Front. Comput. Sci., 2014, 8(3): 378-390
DOI 10.1007/s11704-014-3503-1

RESEARCH ARTICLE

Iaso: an autonomous fault-tolerant management system for supercomputers

Kai LU (✉)[1,2], Xiaoping WANG[1,2], Gen LI[2], Ruibo WANG[2], Wanqing CHI[2], Yongpeng LIU[2], Hongwei TANG[2], Hua FENG[2], Yinghui GAO[3]

1 Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China

2 College of Computer, National University of Defense Technology, Changsha 410073, China

3 ATR Laboratory, National University of Defense Technology, Changsha 410073, China

Abstract With the increase of system scale, the inherent reliability of supercomputers becomes lower and lower. The cost of fault handling and task recovery increases so rapidly that the reliability issue will soon harm the usability of supercomputers. This issue is referred to as the "reliability wall", which is regarded as a critical problem for current and future supercomputers. To address this problem, we propose an autonomous fault-tolerant system, named Iaso, in MilkyWay-2 system. Iaso introduces the concept of autonomous management in supercomputers. By autonomous management, the computer itself, rather than manpower, takes charge of the fault management work. Iaso automatically manage the whole lifecycle of faults, including fault detection, fault diagnosis, fault isolation, and task recovery. Iaso endows the autonomous features with MilkyWay-2 system, such as self-awareness, self-diagnosis, self-healing, and self-protection. With the help of Iaso, the cost of fault handling in supercomputers reduces from several hours to a few seconds. Iaso greatly improves the usability and reliability of MilkyWay-2 system.

Keywords supercomputer, autonomous management, fault tolerant, fault management, MilkyWay-2 system

Received December 17, 2013; accepted March 6, 2014

E-mail: kailu@nudt.edu.cn

1 Introduction

It is a miracle that the computer science seems not to obey the Murphy's Law: anything which can go wrong will go wrong. We only notice the capability increase of CPU, memory, network, and other peripherals, while computers seem never suffer faults. Nevertheless, Murphy's Law still dominates computer science, when the scale of computer goes large, especially in supercomputers.

A supercomputer is the interconnection of a large amount of compute nodes and service nodes. Though the reliability of each node is high, the overall reliability is low because of the scale accumulation effect. For example, MilkyWay-2 supercomputer consists of compute nodes, communication network and storage array. The system contains 16 000 compute nodes with 32 000 CPU and 48 000 accelerators. For such large scale, faults become frequent events for the whole system. Once faults occur, they not only introduce costly fault diagnosis and repair, but also disturb the running task, which requires additional task recovery cost. The overall system usability is greatly reduced by system faults. Consequently, fault management becomes a new dimension of challenges for supercomputer design, which is also named as "reliability wall" [1].

Traditional system management consists of resource management and state monitoring. Only state monitoring is re-

lated to fault management. However, state monitoring is dealt by periodic check of system administrator. This inevitable introduces delay on fault detection, and time cost on interactive fault diagnosis. The diagnosis process requires backtracking the reported errors and retrieving the lost information about the fault context level by level. This process typically requires from several minutes to many hours. Fault management by manpower is inevitable inefficient and costly.

Many existing system tries to accelerate fault management by automating part of the fault management procedure. Li and Lan introduce a concept of proactive fault management [2], which proactively migrates processes or performs checkpointing based on fault alarm. This approach requires accurate fault prediction for properly issuing the proactive actions, which is over-idealized for current computers. Solaris introduces fault management architecture (FMA) [3] to report and handle hardware faults inside operating system. FMA can event fix some noncritical faults and prevent the kernel panic, but it can only manage fault inside the range of operating system and the inter-system faults are not considered at all. Oliner and Stearley propose to filter the system logs to identify the faults in the supercomputers [4]. However, their work is mainly based on empirical knowledge and is hard to be automated. Sun et al. propose to build fault-aware computing environment by adaptively directing runtime system to tolerant faults [5]. This approach also suffers the false positive of the fault prediction, so that the cost prevents this technique from using in supercomputers. There are also many works [6–13] focusing on the fault management problems of clusters, grids, or datacenters. They all work well in their proposed scenarios, but these systems are tightly coupled with certain system or application. None of them is suit for supercomputer management.

To conquer reliability wall of supercomputers, we propose the concept of autonomous fault management. Autonomous fault management targets automatically reacting hardware faults and providing a relatively fault-free environment for supercomputer applications. Compared with existing autonomic computing systems [14], autonomous fault management focuses more on the faults in the supercomputer itself rather than environmental changes.

We implement an autonomous fault management system, named Iaso. Iaso is the Greek goddess of recuperation from illness. We adopt this name since our system does exactly the same thing on supercomputers. The key of fault management is managing the lifecycle of all faults automatically and minimizing the cost of each stage, including fault detection, fault diagnosis, fault isolation, and task recovery. That is, Iaso makes the supercomputer to be self-managed. Iaso implements the following five autonomous features for fault management:

- Self-awareness: the supercomputer can be aware the execution status and detect faults automatically.
- Self-diagnosis: the supercomputer can analyze the errors and locate the root cause of the errors automatically.
- Self-healing: the supercomputer can isolate the faulty hardware, reconfigure itself, and preserve the usability and efficiency automatically.
- Self-protection: the supercomputer can protect the tasks running on it from the impact of faults automatically.
- Self-evolution: the supercomputer managing capability can be improved with more and more experiences gathered during system managing.

By autonomous management, Iaso greatly reduces the overhead of fault management. Therefore, it increases the usability and reliability of computer systems. In MilkyWay-2 supercomputer, Iaso speeds up the efficiency of fault management by two orders of magnitude.

In addition, Iaso is a general autonomous management frame work that can be extended for a broad range of computers beside supercomputers. We propose self-similar system architecture for Iaso, by which Iaso does not sensitive the scale of target computer system. Moreover, Iaso also provides open interface that allows any developers to insert new function models. That is, Iaso supports dynamically and seamlessly extension. The architecture related and system specialized functions are confined into several models. Hence, the work for the transplantation, extension, or upgrade of Iaso is minimized. Besides this feature, the Iaso can also support evaluative function. The Iaso can get more and smarter based on ever increasing knowledge gathered from normal days.

To summarize, the contributions of this work are as follows:

1) We propose a scalable management framework that can provide autonomous management service for almost all existing large computer systems.
2) We abstract all kernel functionalities into mechanisms and knowledge, so that the system functionality can be evolved smoothly.
3) To the best of our knowledge, our system is the first one for scaling to 50P system that contains thousands

of compute nodes.

