RESEARCH ARTICLE OPEN

# To Improve Reliability of Message Passing In MPI Libraries Using Flow Checker

[#1] N.Kanagavalli, S.Krishanth[#2]
[#1] Assistant Professor, Department of Information Technology, Saveetha Engineering College
Chennai, India [1] kanagavalli@saveetha.ac.in
[#2] PG Scholars, ME Software Engineering, Saveetha Engineering College
Chennai, India [#2]skrishanth@gmail.com

**Abstract**—
This paper presents Standard testing methods of MPI programs do not guarantee coverage of all non-deterministic interactions (e.g., wildcard-receives). Programs tested by these methods can have untested paths (bugs) that may become manifest unexpectedly. Previous formal dynamic verifier's cover the space of non-determinism but do not scale, even for small applications. We present DAMPI, the first dynamic analyzer for MPI programs that guarantees scalable coverage of the space of non-determinism through a decentralized algorithm based on Lamport-clocks. DAMPI computes alternative non-deterministic matches and enforces them in subsequent program replays. To avoid interleaving explosion, DAMPI employs heuristics to focus coverage to regions of interest. We show that DAMPI can detect deadlocks and resource-leaks in real applications. Our results on a wide range of applications using over a thousand processes, which is an order of magnitude larger than any previously reported results for MPI dynamic verification tools, demonstrate that DAMPI provides scalable, user-configurable testing coverage. The MPI issues a Flow checker to both the sender and the receiver after the verification of the secret key. The generation of the Flow checker involves the selection of 8-bit random key using the appropriate function available in .Net. By using RSA algorithm session key is generated. The session key is converted into binary from which the last two binary digits are chosen through which the Flow checker is created. Once the Flow checker matches on the sender and the receiver side, the data can be encrypted and the intermediate encrypted form is viewable. Similarly after decryption the encrypt decrypt file is also available. Thus a secure transmission of data takes place between the sender and the receiver using MPI Libraries

**Keywords:** Flow checker, MPI algorithm, RSA algorithm, encryption, decryption.

## I. INTRODUCTION

In Flow checker cryptography, Flow checker (FC) employ MPI mechanisms to distribute session keys and public discussions to check for eavesdroppers and verify the correctness of a session key. However, public discussions require additional communication rounds between a sender and receiver and cost precious quebits. By contrast, classical cryptography provides convenient techniques that enable efficient key verification and user authentication. Key Distribution Protocols are used to facilitate sharing secret session keys between users on communication networks. By using these shared session keys, secure communication is possible on insecure public networks. However, various security problems exist in poorly designed key distribution protocols; for example, a malicious attacker may derive the session key from the key distribution process. A legitimate participant cannot ensure that the received session key is correct or fresh and a legitimate participant cannot confirm the identity of the other participant. Designing secure key distribution protocols in communication security is a top priority.

## II. BACKGROUND

### A. *Flow Checker*

Flow checker (FC) which works on network security by the use of key agreement. Secrete Key which is used by each user in the network. Each user has unique Secrete and which will be shared by each user to MPI. In MPI we have generate a Key for network Security with the Help of Algorithms and Flow checker Mechanics. Through that we have to prove how secure the data has been transmitted over network to receiver.

