Fault Detection and Diagnosis Software of LHAASO

Hangchang Zhang, Minhao Gu, and Shaoshuai Fan

***Abstract*— The Large High Altitude Air Shower Observatory (LHAASO) is a mega-scale dual-task facility designed to study cosmic rays and γ-rays. Online computing system of LHAASO supports its online operation and computation. Physical phenomena such as cosmic rays occur unpredictably and therefore require the online computing system to run uninterruptedly. LHAASO is large and the environment is harsh, so the online computing system is subject to failure. Once a system fails, maintenance personnel are required to quickly analyze the cause of the failure and repair it. The fault detection and diagnosis software (FADD) is designed to quickly detect and analyze system faults. The software implements comprehensive monitoring of each component of LHAASO's online computing system (computing nodes, switches, and data flow software) and collects real-time status information. When a fault occurs, FADD can quickly analyze the cause of the fault and provide alarm information to the on-call staff as soon as possible. In addition, it can also analyze historical data within a specified period and generate data reports as needed. FADD is designed to fully consider the characteristics of large-scale high-energy physics experiments and satisfy the requirements of high throughput and high efficiency by using a distributed architecture. The software consists of the following layers: information collection layer, data analysis layer, and result layer, and contains metrics detection software, fault monitoring module, fault diagnosis module, and other functional modules. FADD has been applied to LHAASO and can diagnose operational faults quickly and accurately, helping to reduce the burden on maintenance personnel.**

***Index Terms*— Fault detection, fault diagnosis, Large High Altitude Air Shower Observatory (LHAASO), monitoring data, root cause.**

I. Introduction

THE Large High Altitude Air Shower Observatory (LHAASO) is a high-energy physics experiment that aims to investigate the origins of high-energy cosmic rays, as well as related research on cosmic evolution, high-energy astrophysical phenomena, and dark matter [1]. It is installed at 4410-m above sea level in Sichuan Province, China [2]. The online computing system for such experiments consists of a large distributed computing cluster. LHAASO's online computing system hardware consists of 170 computing nodes and 400 network switches. During operation, around 200 processes need to run simultaneously to achieve real-time data processing at a rate of 3 GB/s.

One of the physical objectives of LHAASO is to detect unpredictable phenomena such as gamma-ray bursts and cosmic rays, which have short detection windows [3]. This imposes high stability requirements on the online system. However, hardware or software failures in the online system can lead to abnormal termination of experiments in practical operation. When such terminations occur, the system generates a large number of fault phenomena and logs, which often require maintenance personnel to manually inspect the logs of various subsystems to determine the cause of the fault and repair it based on their experience. In some cases, manual diagnosis can be time-consuming, leading to long experiment downtime. Therefore, LHAASO needs a real-time fault monitoring and diagnostic system to monitor the operational status of the system and promptly notify maintenance personnel of any faults that occur.

The typical structure of this kind of a high-energy physics experiment is illustrated in Fig. 1. The detector system converts particle signals into electrical signals, which are then transformed into digital signals by the electronics system and sent to the online computing system in the form of data packets [4]. The online computing system is responsible for data acquisition, online processing, and storage. It sends valuable data to the offline system for preservation. LHAASO's online computing system is a large-scale distributed software and hardware system. It is composed of data flow software, computing nodes, and switches. The system monitors various types of data generated by different subsystems and conducts unified fault diagnosis, which poses a significant challenge. In high-energy physics experiments, the data flow software operates as a stream processing system, where data are processed in time slices across various software components. It is important to consider the mutual influence between software system components when analyzing the causes of errors, as an anomaly in one component can lead to abnormalities or errors in the entire data flow.

Locating faults is based on comprehensive monitoring of the online computing system, especially the state of the data flow software. The software accurately detects the data flow status by collecting the length of time that each data slice is being processed by the data flow software. Node status and switch status are monitored through parameter collectors distributed at each node.

Manuscript received 19 May 2024; revised 18 July 2024 and 29 August 2024; accepted 31 August 2024. Date of publication 5 September 2024; date of current version 17 March 2025. This work was supported by the State Key Laboratory of Particle Detection and Electronics under Grant SKLPDE-ZZ202414. *(Corresponding author: Minhao Gu.)*

The authors are with the Institute of High Energy Physics, CAS, Beijing 100049, China, also with the School of Nuclear Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China, and also with the State Key Laboratory of Particle Detection and Electronics, Beijing 100049, China (e-mail: zhanghc@ihep.ac.cn; gumh@ihep.ac.cn; fanss@ihep.ac.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2024.3454806.