4) We implement the whole management system on the MilkyWay-2 system for large-scale test. The results prove the effectiveness of our design.

The rest of this paper is organized as follows. We introduce the challenges of supercomputer management in detail in Section 2. In Section 3, we describe the overview of Iaso, including system architecture and the information flow for fault management. We present the key enabling techniques of Iaso in Section 4, and the deployment on MilkyWay-2 in Section 5. Finally, we conclude the work in Section 6.

2 Challenges of supercomputer management

Supercomputer management is a key factor for the whole system efficiency. Currently, most work depends on human supervision. In contrast, the increased system scale imposes more critical demands on management capabilities, including real-time, flexibility, adaptability, and integration. Unfortunately, human-centric management cannot fulfill most of these requirements.

Real-time indicates the delay of reaction on system failures, including the hardware faults and the software exceptions. In supercomputer, the processes of a single application are always tightly coupled. If one process is blocked or slowed down by faults, other ongoing processes have to wait for it because of the inter-process synchronization. Hence, any problem in supercomputer introduces global degradation. As a result, the real-time property of the reaction on system failures is crucial for system efficiency. Clearly, human-centric management inevitably introduces much delay on failure discovery and reaction. In contrast, Iaso adopts automatic self-monitoring and failure reaction, thus to greatly increase the real-time performance.

Adaptability is the capability to manage similar system by historical knowledge of current system. Supercomputer management involves many aspects of supercomputer. It is desired to use historical knowledge to reduce the difficulty of managing a similar system. Human-centric management is adaptable for this case. However, it suffers great difficulties in training a new system manager, because the knowledge of supercomputer management is typically gained in a case-by-case way. In contrast, Iaso formalizes the knowledge of system management, so that the knowledge can be easily reused in a similar system.

Flexibility evaluates remaining usability against system failures. Once failure occurs, it usually requires system reconfiguration to isolate the faulty components, so as to preserve the system usability. System reconfiguration is based on the accurate diagnosis of the source of the failures, because the error message or the error phenomenon does not always imply the root cause of the errors. Human-centric management requires interactive error diagnosis and error fix, so that it takes much time to recover the system usability. In contrary, Iaso introduces automatic error diagnosis and error reaction, thus to obtain much more flexibility on system management.

Integration measures the unification of all management aspects. Computer management includes task management, resource management, fault management, and so on. It is desired to manage the entire system through a unified interface. However, fault management typically requires tracking from software to hardware, and the amount of knowledge typically goes beyond the ability of a single human. Consequently, the system administrator always leaves the fault management to the developers. With the help of Iaso, the fault management work is highly automatic, thus requiring very little human participation. Moreover, Iaso can also directly call functions of the resource management and task management subsystems, so that Iaso integrates all management functions into a single system.

In a word, traditional human-centric fault management encounters problems in the capabilities of real-time, flexibility, adaptability, and integration. All of these capabilities then restrict the overall system efficiency. To solve such issues, we introduce the concept of autonomous management in Iaso design, which enables automatically self-management of supercomputers. Iaso can greatly reduce the cost of supercomputer management, so that improves the system usability and efficiency.

3 System overview

3.1 System architecture

The aim of Iaso is not limited to supercomputer management. It can cover a much larger range of computer systems, including mainframe, MPP, CCNUMA, clusters, and even distributed cloud servers. To efficiently support such wide range of computer systems, the design of Iaso achieves high adaptability and configurability.

Iaso consists of three layers, i.e., the basic autonomous management system (BAMS), the core autonomous management system (CAMS), and the global autonomous management system (GAMS), as illustrated in Fig. 1. The BAMS is

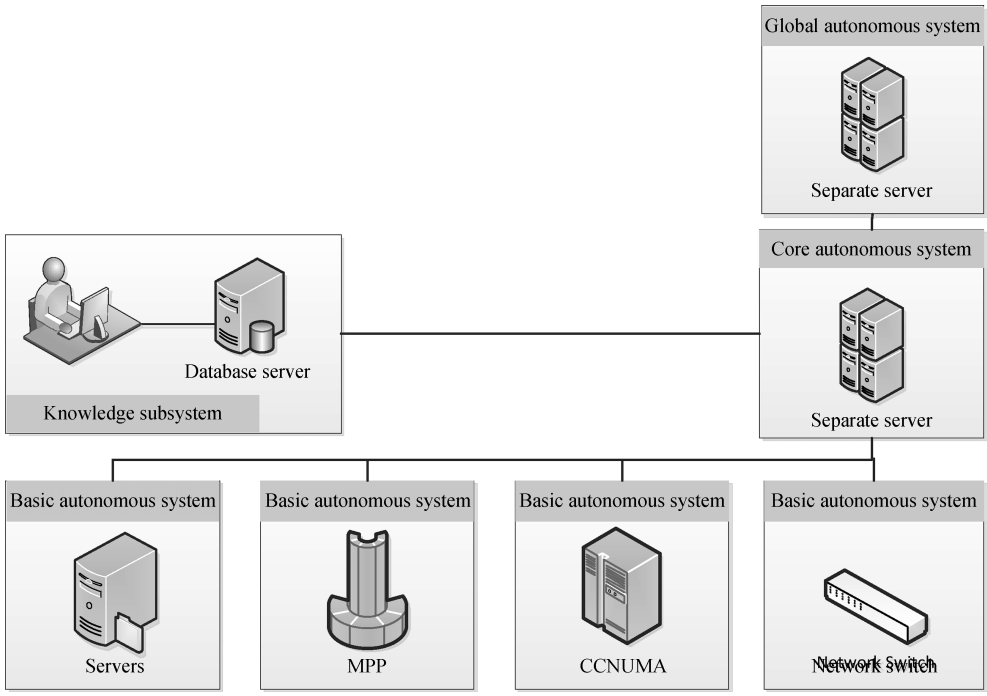


Fig. 1 System architecture

the bottom layer of the whole system. Each BAMS manages an independent function unit of a computer. We classify computer components into intelligent devices and non-intelligent devices. An intelligent device can run program, report its status, and manage itself, such as a compute node. Typically, the management boundary of a BAMS on intelligent devices coincides with the boundary of the operating system. In contrast, non-intelligent devices cannot run program on them, such as network switches. For a non-intelligent device, Iaso utilizes neighboring intelligent devices, which are associated with the non-intelligent device, to report and manage its status. BAMS takes charge of low level fault detection, status report, and local management. It reports faults to higher level, does basic fault diagnosis, and conducts local fault handling or task recovery.