### B. *Flow Checker Process*

Flow checker (FC) is a method of securely distributing cryptographic key material for subsequent cryptographic use. In particular, it is the sharing of random classical bit strings using Flow checker states. Its use of a set of non-orthogonal Flow checker states then requires this key material to be considered Flow checker information. The Flow checker encoding of cryptographic keys for distribution is valuable because the no-cloning theorem and the superposition principle governing Flow checker states confer a uniquely powerful form of information security during transmission of key bits. For maximal security, it can be followed by one-time pad message encryption, which is the only cryptographic method that has been proven to be unbreakable once a random key has been securely shared. Flow checker—the creation of secret keys from Flow checker mechanical correlations—is an example of how physical methods can be used to solve problems in classical information theory. Flow checker Cryptography, or Flow Checker Distribution (FCD), uses Flow checker mechanics to guarantee secure communication. It enables two parties to produce a shared random bit string known only to them, which can be used as a key to encrypt and decrypt messages. An important and unique property of Flow checker cryptography is the ability of the two communicating users to detect the presence of any third party trying to gain knowledge of the key. This result from a fundamental part of Flow checker mechanics: the process of measuring a Flow checker system in general disturbs the system. A third party trying to eavesdrop on the key must in some way measure it, thus introducing detectable anomalies. By using Flow checker superposition or Flow checker entanglement and transmitting information in Flow checker states, a communication system can be implemented which detects eavesdropping. If the level of eavesdropping is below a certain threshold a key can be produced which is guaranteed as secure (i.e. the eavesdropper has no information about), otherwise no secure key is possible and communication is aborted.

### *C. Flow checker cryptography*

The security of Flow checker (FC) cryptography relies on the foundations of MPI mechanics, in contrast to traditional public key cryptography which relies on the computational difficulty of certain mathematical functions, and cannot provide any indication of eavesdropping or guarantee of key security. Flow checker cryptography is only used to produce and distribute a key, not to transmit any message data. This key can then be used with any chosen encryption algorithm to encrypt (and decrypt) a message, which can then be transmitted over a standard communication channel. The algorithm most commonly associated with Flow checker is the one-time pad, as it is provably unbreakable when used with a secret, random key.

## III. RELATED WORK

G.Carozza et al. [11] addresses the problem of software fault diagnosis in complex safety critical software systems. The transient manifestations of software faults represent a challenging issue since they hamper a complete knowledge of the system fault model at design/development time. By taking into account existing diagnosis techniques, the paper proposes a novel diagnosis approach, which combines the detection and location processes. More specifically, detection and location modules have been designed to deal with partial knowledge about the system fault model. To this aim, they are tuned during system execution in order to improve diagnosis during system lifetime. A diagnosis engine has been realized to diagnose software faults in a real world middleware platform for safety critical applications. Preliminary experimental campaigns have been conducted to evaluate the proposed approach.

R.M.Kirby et al[12] considers the problem of formal verification of MPI programs operating under a fixed test harness for safety properties without building verification models. In this approach, they directly model-check the MPI/C source code, executing its interleaving with the help of a verification scheduler. Unfortunately, the total feasible number of interleaving is exponential, and impractical to examine even for our modest goals. Our earlier publications formalized and implemented a partial order reduction approach that avoided exploring equivalent interleaving, and presented a verification tool called ISP. This paper presents algorithmic and engineering innovations to ISP, including the use of OpenMP parallelization that enables to handle practical MPI programs, including: (i) ParMETIS - a widely used hypergraph partitioner, and (ii) MADRE - a Memory Aware Data Re-distribution Engine, both developed outside our group. Over these benchmarks, ISP has automatically verified up to 14K lines of MPI/C code, producing error traces of deadlocks and assertion violations within seconds.

Z. Chen et al [13] suggest that Many MPI libraries have suffered from software bugs, which severely impact the productivity of a large number of users. This paper presents a new method called Flow Checker for detecting communication-related bugs in MPI libraries. The main idea is to extract program intentions of message passing (MPintentions), and to check whether these MP-intentions are fulfilled correctly by the underlying MPI libraries, i.e., whether messages are delivered correctly from specified sources to specified destinations. If not, Flow Checker reports the bugs and provides diagnostic information. We have built a Flow Checker prototype on Linux and evaluated it with five real-world bug cases in three widely-used MPI libraries, including Open MPI, MPICH2, and MVAPICH2. Our experimental results show that Flow Checker effectively detects all five evaluated bug cases and provides useful diagnostic information. Additionally, our experiments with HPL and NPB show that Flow Checker incurs low runtime overhead (0.9-9.7% on three MPI libraries).