Digital Object Identifier 10.1109/TNS.2024.3454806

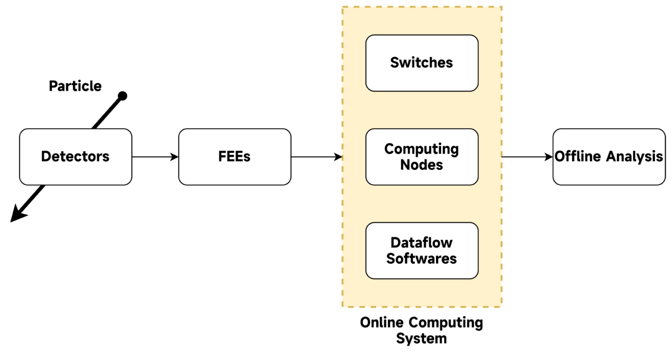


Fig. 1. Structure of the typical high-energy physics experiment.

The control and monitoring system of LHAASO implemented the monitoring of data streaming software, web page display, and basic data flow software's auto-recovery function by collecting information from Redis and Kafka [5]. We found that the failures of online computing systems include both hardware and software, and there are obvious dependencies between the failures, so we decided to develop fault detection and diagnosis software (FADD). From the beginning of the design, FADD was positioned separately from other monitoring software. The purpose of FADD is to discover the root causes of failures based on the dependencies between the faults when the online computing system (not only the data flow software) fails. Thus, the core of FADD is fault diagnosis and fault dependencies.

Currently, there are several fault analysis methods for such systems, including analyzing metrics [6], mining log files [7], [8], and tracking system executions [9]. This article presents a software solution that includes monitoring metrics, fault detection, and fault root cause analysis. To analyze the root causes of faults, we propose a data structure based on a directed acyclic graph, named fault directed acyclic graph (Fault DAG).

The main contributions of this article are as follows.

1) Comprehensive monitoring of the software and hardware implementation of the online computing system of LHAASO was conducted.
2) An algorithm was implemented to analyze the fundamental causes of faults based on Fault DAG dependencies, which is a data structure for analyzing fault causes based on a directed acyclic graph.
3) The software has been applied at LHAASO, providing assistance in experimental operations.

II. Design of the Software

The FADD collects data metrics from the online system's software and hardware, analyses them, identifies faults, and determines their real cause based on causal relationships. The software consists of several layers, including collect layer, analysis layer, and result layer, as shown in Fig. 2.

The system architecture needs to consider the key points as follows.

1) Comprehensive status data collection for the online computing system instead of collecting logs. Unlike other monitoring software, FADD does not collect text logs. We believe that the amount of data in text logs is too large for subsequent analysis, and most of the logs or error reporting information is also obtained by analyzing the changes in the data, so a comprehensive collection of data can be used instead of logs.
2) Small granularity monitoring of data flow software. Data flow software is the software implementation of online computing and the core of the online computing system. We hope to realize the performance monitoring of each component within the data flow software and also follow up the study of data flow performance fluctuation caused by other parts of the online computing system.
3) High throughput information collection and preservation. The scale of monitoring data can reach 100 000 entries per second, with the annual storage exceeding the TB level. The software needs to implement collection, process, and preservation of data. From the administrative management of the experimental point of view, the permanent preservation and regular trace-back of the collected operational parameters are very important.

The collect layer collects operational metrics from various components of the online computing system and consists of a distributed information collection program, Monitor Agent, and a management program, Monitor Master. By deploying Monitor Agent in nodes and switches, it collects operational metrics and statuses at regular intervals, and the collected data are sent and saved to the database through Kafka, a messaging middleware [10]. The management program manages the collection process distributed in each node through HyperText Transfer Protocol (HTTP). State collection of the data flow software is achieved by adding probes to the software and sending the timestamps of the data slices being processed at different stages through Kafka.

The analysis layer provides the determination of faults and root cause analysis of faults, which mainly consists of the fault detection module and the root causes analysis module. The data collected from the information collection layer into the database will go through the fault detection module to determine whether the parameters are within a reasonable range. When the online computing system suddenly fails to operate because of a component problem, a large number of associated fault phenomena and alarms will be generated, and the root causes analysis module will utilize automated methods to find the root cause of the fault with the reference of the expert database.

The result layer is responsible for human–computer interaction with the user, and its main functions include generating alarms by various means when a fault occurs and a web-based front-end interaction page for operation and maintenance personnel.