The CAMS manages a group of BAMSes. In large-scale systems, task execution requires the cooperation of many compute nodes and related network devices. CAMS manages the group-level faults. It takes into account the network and the task distribution. All faults reported by BAMSes are gathered by CAMS. It conducts two actions on the fault reports. The first is inter-node diagnosis. Some faults exceeds the boundary of BAMS, thus requires higher level diagnosis and fault handling. The other work for CAMS is system-level analysis. When a fault occurs in the network, CAMS analyzes the influenced nodes, among which the communication is degraded. Besides, CAMS also analyzes the impact of faults on the tasks, and determines whether Iaso should recover tasks. There is a knowledge subsystem connected to CAMS. It provides system-level information, such as the network topology, diagnosis rules. Moreover, the faults are recorded to the database for further query or data mining. Similarly, the GAMS has global view of the system and globally manages all CAMSes.

The self-similar design of each layer greatly increases the adaptability of Iaso. If the target system is the simplest single server, we only need to deploy BAMS and provide basic knowledge to the system. In contrary, if the target system has sophisticate structures, we can accordingly deploy multiple layers of the autonomous management system. In MilkyWay-2, we deploy all three layers because of its huge scale.

3.2 Information flow

Iaso provides autonomous management of faults. It manages the whole lifecycle of system faults. The information flow of Iaso is thus fault-centric, as shown in Fig. 2.

When a fault occurs in the target system, the fault is first detected by the fault detection model of BAMS. Then, the fault detection model reported the fault to basic diagnosis model to analyze the root cause of the fault. If it is successfully diagnosed, the local fault handling model will solve the fault directly. At the same time, the fault is reported to the upper level management system, i.e., CAMS, through in-band or out-band communication. CAMS conducts

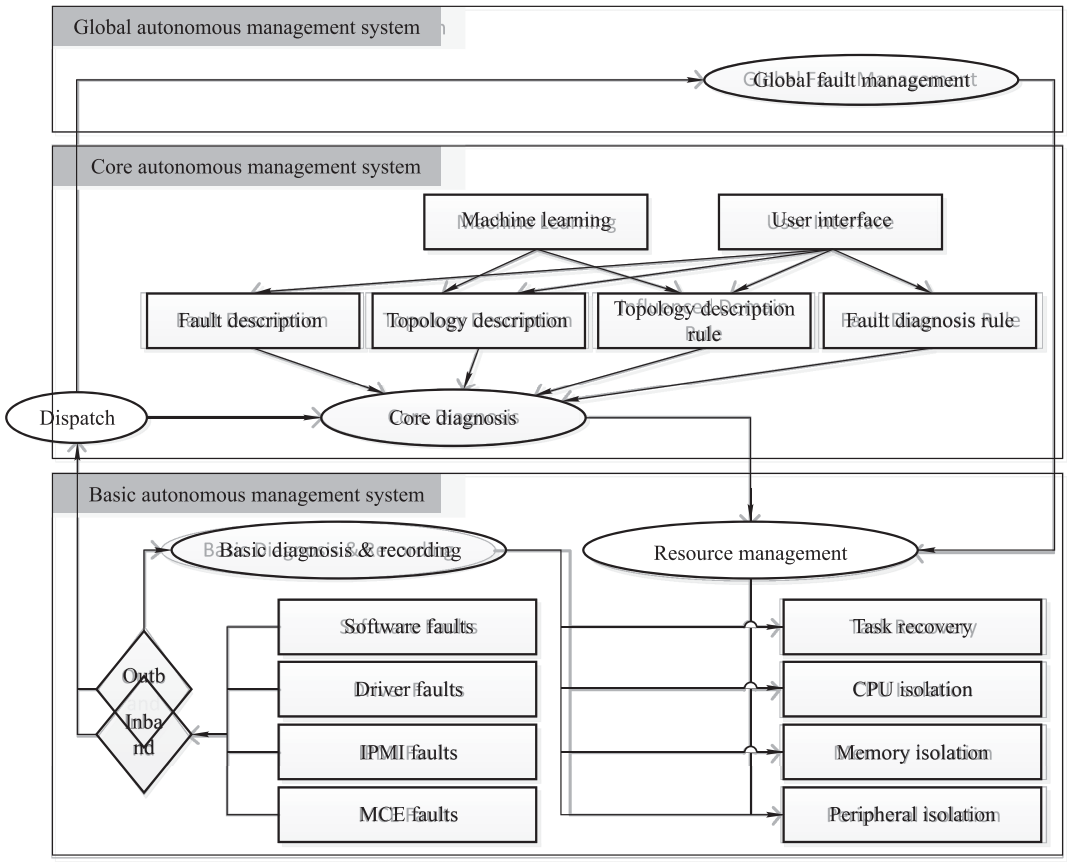


Fig. 2 Information flow

system-level diagnosis and determines whether it handles the fault. CAMS may call basic fault handling model in BAMS to recover or isolate the fault. In addition, CAMS also analyzes the influenced domain of the fault and determines which part of the system is degraded. The degraded resources and the faulty resources are marked in the resource management system. This information guides the future resource allocation. Moreover, CAMS evaluates the tasks running on the faulty or degraded nodes, and performs task recovery if needed. The GAMS does similar work as CAMS under the view of the whole system. Finally, if the fault requires a human fix action, the user terminal will receive a message for the exactly location of the faulty hardware to be replaced. Except the hardware replacement, no further human action is required in all the process. Hence, we call this process as autonomous management.

4 Key technologies

4.1 Instant fault detection

Fault detection is the basis of autonomous fault management. Traditional task management only focuses on task-level faults instead of low-level faults. As low-level faults are probably the root cause of task faults, managing low-level faults implicitly covers task-level faults. Hence, Iaso targets low-level hardware and software faults. On the other hand, a crucial issue on fault detection is the detect delay of faults. Generally, lower delay results in lower diagnosis cost and less fault propagation, thus requires lower repair cost. For this reason, Iaso adopts customized software and hardware to catch low-level faults as early as possible.