## IV. EXISTING WORK

Existing MPI tools either lack coverage guarantees or do not scale well. In this paper we present DAMPI, a scalable framework that offers coverage guarantees for programs that use non-deterministic MPI calls. Our contributions include a novel method for detecting different outcomes of a nondeterministic receive without relying on a centralized process/thread. These different outcomes can be enforced through replays. We also present two different search bounding heuristics that provide the user with the ability to limit the coverage to areas of interests. We report our results on applying our tools on medium to large benchmarks running with thousands of processes, many of which make extensive use of nondeterministic calls. DAMPI is the first (and currently only) tool that can guarantee coverage at such scales.

### *A. Basic authentication scheme*

The "basic" authentication scheme is based on the model that the client must authenticate itself with a user-ID and a password for each realm. The realm value should be considered an opaque string which can only be compared for equality with other realms on that server. The server will service the request only if it can validate the user-ID and password for the protection space of the Request-URI. There are no optional authentication parameters.

### *B. Disadvantages of this scheme*

Web Browser or other client Program provides credentials in the form of username and Password. Although the scheme is easily implemented, it relies on the assumption that the connection between the client and server computers is secure and can be trusted. The credentials are passed as plaintext and could be intercepted easily. The scheme also provides no protection for the information passed back from the server.

### C. *Digest authentication scheme and its disadvantages*

MPI Digest access authentication is one of the agreed methods a web page can use to negotiate credentials with a web user (using the MPI Libraries). Digest authentication is intended to obsolete unencrypted use of the Basic access authentication, allowing user identity to be established securely without having to send a password in plaintext over the network. Digest authentication is basically an application of cryptographic hashing with usage of nonce values to prevent cryptanalysis.

Digest access authentication is intended as a security trade-off; it is intended to replace unencrypted MPI Basic access authentication which is extremely weak. However it is not intended to replace strong authentication protocols, such as Public key or Kerberos (protocol) authentication. Security wise, there are few drawbacks with Digest access authentication.

Much of the security options are optional. If quality-of-protection (qop) is not specified by server, the client will operate in a security reduced legacy mode. Digest access authentication is vulnerable to Man-in-the-middle attack; a Man-in-the-middle attack (MitM) attacker could tell clients to use Basic access authentication or legacy Digest access authentication mode. Internet Explorer does not comply with the digest access authentication standard.

### *D. Time stamp and its disadvantages*

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred. This data is usually presented in a consistent format, allowing for easy comparison of two different records and tracking progress over time; the practice of recording timestamps in a consistent manner along with the actual data is called time stamping.

MPI Message Libraries (MML) is a maintenance Libraries that allows Flow and host Mobile to swap basic control information when data is sent from one Mobile to another. It is generally considered a part of the IP layer. It allows the Mobile on a network to share error and status information. An MML message, which is encapsulated within an IP datagram, is very useful to troubleshoot the network connectivity and can be Flow checker throughout the MML.

In Timestamp Ping Operation the source workstation sends an MML Get Timestamp message and waits for an ICMP Send Timestamp response.

Although the MML timestamp ping uses little network traffic, the timestamp message is not usually found in normal network conversations. The function itself is esoteric, and although it can provide a time synchronization function for a workstation, most environments rely on Network Time Protocol (NTP) to provide clock synchronization. The MML timestamp ping relies on ICMP, which is often prevented from traversing firewalls or packet filters. This ping is probably not the best choice for scanning through firewalls.

## V. PROPOSED SYSTEM DESIGN

However, all the above trace-based tools only replay the observed schedule. They do not have the ability to analyse the observed schedule and derive from them alternate schedules that can arise if nondeterministic matches are enforced differently. This crucial ability of DAMPI helps explore traces that may never natively appear on the given platform, but can suddenly show up (and potentially result in bugs) when the code is ported to another platform. In short, the aforementioned tools do not meet our stated objectives of guaranteed non-determinism coverage and scalability.