III. Implementation of the Software

A. Information Collection

The state information to be collected by the software includes: CPU utilization and memory utilization occupied by the online computing software, CPU temperature of each computing device in the online computing system, switch port

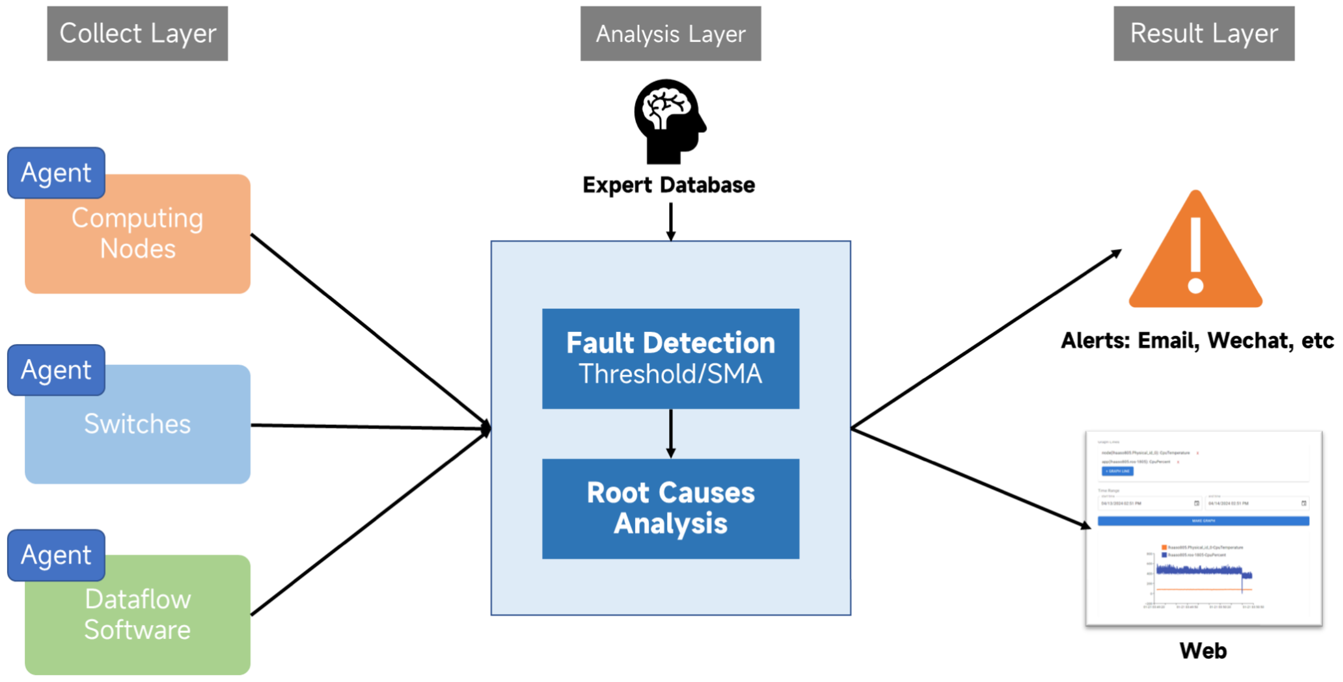


Fig. 2. Structure of FADD.

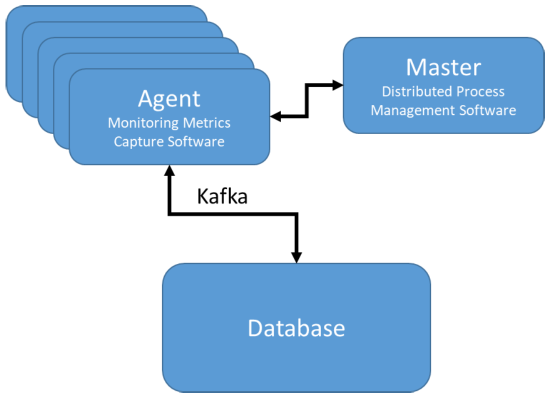

Agent
Monitoring Metrics Capture Software
Master
Distributed Process Management Software
Kafka
Database

Fig. 3. Implementation of fault detection module.

ON/OFF status, and performance (throughput rate and processing latency) of the data flow software.

Metrics other than software performance are acquired by a collection program called Monitor Agent. The Monitor Agent is responsible for collecting information on node metrics and switch metrics and sends the information periodically. Through the modularization of monitoring metrics, the Monitor Agent collects information including processes, disks, CPUs, memory, and simple network management protocol (SNMP). The Monitor Agent is deployed on various nodes that need monitoring. Monitor Master is the management program for Monitor Agent, responsible for the deployment of agents. The relationship between the Monitor Agent and the Monitor Master is shown in Fig. 3.

How to collect and analyze the software performance of online computing software is a key point of FADD, and the implementation of this part can be seen in Fig. 4. LHAASO's online computing software is a stream processing, which includes data readout, assembly, processing, and storage. A performance change in one component may cause anomalies or errors in the whole flow. Data are read from the electronics and segmented into time fragments (TFs) with unique time fragment IDs (TFIDs) based on absolute timestamps. TFIDs remain constant during processing. For such a stream processing system, performance is evaluated using two metrics: latency and throughput. Latency is the time elapsed from the start of a TF to the completion of processing in a component. Throughput is the number of TFs processed by a component per second. Latency and throughput cannot be obtained directly by FADD, but need to be counted and calculated.