4.1.1 OS kernel instrumentation

A typical hardware error may experience three stages before it can be observed by the system administrator. First, a hardware component goes wrong by aging, vibration, static electricity, fluctuation of voltage, interference, and so on. Though this error does exist at this moment, it cannot be caught by software. Then, the faulty hardware is used by the system, and it triggers a failure state, such as ECC or other sanity check mechanisms. The state failure may further be reported to the operating system by interrupt or function call exception. Finally, system administrator receives an error message about the fault, for example a kernel panic or a process crash.

Iaso requires the fault detection be as early as possible. As we have discussed, we cannot detect a fault in the first stage by software, thus the second stage is a proper choice for error detection. Further, if Iaso properly handles the fault in the second stage, administrator will not receive any error reports in the third stage. Since the error handling is done by the computer itself and is transparent for system administrator, Iaso makes the computer fault management autonomous.

Iaso instruments the operating system kernel to catch faults of the target system. For peripherals, Iaso instruments the device drivers to catch the exceptions. For PCIe bus, Iaso adopts AER mechanism to catch faults. For hard disks, Iaso also implements SMART-based fault prediction besides host driver instrumentation. For Intel CPU, Iaso utilizes MCE mechanism to catch faults on CPU, system bus, and memory. Besides, Iaso also reports all software faults, such as out of memory, segment faults.

To facilitate OS kernel instrumentation, we implement a branch coverage tool. The tool is designed by a key observation that most errors in device drivers are reported by status check. Moreover, status check is always expressed as branch instructions in driver code. However, it is complicated for some drivers to manually check the whole status space. Hence, we implement a special static analysis tool to check branch code in device drivers. It can automatically identify the error reporting path, as well as uncovered path. The error reporting path is recognized by the return value analysis and several programming patterns in kernel code, such as "return -EIO", "printk(KERN_ERR ...". Then, the tool reinforces the code on the identified path, which facilitates driver instrumentation.

The instrumentation adopts the concept of aspect-oriented programming (AOP). We take the error reporting as an aspect for OS kernel. As OS kernel is programmed by pure C, it cannot directly support AOP model. To solve this problem, we introduce a pre-compile phase. We first insert instrumentation hints in the OS code, then the pre-compiler turns all the hints to standard C code. By this means, the error reporting function can be configured to the specified implementation. This mechanism greatly increases the flexibility of fault detection.

All faults reported by the instrumentation code are first collected by a kernel message packer. The kernel message packer ensures the message can be proper delivered at any kernel context. Then, the message packer transmits the fault messages to the user-level message dispatcher model by netlink mechanism, as shown in Fig. 3. Finally, the message dispatcher sends the message to proper local handlers. For example, there is a message report model on each BAMS that reports faults to CAMS server.

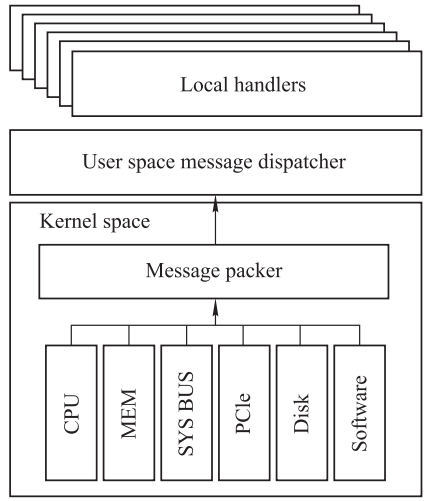


Fig. 3 Message gathering and transmission

4.1.2 Hardware fault detection

Hardware fault detection mechanism catches the first-hand errors from the hardware components. However, as the performance of supercomputers strides over the phase of 10PFlops, fault detection from tens of thousands of hardware components becomes extremely difficult. To conquer this challenge, Iaso implements a hierarchical hardware management architecture. As shown in Fig. 4, this architecture is composed of three tiers. At the bottom tier, hardware components are partitioned into several management clusters, each of which is managed by a system management controller (SMC). At the middle tier, a tree-based monitoring network bridges the SMCs to the management and control center. At the top tier, the management and control center takes charge of policy dispatch and system-level hardware management.

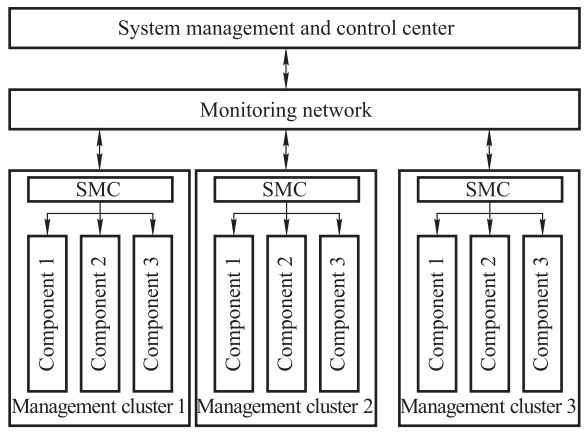


Fig. 4 Hardware management architecture

In each management cluster, diverse hardware compo-

nents, including the computing modules, the communication modules, the power modules, and the cooling modules, are all highly encapsulated as unified targets. All these targets are directly connected to the SMC through various types of the interfaces, such as I2C, JTAG, UART, GPIO, and so on. The SMC jointly applies both periodical scan and instant interrupt approaches to monitor hardware states. To guarantee monitoring efficiency, hardware information gathered in each scan circle is well-tailored according to the top tier policies. It may consist of parts of the following quantities, including various voltages, temperatures and powers of each target, the environment physical measurements, the inner states of the diverse chips, and so on. Fatal hardware errors are allowed to be issued as interruptions, which have higher priorities in SMC. Based on the internal detection algorithms and the external configurations, the SMC picks out the first-hand hardware failures from the regular hardware information and filters the outliers. Whenever faults are detected, the SMC periodically reports the refined hardware information up to the management system and control center via the monitoring network.

The system management and control center visualizes the out-band hardware information, and submit a copy to the autonomous system for further fault diagnosis and prediction. It also takes recovery measures to separately solve parts of the hardware faults. Based on the feedbacks from the bottom tier, the center periodically evaluates the validity of the hardware fault detection mechanism. If needed, it adjusts the policies on the SMCs for more efficient system-level hardware management.

4.2 Message driven model cooperation

Iaso integrates many separate models, each of which accomplishes a partial function of the whole system. The design of Iaso is an open system, which allows dynamically inserting or offloading function models. To support the cooperation between the models and reduce the development cost, Iaso defines a message-driven interface as well as the morphology for the messages.