Development of this application is highly economically feasible. We need not spend much money for the accomplishment of the project since the resources needed for the development of the system is already available. The only thing to be done is making an environment for the development with an effective supervision. If we are doing so, we can attain the maximum usability of the corresponding resources.

*A. System Architecture*

This system consists of sender, receiver, and a flow checker which enables to send and receive messages in a reliable manner. The RSA and RC4 algorithm are used for encryption and decryption purpose so that the message sent cannot be hacked.

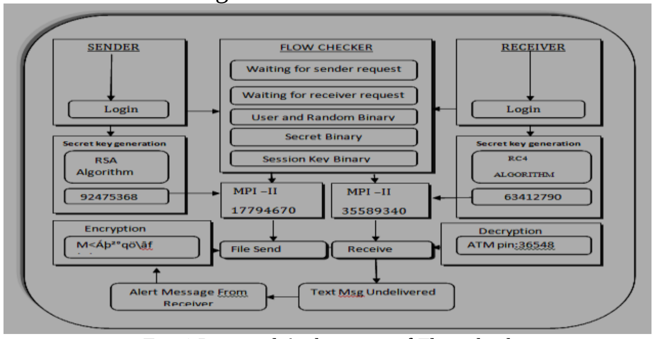


Fig. 1 Proposed Architecture of Flow checker

B. *Sender*

Getting Authorization is the first stage in sending phase. If a user wants to send a text to Destination user, he wants unique Identification. By using that Identification System knows that the person is an authorized person. This system includes (1) Registration (2) Login and (3) Receive data.

Registration is the Initial state for getting Authentication. By Providing username and Password user sets their Authentication. And System provides one more credentials that is Secrete key which is generated by the system for each user. By using username, Password and Secret key system will identify the Authorized person.

In the login if a user wants to send a file means, he/she must log in by using his/her authentication credentials. In this module we have to give username, password and Secret key which was generated by the system.If the user does not provide proper information or the given information is mismatched with database then our system shows Exception message immediately and if the user's details are verified and matched with the existing database then our system allows the person to transmit the file.

After login the MPI program calls i.e. our MPI program starts listening to the client or sender. Through Login we send the sender's secrete key for Identification.

*C. Methodology used to send the data*

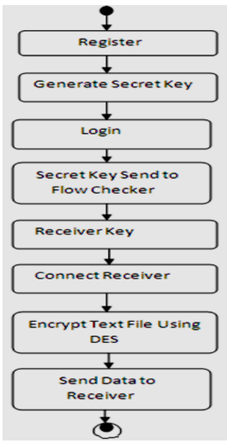


Fig. 2 Sender

The main aim of this module is to encrypt a file and send that encrypted file to receiver. Encryption will happen only if the system gets a key from MPI with in Flow checker. So after verification of user identification system will send the current user's name and his/her secrete key to MPI. In Registration phase after providing username and password, user must generate one unique key for identification. That is Secrete key. This key will take part in our final key (Flow checker). At this instance our system will store every details such as username, password, secrete key, Registration Date and Registration Time. After Registration for New User system redirects the user to Login Stage. At this stage the user must provide the relevant details which was noted or given through registration. The secret key generation is in separate class which will return.

D. *Flow checker*

The Designed Flow checker consist of MPI verification system which Verifies the secret key received from the user and authenticate the corresponding user for secure transmission, the session key generation where the shared secret key is used for encryption and decryption. The size of session key is 8 bits. This session key is generated from pseudo random prime number and exponential value of random number and the Qubit Generation to generate the secret key and random string, then convert into hex-code and then convert it into binary, find the least bit of two binary values and get the quantum bit of 0 and 1.

