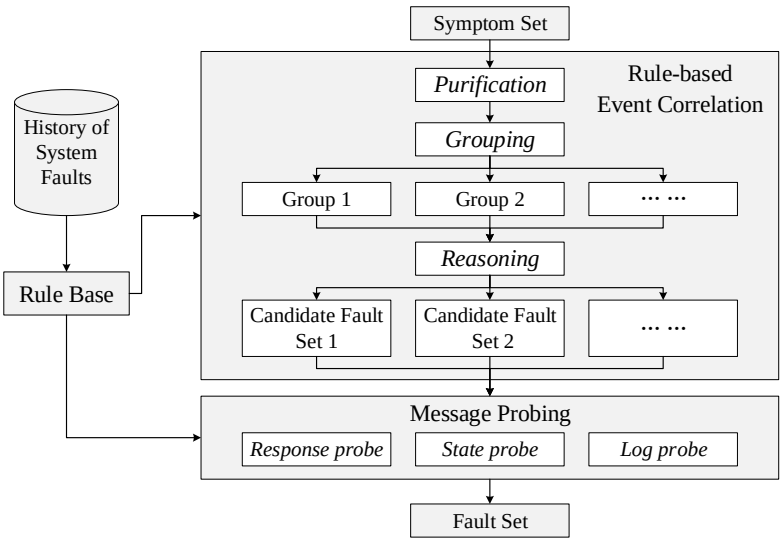

Fig. 3. The structure of the TFA algorithm

Rule-based Event Correlation. The implementation of rule-based is the most widely used way of event correlation. This approach tries to reflect actions of a human expert when solving problems in a particular domain. It requires the creation of a rule base in advance, which may be either surface-resulting from experience, or deep-resulting from understanding the system behavior.

The expression of a rule is usually: *IF condition A THEN action B*. The inference engine uses a forward-chaining inferencing mechanism, which executes in a sequence of rule-firing cycles. According to Schroeder et al's analysis and discussion on the probability, position and time distribution of the faults in HPC systems [10], and Jakobson et al's classification and summary on the compression, clustering, generalization and other event handling rules [11], the event correlation in the TFA algorithm mainly consists of three steps:

(1) *Purification*, responsible for excluding a large number of repetitive, redundant, meaningless events, thereby effectively reducing the number of events.
(2) *Grouping*, merges different events that are relevant into an event group, as a result, the symptom set is divided into different event groups, the semantics of events are fully enhanced.
(3) *Reasoning*, respectively deduces the corresponding candidate set, which contains all possible faults that may trigger the specific event group.

Message Probing. By drawing on ideas of active probing, we develop the message probing to achieve point-to-point test transactions based on message-passing, which can get more extra information in order to identify the most appropriate fault in the candidate fault set. Message probing includes three types of message probes:

(1) *Response probe*, is used to determine whether the target node can communicate normally, that is, to detect the availability of the node.
(2) *State probe*, responsible for obtaining specific performance indicators of the target node such as bandwidth, memory utilization.
(3) *Log probe*, requires the target node to return its message log, which typically records the type, time, tag etc. of each message associated with the node.

When message probing works, it uses different probing strategies for the corresponding candidate fault set, including the combination and execution order of the probes. In general, message probing can be viewed as a light-weight active probing for HPC systems, because (a) message probes are easy to implement and deploy, (b) each computing node can be used as a probe station to send the probe, (c) the execution of message probes does not increase the network load too much.

Rule Base. The rule-based event correlation and message probing need to be supported by the rule base, which can be obtained by mining the correlation among the historical data of system faults. Also, the rule base provides an interface for users to dynamically deploy and update rules. For example, Table 2 shows some of the rules related to node outage fault.

Table 2. Some of the rules related to node outage fault

Analysis Steps	Related rules
Purification	**IF** the node *x* reported that node *y* message timeout > N times **THEN** the symptom will only be saved once
Grouping	**IF** M different symptoms satisfy the constraint of time window T **THEN** merge these symptoms into the same event group
Reasoning	**IF** S nodes reported that node *y* timeout **and** node *y* has no heartbeat **THEN** infer node *y* outages, link down or port down
Message probing	**IF** the response probes sent by multiple nodes to node *y* are invalid **THEN** determine the node *y* outages

Algorithm 2 describes the pseudocode of TFA. The node performs fault analysis start with using the rule-based event correlation to reason the symptom set. Once this work is completed, we can obtain multiple candidate fault sets, which will be further analyzed using message probing. It must be said that, although the message probing

strategies for different candidate fault sets are not the same, they may get the same actual fault. We finally can get a collection containing one or more actual faults.

Because global fault localization tasks are assigned to multiple nodes and the symptoms obtained at runtime are timely and targeted, the TFA algorithm effectively alleviates the performance degradation of event correlation in HPC systems. Besides, the message probing is well suited for HPC systems. It can obtain the node states proactively so that avoid negative impact due to the delay or loss of symptoms.