The message-driven interface has two layers, the inter-system layer and the intra-system layer. The goal of inter-system message-driven interface is to gather fault messages from BAMS to CAMS. The key of this process is reliable message passing. Iaso adopts an adaptive in-band and out-band communication to ensure reliability. In MilkyWay-2 system, there are two communication channels. One is the computing network, which provides high bandwidth and low latency communication. The other is the monitoring network based on our SMC hardware, which provides relatively lower bandwidth and higher latency communication. The preferred communication channel of Iaso is the in-band channel. Nevertheless, if the computing network is congested or broken, Iaso adaptively sends the message through the out-band channel.

The goal of intra-system message-driven interface is to route all message to the proper function model. The basis of this mechanism is a morphology which defines the forms of message. A message in Iaso is a layered tree-structure text. Each layer defines the taxonomy of message type, and the sub-layers define the sub-types. The intra-system message-driven interface implements a message subscribe-dispatch mechanism to send all messages to proper function model. As shown in Figs. 5 and 6, the system structures of BAMS and CAMS are quiet similar. The central model is the message dispatch model. Iaso provides API for function model developers to subscribe the types of message that this function model handles. The message type can be any layer defined by the morphology. When the model is inserted to the system, the message dispatch model will record the message-model subscription relationship. In the runtime, when the message dispatch model receives a message from any model, it will check the subscription relationship and dispatch the message to proper function models. Note that all subscribed model will receive a copy of the message in this procedure.

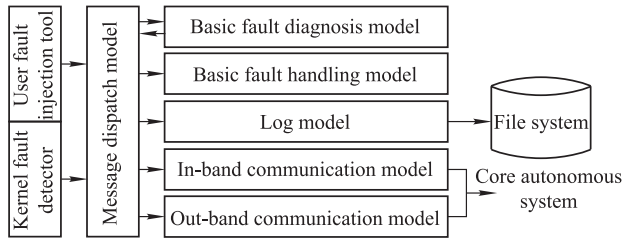


Fig. 5 Basic message subscribe-dispatch

The message subscribe-dispatch mechanism defines a clear interface for all function models. The cooperation of the models is implicitly accomplished by message handling. All models just subscribe the types of messages they handle, and send all messages they created to the message dispatch model. Hence, the development of each single model is all independent and the runtime system can dynamically insert, offload, or update the function models safely and freely.

4.3 Rule-based fault diagnosis

The error reported by status exceptions cannot always

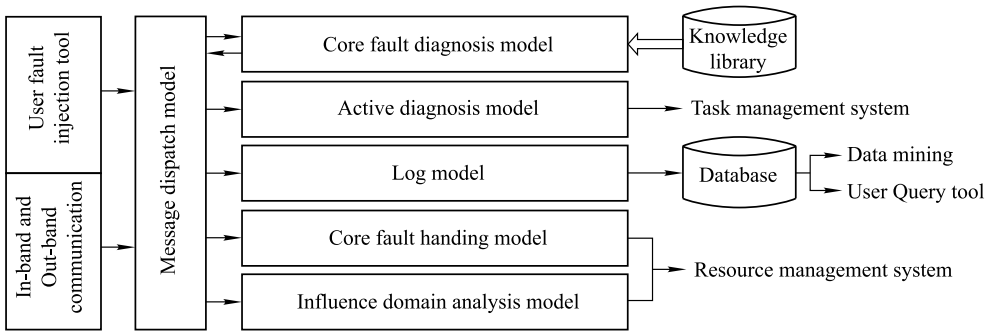


Fig. 6 Core message subscribe-dispatch

identify the root cause of the error. This brings about the difficulties of fault handling and fault isolation. Hence, Iaso proposes rule-based fault diagnosis mechanism, which combines the passive and active diagnosis techniques.

4.3.1 Passive diagnosis

Passive diagnosis model utilizes the error message reported by the fault detection model to determine the real source of the faults. A critical difficulty for passive diagnosis is that the faults and the error messages have complicated relationships. In a real-world system, an error message can be triggered by a set of possible reasons that make the status abnormal. For example, MilkyWay-2 adopts centralized network file system. As a result, a file-read error may be caused by either a disk fault or a network error. Similarly, a fault may create a series of status check failure when the fault propagates in the system. Take a network error as an example. If a router does not work properly, there may be both communication errors and file access errors. Though we receive multiple error messages, the root cause of the errors is the same in this case.

To automatically diagnose the root cause, Iaso models the error propagation as a directed acyclic graph (DAG). The roots of the DAG are all root causes, which are the output of the diagnosis process. The leaves of the DAG are the messages which Iaso can receive by the fault detection model. There are also some middle states which represent some transition events from the root cause to the error messages. Figure 7 shows an example of the error propagation DAG, where three root-causes may trigger four error detectors with four middle states.

The diagnosis process is a reverse searching along the propagation direction. We also use the example shown in Fig. 7. If we receive error message #2, we can reversely go to middle state #1 and #4, and finally get root cause #1 and #3, as marked by the dashed line. Similarly, if we receive error message #4, we can get root cause #2 and #3, as marked by the dotted line. If we receive error message #2 and #4 simultaneously, we use a set *and* operation on the two root cause sets, and the final result is root cause #3. In real world system, the DAG is far more complex than this example. This diagnosis may suffer two major problems. The first problem is that there may be multiple root-cause sets can explain the received error messages, leading to the ambiguity of the diagnosis. Iaso adopts the rule of Occam's Razor to solve this issue. That is, Iaso selects the root-cause set with minimum cardinality. The reason behind this principle is that the possibility of raising multiple faults in the same time is low. Also, this principle leads to minimum fault isolation and fix action, thus leading to minimum disturbance to the system and the tasks. Even if this principle does miss part of root-causes, these missed faults will create further errors and require another round of diagnosis to solve them. The second problem is that the final root-cause set may contain multiple items that may cause the received error messages. This may confuse the error isolation and fault repair. Clearly, passive diagnosis cannot solve this issue by itself, and it requires additional knowledge, for example, by active diagnosis.