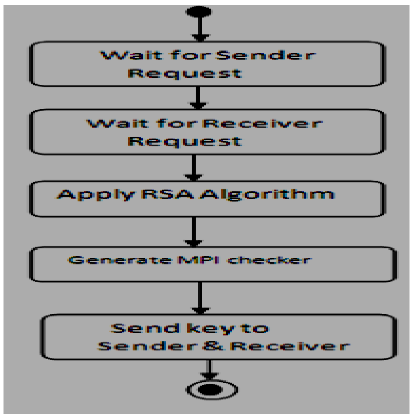


Fig. 3 Flow Checker Process

*E. Flow checker key generation and key distribution*

The To generate the quantum key using the qubit and session key which depends on qubit combinations, such as

1. If the value is 0 and 0, then 1/0.707(p[0]+p[1])
2. If the value is 1 and 0, then 1/0.707(p[0]-p[1])
3. If the value is 0 and 1, then p[0]
4. If the value is 1 and 1, then p[1].

The key distribution distributes the original session key and qubit to the sender for encryption. Also, it distributes the qubit and the session key on the receiver side for decryption.

F. *Reciever*

Getting Authorization is the first stage in receive phase. If a user wants to receive a text from source user, he wants unique Identification. By using that Identification System knows that the person is an authorized person.The receiver consist of registration , login as in sender.

G. Methodology used to receive the data

The main aim of this system is to decrypt a file. Decryption will happen only if the system gets a key from Flow Checker (FC). So after verification of user identification system will send the current user’s name and his/her secret key to Flow Checker (FC).In this phase after providing username and password, user must generate one unique key for identification. That is the secret key. This key will take part in our final key (Flow Checker). At this instance our system will store every detail such as username, password, secret key, and registration date and registration time.

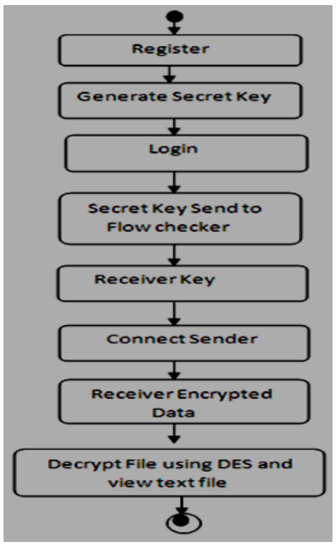


Fig. 4 Receiver Process

## VI. RESULTS AND DISCUSSIONS

A. *Registration*

The registration Page consist of two sections the secret ID and the secret pin to send and receive the message. The secret pin does not allow the unauthorized person to send or to view the data, hence the message can be kept confidential.

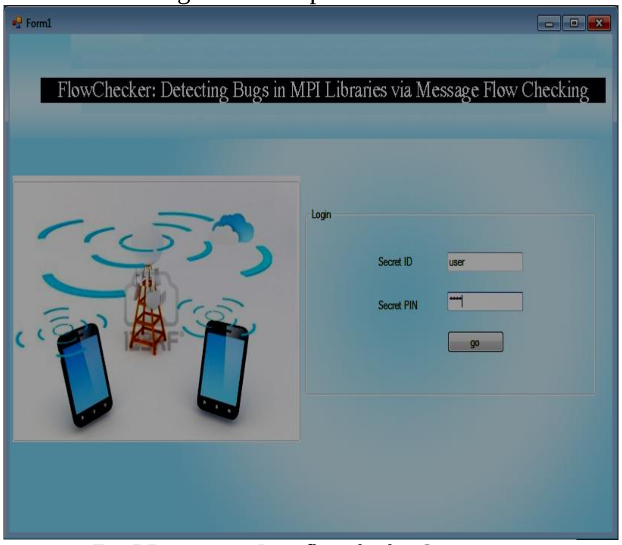
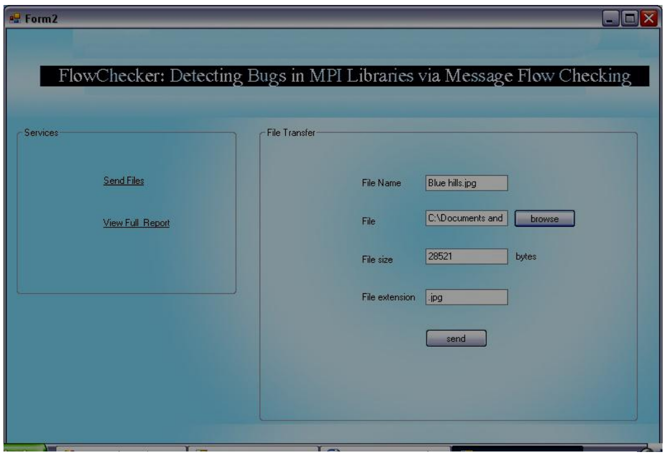