Algorithm 2. The TFA algorithm

Input: the symptom set *Symp_Set*
Output: the fault set *Fault_Set*

```
1.  Initialization: Fault_Set = ∅
2.  Purification_Set = Event_Purification(Symp_Set);
3.  Grouping_Set = Event_Grouping (Purification_Set);
4.  FOR i=0 to |Grouping_Set| DO
      /* respectively deduces the corresponding candidate set */
5.      Candidate_Faults_i = Event_ Reasoning (Group_i);
6.      Add Candidate_Faults_i to Candidate_Fault_Sets;
7.  END FOR
8.  FOR i=0 to |Candidate_Fault_Sets| DO
      /* message probing for different candidate fault set */
9.      FOR j=0 to |Candidate_Faults_i| DO
10.         IF Msg_Probing(Candidate_Faults_ij) THEN
11.             Add Candidate_Faults_ij to Fault_Set;
12.             break;
13.         END IF
14.     END FOR
15. END FOR
16. Release(Symp_Set);
17. Return(Fault_Set);
```

4 Experimental Evaluation

In this section, we describe the simulation experiments to demonstrate the capability, and evaluate the performance impact of MPFL. The computing cluster used for the experimental evaluation is a 10-node Linux-based cluster. Each computing node has 16 Intel Xeon cores organized as 2 sockets with 8 cores per socket and has 64 GB of memory. The nodes of cluster are interconnected through an InfiniBand network. The operating system used is Red Hat Enterprise Linux Server release 6.3. The message-passing library used is MVAPICH2-2.2.

4.1 Capability

The computing node outage fault is the main cause that disturbs the stability of parallel applications and even leads to the system failure [12, 13]. This experiment

focuses on locating the node outage fault to demonstrate the capability of MPFL. We have developed the fault localization engine prototype within the MVAPICH2 library, which includes both the TFD and TFA. The following paragraphs describe how to simulate the node outage fault.

By simply modifying the MVAPICH2 source code, all processes know which processes are simulating the node outage fault after the application is initialized based on the configuration file. And, ensuring that the process which simulates a node outage fault neither sends nor receives any messages, or even does any calculations. In other words, it doesn't do anything. In addition, multiple symptoms related to a node outage fault listed in Table 1 and others were injected into the application in order to simulate the behavior of other MPFL-enabled middleware.

Because the outage of a node usually causes all the applications on the node to fail, running a single parallel application with only one process that simulates the outage fault is enough. And all processes in the application perform *all-to-all* communication multiple times to simulate heavy communication. Each process implements the fault detection and fault analysis based on multiple threads. Table 3 shows the specific parameters of this experiments.

Table 3. Specific parameters of this experiments

Parameters	Meaning	Values
N	Number of experimental groups	4
M	Number of repetitions for each group	100
P	Number of processes in the application	16, 32, 64, 128
P_f	Rank of process that simulates the outage fault	randomly selected, $0<P_f<P$

Experimental results show that the process P_f always be found in a few seconds. Compared with the absence of MPFL, it is not necessary to hand over a lengthy error context to the user for manual analysis. Therefore, it can be considered that the MPFL framework we proposed is effective.

In fact, because some rules can’t cover all situations in the system, the correct rate of fault localization is almost impossible to reach 100%. For example, the grouping rule in Table 2 merges multiple symptoms into the same event group depending on the value of the time window T. If the value of T is too small, some symptoms may arrive outside of the time window due to the delay of the message. On the contrary, if the value of T is too large, the relevance between events in the same event group will be reduced. In either case, the accuracy of the fault analysis is compromised.

4.2 Scalability

We evaluate the MPFL overhead on the Fast Fourier Transformation (FT) and the Integer Sort (IS) benchmarks from the NAS Parallel Benchmarks (NPB) [14]. The NPB consists of five kernels and three pseudo-applications that mimic the computation and data movement characteristics of large scale CFD [15] applications.

The FT is a popular benchmark stressing global communications. In particular, the FT benchmark begins with an initialization phase followed by a certain number of

iterations. Each iteration contains multiple all-to-all communication. And, the IS benchmark performs a sorting operation that is considered important in particle physics code, which also contains multiple all-to-all communication.

We evaluate the original FT benchmark (version 3.3) without MPFL and compare it against the modified MPFL-enabled FT. In order to simulate the situation that the application be notified about symptoms of interest from the MPFL-enabled middleware, multiple symptoms that does not lead the TFA engine to derive wrong conclusions were injected periodically when the MPFL-enabled FT is running. For example, the operating system reports CPU utilization within the normal range. Certainly, we will do same evaluation for the IS benchmark.

Because the FT benchmark requires a number of processes with a power of 2, we conducted four groups of comparative experiments, the number of processes were 16,32,64 and128 respectively. In addition, the benchmark size we choose is CLASS A. The experimental results of the NPB-FT are shown on the left side of Fig. 4, while the NPB-IS are shown on the right side. Obviously, the compute performance for the original application and MPFL-enabled application is close. These results indicate that the MPFL service does not affect the performance of an application in practice.