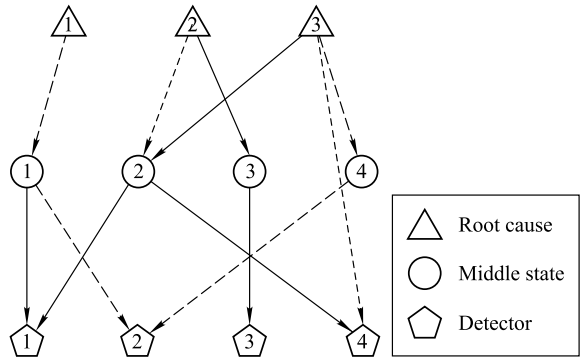


Fig. 7 Passive diagnosis model

By this means, Iaso can automatically diagnose all faults that follow the DAG model. Fortunately, all existing hardware faults fall into this kind, including the CPU, memory,

PCIe bus, and disk. Clearly, there are cases that cannot be described by DAG model, such as dead lock. Nevertheless, such complicated fault reasoning is out of the scope of Iaso.

The syntax of expressing the DAG is a propagation centric way. The DAG expression file is a set of rules, where each rule represents a preparation relationship between a reason event and a result event. Passive diagnosis model loads the rule files, parses the rules, and constructs the DAG for fault diagnosis. The rules represent the diagnosis knowledge for Iaso. Currently, the rules are written by the system administrator based on the knowledge of the system. With the accumulation of knowledge, Iaso becomes smarter and smarter.

4.3.2 Active diagnosis

Passive diagnosis is to diagnose the system faults by the passively received error messages. In contrast, active diagnosis is to diagnose by actively run specialized diagnosis scripts to determine whether the target component is in healthy state.

Active diagnosis consists of a set of test scripts for specified hardware components, such as CPU, memory, network, and peripherals. Clearly, active diagnosis consumes some system resources for test tool execution and result reporting. Hence, the occasion of issuing active diagnosis is restricted.

Active diagnosis mainly has two responsibilities. The first component that issue active diagnosis is the passive diagnosis model. In some special case, we can only get a list of suspicious components by the rule-based passive diagnosis. When the passive diagnosis model cannot determine the real source of error by the knowledge of DAG, it will run active diagnosis against the suspicious components and determine the final result by the execution result. The second occasion of issuing active diagnosis is for regular system state check. Before users run tasks on supercomputers, they always want to ensure that the computer is in healthy state. Hence, a thorough check is preferred before critical task running, and this is done by issuing active diagnosis on the allocated compute nodes. Besides, users want to regularly check computer state on some specified occasions, for example, when the system is ideal, when the compute node is started up, or when the specified day or time is reached.

The syntax for expressing the user specified active diagnosis is also rule-based. Based on the user requirement, the system administrator writes a configuration file by rules that direct Iaso to perform active diagnosis on proper occasion. The rule mainly consists of three fields. The first one is the occasion, such as ideal, a specified time or day. The second filed specifies the test script for active diagnosis with a parameter for the target hardware component. The third filed is an optional script for result check. For some test scripts, it only returns a performance evaluation, while the threshold is related to the expected hardware performance. Iaso uses result check script to sparse the test result and translate the result to a yes/no style result.

4.4 Fault influence domain analysis

Supercomputer is a complex system that requires the cooperation of many subsystems. As a result, an error in one subsystem may influence the function of another subsystem. This phenomenon typically exists in the network subsystem, where the degradation or breakdown of a router may affect the performance of a group of nodes. Iaso introduces fault influence domain analysis technique to infer which part of the system is influenced by the faults.

Fault influence domain analysis mainly focuses on the network subsystem. When some routers or links are broken or degraded, the packages may reroute to bypass the faulty area. Bypassing results in latency increase and bandwidth decrease. This further slows down the computation process of user tasks. The fault influence domain analysis requires two kinds of basic knowledge, the network topology and the route strategy. The route strategy is relatively fixed, falling into several kinds. Nevertheless, the topology varies much for different systems. Iaso adopts a XML-based syntax to express network topology. XML provides layered syntax which perfectly matches the layered structure of supercomputers. The system manager writes proper XML file to express the neighboring relationship of the routers and the nodes. Iaso parses the XML file and constructs the network topology for fault influence domain analysis.

When a fault is diagnosed as a network fault, this fault is dispatched to fault influence domain analysis model. The model infers the node set, among which the communication is degraded. Further the model also evaluates the level of degradation and marks it to three grades, that is, fatal, severe, and moderate. This information is reported to the resource management system. The resource management system will allocate healthy nodes in prior, then use less degraded nodes. Moreover, Iaso also evaluates the slowdown of the running tasks on the influenced node set. If the slowdown reaches certain level, Iaso needs to recover the whole task on a healthy node set to obtain better performance.

4.5 Task-based fault handling

When Iaso receives a fatal fault, it has to handle the fault to

minimize the impact of the fault and recover the running tasks on the faulty node. Fault handling has two major components, fault isolation and task recovery.

4.5.1 Fault isolation

The aim of fault isolation is to prevent to run tasks on the faulty or degraded nodes. In MilkyWay-2 system, there is a resource management subsystem to allocate system resources. Iaso integrates the resource management system to maintain the available resources.

The state-machine of a node is shown in Fig. 8. Initially, all nodes are at the state *power off*. After the power supply management system powers on this node, its state transfers into *initialized*. Before we run tasks on a set of compute nodes, we typically issue an active diagnosis on the target node set. If a node passes the active diagnosis, its state is set to *healthy*. Then, we can run user tasks on these healthy nodes. If a node locates in an influenced domain of a fault, Iaso marks its state into *degraded*. Iaso also scores the *degraded* state into several grades. When we run user tasks, we first select healthy nodes then use the nodes that less degraded. When the source of the influenced domain is repaired, the state of the degraded node is transferred back to *healthy*. When a node is in *initialized*, *healthy*, or *degraded* state and a fatal error occurs in this node, Iaso changes the state of the node into *faulty*. Once in the *faulty* state, the node is drained from running user tasks. The graphic terminal of Iaso reports this error, as well as the precise location of the faulty hardware component. The system administrator can then fix the fault by replacing the hardware. This procedure generally requires the node to be powered off, thus the node come back to the initial *power off* state.