Form1
FlowChecker: Detecting Bugs in MPI Libraries via Message Flow Checking
Login
Secret ID
user
Secret PIN
go

Fig. 5 Registration Page flow checker System

B. *sender*

The sender after login can send the data to the receiver. The file name is the name of the file to be transferred and the size of the file need to be specified with its extension too.


Form2

FlowChecker: Detecting Bugs in MPI Libraries via Message Flow Checking

Services
Send Files
View Full Report

File Transfer
File Name: Blue hills.jpg
File: C:\Documents and
File size: 28521 bytes
File extension: .jpg

Fig. 6 Send data

C. *Flow checker*

The flow checker identifies the hacker through the algorithms proposed in this paper and alerts the receiver about the hackers. Receiver can be aware of all the hackers and receive the data accordingly.

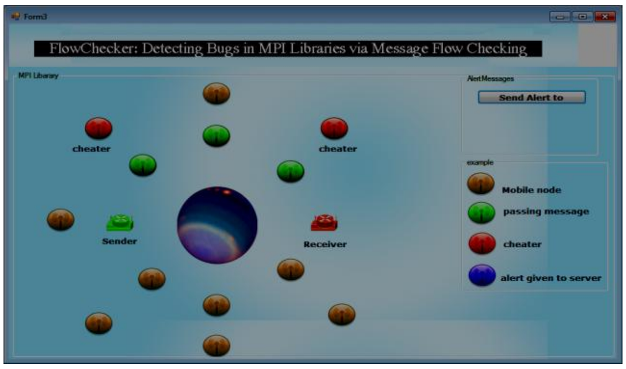
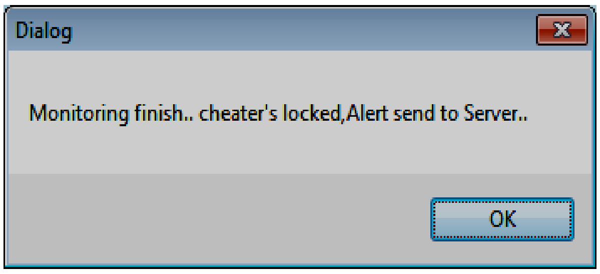


Fig. 7 Identification of hackers


Dialog

Monitoring finish.. cheater's locked,Alert send to Server..

Fig. 8 Alert sent to the receiver

D. Receiver

Once the receiver is aware of the hackers the receiver decrypts using RSA algorithm and obtain the data in a secured manner. The session time is noted and the view report can be used to view the data sent and received.