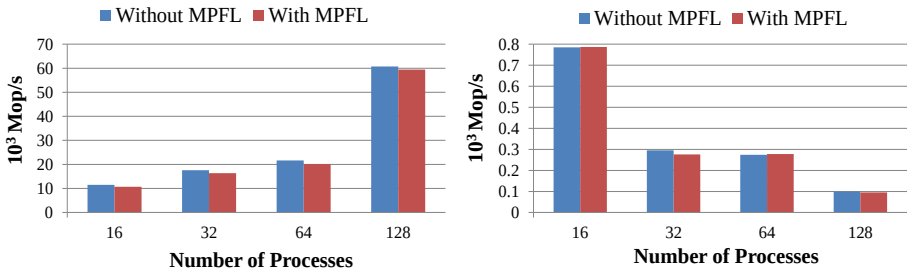

Fig. 4. The results of NPB-FT (*left*) and NPB-IS (*right*) benchmarks (CLASS=A)

5 Conclusions

Current fault localization techniques have several disadvantages when applied to HPC systems. The lack of an efficient and real-time approach prevents HPC systems from finding fault timely and taking proactive decisions that could improve the overall reliability of the systems. In this paper, we proposed the MPFL, a fault localization framework for HPC systems based on message-passing. The goal of MPFL is to provide a light-weight distributed service that enables HPC systems timely and accurately performs fault detection and analysis by summarizing the abnormal information from different middleware. We developed the TFD and TFA algorithms to achieve this goal. We presented the architecture of MPFL along with the design and implementation of TFD and TFA. We also described a detailed experimental evaluation to demonstrate the capability, and evaluate the performance impact of MPFL. In the future work, we plan to provide an interface specification that enables

various middleware to better support MPFL and mining historical information from important system logs more deeply to enhance the TFA algorithm. We also plan on making more comprehensive evaluation for MPFL.

Acknowledgments. The MPFL project is an on-going collaborative project between multiple teams from Jiangnan Institute of Computing Technology. We would like to thank every member for their constructive suggestions and contribution during the design phase of MPFL. This research was financially supported by The National Key Research and Development Program of China (Grants No. 2016YFB0200502).

References

1. Siewiorek D, Swarz R. Reliable Computer Systems: Design and Evaluatuion[M]. Digital Press, 2017.
2. Liu G, Mok A K, Yang E J. Composite events for network event correlation[C]//Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on. IEEE, 1999: 247-260.
3. Rish I, Brodie M, Odintsova N, et al. Real-time problem determination in distributed systems using active probing[C]//Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP. IEEE, 2004, 1: 133-146.
4. łgorzata Steinder M, Sethi A S. A survey of fault localization techniques in computer networks[J]. Science of computer programming, 2004, 53(2): 165-194.
5. Ficco M. Security event correlation approach for cloud computing[J]. International Journal of High Performance Computing and Networking 1, 2013, 7(3): 173-185.
6. Natu M, Sethi A S. Active probing approach for fault localization in computer networks[C]//End-to-End Monitoring Techniques and Services, 2006 4th IEEE/IFIP Workshop on. IEEE, 2006: 25-33.
7. Patil B M, Pathak V K. Survey of Probe Set and Probe Station Selection Algorithms for Fault Detection and Localization in Computer Networks[J]. Transactions on Networks and Communications, 2015, 3(4): 57.
8. Panda D K. Tutorial: InfiniBand Architecture[J]. 2001.
9. Peng J, Lu J, Law K H, et al. ParCYCLIC: finite element modelling of earthquake liquefaction response on parallel computers[J]. International Journal for Numerical and Analytical Methods in Geomechanics, 2004, 28(12): 1207-1232.
10. Schroeder B, Gibson G. A large-scale study of failures in high-performance computing systems[J]. IEEE Transactions on Dependable and Secure Computing, 2010, 7(4): 337-350.
11. Jakobson G, Weissman M. Real-time telecommunication network management: extending event correlation with temporal constraints[M]//Integrated Network Manage- ment IV. Springer US, 1995: 290-301.
12. Schroeder B, Gibson G A. Understanding failures in petascale computers[C]//Journal of Physics: Conference Series. IOP Publishing, 2007, 78(1): 012022.
13. Wu L, Meng D, Liang Y, et al. LUNF-A cluster job scheduling strategy using characterization of nodes' failure[J]. Jisuanji Yanjiu yu Fazhan(Comput. Res. Dev.), 2005, 42(6): 1000-1005.
14. Bailey D H, Barszcz E, Barton J T, et al. The NAS parallel benchmarks[J]. The International Journal of Supercomputing Applications, 1991, 5(3): 63-73.
15. Ragheb M. Computational fluid dynamics[J]. 1976.