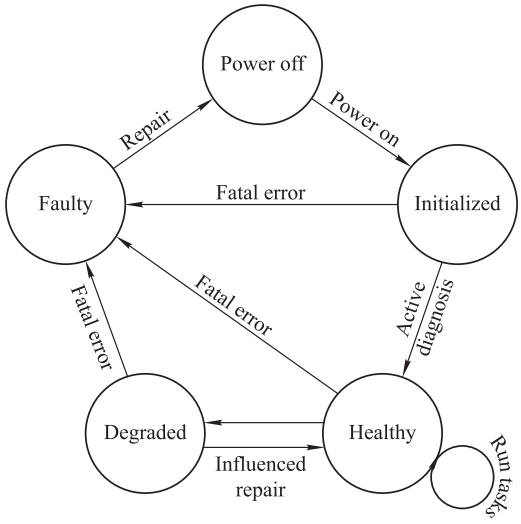


Fig. 8 State transition

Note that all the aforementioned process is all automatic, except the hardware replacing action. Even for this action, Iaso also provide directive information about the exact location of the faulty hardware. Hence, Iaso makes the supercomputer management autonomous, and greatly reduces the overhead of fault management.

4.5.2 Fast task recovery

Iaso provides a fault-tolerance runtime which makes the program actively tolerate the system fault at low cost. The runtime provides a checkpointing mechanism with a simple programming interface, by which applications can achieve sustainable and efficient performance. By checkpointing mechanism, Iaso can recover user tasks from any fatal faults.

Compared to the parallel file system, memory access is with the properties of high bandwidth, low latency. Using memory to save checkpoint file can significantly improve the read and write speed. The problem for memory checkpointing is that it is volatile. Once the node was failed, the checkpoint file would be lost. The runtime uses an in-memory double checkpointing technique, i.e., each node has its own checkpoint copy in local memory, and in addition saves the copy of the checkpoint on a partner node. When a node fails, the image of the node can be recovered from the checkpoint copy of the partner node, so that the checkpoint backup and redundancy backup between each other overcome the volatile memory problems.

The two mutual-backed nodes may also fail simultaneously, and as a result checkpoint data cannot be recovered. Therefore, we should use appropriate strategies to pick partner node avoid this from happening. On MilkyWay-2 system, two nodes on the same board, multiple boards on the same frame, are sharing communication, backboard, power supply and other hardware resources, so that the probability of simultaneous failure is higher than the nodes/boards on different boards/frames. We therefore provide a partner node selection method based on stride of MPI rank of the process. The user can specify *stride* parameter while loading jobs. Each node selects a partner node, and the MPI rank interval between two nodes is equal to the stride. The method can reduce the probability of simultaneous failure of the mutual-backed nodes, and consequently improve fault tolerance by carefully selecting stride parameters.

In-memory double checkpointing requires twice the memory space as big as the checkpoint file. Because each computing node of MilkyWay-2 is installed with 64 GB of memory, the memory is adequate for most application with the check-

point enabled. For some applications that require large memory, we design and implement the XOR checkpoint mechanism to reduce the required memory usage. Nodes involved in the calculation are assigned to XOR sets or disjoint sets, and the checkpoint files in a XOR set is XOR-computed and split into many segments. Besides one full checkpoint file in local memory, each node includes one segment. If a node fails, its lost file can be reconstructed using the XOR parity segments. If a XOR node set includes N nodes, then the memory overhead is reduced from the size of two checkpoint files to $1 + 1/(N - 1)$ the size of a checkpoint file.

When the runtime tries to restart a job, it checks most recent checkpoint files in memory at runtime, automatically analyzes whether they constitute a complete set of checkpoints for job recovery. If they pass the check, fault-tolerant runtime restarts the job and rebuilds the redundancy data according to the new MPI rank. Otherwise, it will delete the checkpoint set, and attempt to recover from the next most recent checkpoint set. Fault-tolerant runtime will continue the process until the job is successfully restarted.

5 Deployment on MilkyWay-2

Iaso is fully deployed on MilkyWay-2 supercomputer. We deploy a BAMS on each node, including the customized operating system kernel and the application-level service models. As the MilkyWay-2 system consists of computing subsystem, storage subsystem, and service subsystem, we accordingly deploy three CAMSes to manage each subsystem. Overall, we deploy a GAMS to collect all messages for the whole system.

The graphical interface of Iaso is shown in Fig. 9. This interface is mainly for monitoring and exhibition of errors. It consists of five windows to show different information of Iaso. The upper left window shows the status of the whole system in graphic mode. The graph exactly shows the physical distribution of the system, including the cabinets and the boards. The gray racks are not powered on. The colors on the boards denote the state of the board, where green, yellow, orange, and red represents healthy, degraded, severely degraded, and broken, respectively. As we cannot show all six rows of the cabinets in a single window, we split them into two windows and display them by turns. The lower left window shows the most recent errors in the system. The lower middle window lists the most recent influenced domain of faults. The upper right window displays the curve of error amount by time. The lower right window shows the raw error messages. By this interface, system administrator can easily get all errors in real time mode.

Besides the graphical interface, we also develop a set of interactive tools to manage Iaso. The tool set consists of fault