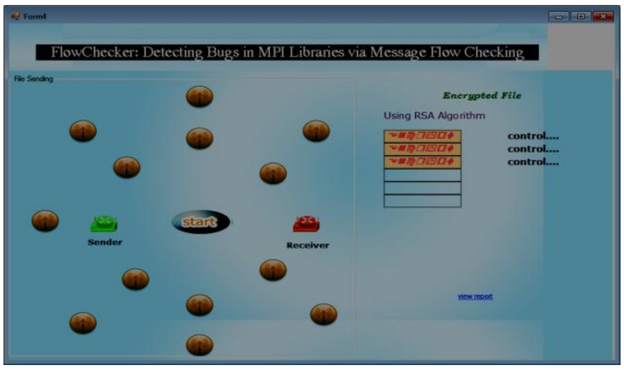


Fig.9 Receiver

## VII. CONCLUSION

This study proposed we have presented Flow Checker, a low overhead method for detecting communication-related bugs in MPI libraries. Based on collected runtime traces, it extracts MP-intentions and checks whether the underlying message flows in MPI libraries fulfil the MP-intentions. If an MP-intention is not fulfilled, Flow Checker reports the bug and provides relevant diagnostic. We have built a prototype of Flow Checker. Our evaluation with five real-world and two injected bug cases in three popular MPI libraries, including Open MPI, MPICH2, and MVAPICH2, shows that Flow Checker detects all the evaluated bug cases effectively. Additionally, Flow Checker provides useful diagnostic information for narrowing down root causes of the bugs. In fact, Flow Checker pinpoints root causes for six out of seven evaluated bug cases. Furthermore, Flow Checker incurs low runtime overhead.

## VIII. FUTURE ENHANCEMENT

The whole can be enhanced for parallel and distributed system between two systems in a local area network through the Flow checker which can be a third system in the local area network. The communication round between the sender and the receiver becomes one by applying this project as well as secret key authentication is being provided by the MPI which in turn generates the Flow checker.

## ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their valuable suggestions and comments which encouraged us towards new scope in our research.

## REFERENCES

[1.] "Message Passing Interface Forum," http://www.mpi-forum.org, 2012.

[2.] “Papers About MPI,” http://www.mcs.anl.gov/research/ projects/mpi/papers, 2012.
[3.] “Architecture Share in top 500 Supercomputers for 06/2009,” http://www.top500.org/stats/list/33/archtype, 2012.
[4.] “MPICH2: A High-Performance and Widely Portable Implementation of the Message Passing Interface (MPI) standard,” http:// www.mcs.anl.gov/research/projects/mpich2, 2012.
[5.] “MVAPICH2: MPI-2 over OpenFabrics-IB, OpenFabrics-iWARP, PSM, uDAPL and TCP/IP,” http://mvapich.cse.ohio-state.edu/ overview/mvapich2, 2012.
[6.] ] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall, “Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation,” Euro PVM/MPI, 2004.
[7.] ] J.M. Squyres and A. Lumsdaine, “A Component Architecture for LAM/MPI,” Proc. Euro PVM/MPI, 2003.
[8.] MPI Bug Tickets,” https://svn.open-mpi.org/trac/ompi/ ticket/689, 2012.
[9.] D.C. Arnold, D.H. Ahn, B.R. de Supinski, G. Lee, B.P. Miller, and M. Schulz, “Stack Trace Analysis for Large Scale Debugging,” Proc. IEEE Int’l Parallel and Distributed Processing Symp. (IPDPS), 2007.
[10.] J. DeSouza, B. Kuhn, B.R. de Supinski, V. Samofalov, S. Zheltov, and S. Bratanov, “Automated, Scalable Debugging of MPI Programs with Intel Message Checker,” Proc. Second Int’l Workshop Software Eng. for High Performance Computing System Applications (SE-HPCS), 2005.
[11.] G. Carrozza, D. Cotroneo, and S. Russo, “Software Faults agnos is in Complex OTS Based Safety Critical Systems,” Proc. Seventh European Dependable Computing Conf. (EDCC), 2008.
[12.] A. Vo, S. Vakkalanka, M. DeLisi, G. Gopalakrishnan, R.M. Kirby, and R. Thakur, “Formal Verification of Practical MPI Programs,”Proc. 14th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP), 2009.
[13.] Z. Chen, Q. Gao, W. Zhang, and F. Qin, “FlowChecker: Detecting Bugs in MPI Libraries via Message Flow Checking,” Proc. ACM/ IEEE Int’l Conf. High Performance Computing, Networking, Storage and Analysis, 2010.