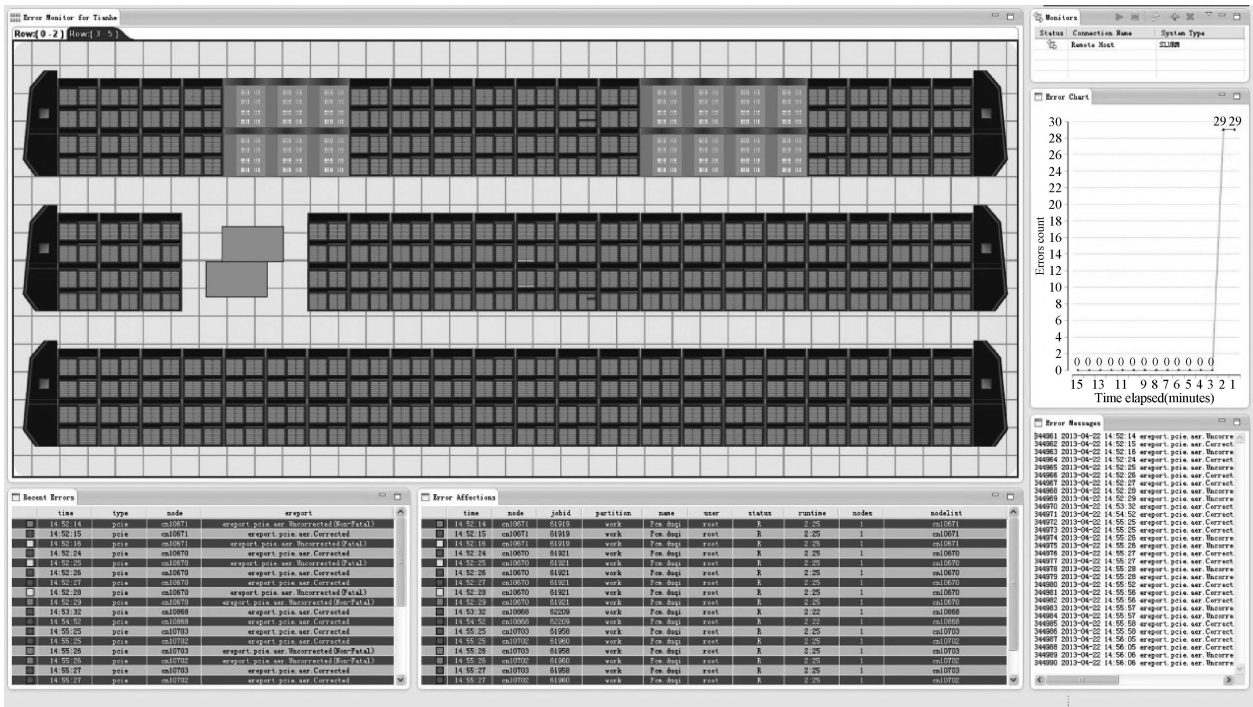


Fig. 9 Graphical interface

query tool, fault injection tool, model management tool, and system deployment tool. All of the tools facilitate the use of Iaso. In addition, these tools support the redevelopment of Iaso to build more intelligent systems.

6 Conclusion

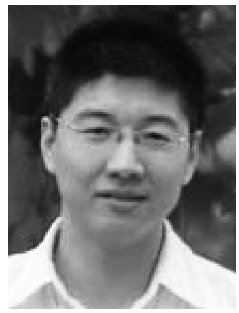
In this paper, we introduce Iaso, an autonomous fault management system in MilkyWay-2. Iaso introduces the concept of autonomous management on system faults, and provides the whole lifecycle management automatically. By Iaso, supercomputers can get several autonomous features, such as self-awareness, self-diagnosis, self-healing, and self-protection. Iaso greatly improve the management performance in the features of real-time, flexibility, adaptability, and integration. With the help of Iaso, the cost of fault handling in supercomputers reduces from several hours to a few seconds. As a result, Iaso greatly increases the reliability and usability of MilkyWay-2 system.

Acknowledgements This work was partially supported by National High-tech R&D Program of China (863 Program) (2012AA01A301, 2012AA010901), by program for New Century Excellent Talents in University and by National Science Foundation of China (Grant Nos. 61272142, 61103082, 61170261, and 61103193).

References

1. Yang X, Wang Z, Xue J, Zhou Y. The reliability wall for exascale supercomputing. IEEE Transactions on Computers, 2012, 61(6): 767–779
2. Li Y, Lan Z. Proactive fault manager for high performance computing. In: Proceedings of the 35th International Conference on Dependable Systems and Networks (Fast Abstract). 2005
3. Shapiro M W. Self-healing in modern operating systems. Queue, 2004, 2(9): 66–75
4. Oliner A, Stearley J. What supercomputers say: A study of five system logs. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 2007, 575–584
5. Sun X H, Lan Z, Li Y, Jin H, Zheng Z. Towards a faultaware computing environment. HAPCW, Mar, 2008
6. Lan Z, Li Y, Gujrati P, Zheng Z, Thakur R, White J. A fault diagnosis and prognosis service for teragrid clusters. Proceedings of TeraGrid, 2007
7. Wang X, Luo J, Liu Y, Li S, Dong D. Component-based localization in sparse wireless networks. IEEE/ACM Transactions on Networking (ToN), 2011, 19(2): 540–548
8. Takemiya H, Tanaka Y, Sekiguchi S, Ogata S, Kalia R K, Nakano A, Vashishta P. Sustainable adaptive grid supercomputing: multiscale simulation of semiconductor processing across the pacific. In: Proceedings of the ACM/IEEE SuperComputing. 2006, 23–23
9. Wang X, Liu Y, Yang Z, Lu K, Luo J. OFA: an optimistic approach to conquer flip ambiguity in network localization. Computer Networks, 2013, 57(6): 1529–1544
10. Santos d T, Santos d L, Farinon F, Homma R, Andrade d R, Khairalla I, Lemos F. Integrating heterogenous applications in control centers based on smart grid concepts. In: Proceedings of the 2013 IEEE PES Conference on Innovative Smart Grid Technologies Latin America (ISGT LA). 2013, 1–6
11. Wang X, Yang Z, Luo J, Shen C. Beyond rigidity: obtain localisability with noisy ranging measurement. International Journal of Ad Hoc and Ubiquitous Computing, 2011, 8(1): 114–124
12. Valverde L, Rosa F, Bordons C. Design, planning and management of a hydrogen-based microgrid. IEEE Transactions on Industrial Informatics, 2013, 9(3): 1398–1404
13. Zhang X, Zhou F, Zhu X, Sun H, Perrig A, Vasilakos A V, Guan H. DFL: Secure and practical fault localization for datacenter networks. IEEE/ACM Transactions on Networking, 2013
14. Huebscher M C, McCann J A. A survey of autonomic computing—degrees, models, and applications. ACM Computing Surveys (CSUR), 2008, 40(3): 7:1–7:28

Kai Lu, PhD, professor, Deputy dean of College of Computer Science, National University of Defense Technology, China. His research interests include parallel and distributed system software, operating system, parallel tool suites and fault-tolerant computing technology.



Xiaoping Wang, PhD, assistant professor, National University of Defense Technology, China. His research interests include parallel computing, system software, and distributed computing.

Gen Li, PhD, assistant professor, National University of Defense Technology, China. His research interests include operating system and system security.

Ruibo Wang, PhD, assistant professor, National University of Defense Technology, China. His research interests include operating system and transactional memory.

Wanqing Chi, associate professor, National University of Defense Technology, China. His research interests include operating system, system software, and high performance computing.

Yongpeng Liu, PhD, assistant professor, National University of Defense Technology, China. His research interests include power management, fault tolerance, and high performance computing.

Hongwei Tang, assistant professor, National University of Defense Technology, China. His research interests include operating system, system firmware, high performance computing, distributed computing and computer architecture.

Hua Feng, associate professor, National University of Defense Technology, China. His research interests include operating system, system software, and computer architecture.

Yinghui Gao, PhD, associated professor, National University of Defense Technology, China. Her research interests include artificial intelligent, machine learning and pattern recognition.