By adding probe code to the software, the software sends all the TFIDs and timestamps to FADD's message structure via Kafka after processing the data slices in different stages, so that the timestamps of the data slices in each stage are obtained. By parsing the timestamps of the same TFIDs, the time consumed by the data slice in different phases, i.e., the latency of different phases, is obtained. The throughput of the software is calculated by analyzing the number of data slices processed per second in each stage.

The detection time interval also needs to be discussed. If the interval is too short, the amount of data will surge and bring in a lot of interfering information, making subsequent data analysis more difficult. If the interval is too long, it will result in some fault information not being detected. After the study, the collected parameter information and configuration information are shown in Table I.

As discussed in Section II, the software has to collect data at a high rate and also for archiving. For that purpose, we have implemented a three-level message structure based on the messaging middleware Kafka, as shown in Fig. 5. The first level is the message directly from the data flow software. Because of the high data rate, it cannot be saved to the

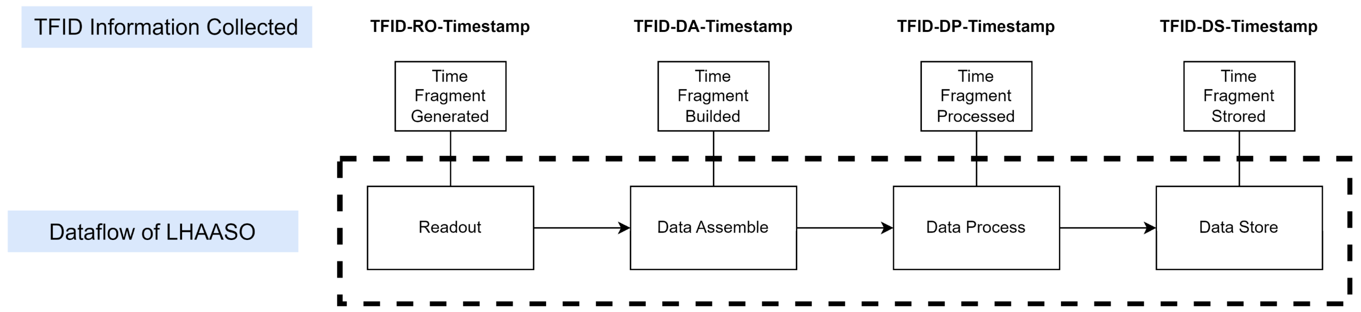


Fig. 4. Collecting TFID information from dataflow of LHAASO.

TABLE I
TIME INTERVALS SETTINGS OF METRICS

Metrics	Source	Time Interval (second)
CPU utilization	Software	5
Memory utilization	Software	5
CPU temperature	Computing Node	5
Switch Port Connection	Switch	5
Throughput rate	Software	-
Processing latency	Software	-

database in real time, so the second-level message structure is responsible for aggregating the data from the first level in 1-s units. In addition, node parameters and switch parameters can be sent directly to the second level because the message rate is not high. All the messages of the second level will be saved in the database. The fault detection module will find the exceptions and faults from the database and save them in a new table for easy interaction with the user. This part of the analysis will be done at the third level.

FADD uses Cassandra, a distributed NoSQL database management system, as a permanent data store [12]. Cassandra can manage and manipulate large amounts of data, as well as being highly available and fault-tolerant. FADD deploys Cassandra using three nodes, which is guaranteed to work if one node fails. MongoDB was chosen for the temporary database because of its variable table structure and simple user interface that makes it easy to use.

B. Fault Detection

The software categorizes outliers that deviate from the normal range into two categories: exceptions and faults. Exceptions are data that are out of the standard range but the data flow software can still consistently produce data that is usable for physical analysis. Faults, on the other hand, are data that are out of the standard range and cause the online computing system to operate abnormally or even interrupt. Exceptions have the potential to evolve into faults. Exceptions and faults are detected by the fault detection module, but the discussion in this article focuses on faults. In order to determine faults in time, it is necessary to establish evaluation judgment criteria. By choosing different evaluation criteria for different hardware or software, a systematic evaluation system can be formed.

A threshold-based approach is utilized to monitor the temperature of the CPU and the status of the switch. Upper and lower limits are set for the parameters, and when the parameters exceed the preset thresholds, they are considered abnormal. The accuracy of this method depends on the reasonable setting of the thresholds.

To effectively identify and evaluate the fluctuation of parameters, we use a simple moving average (SMA) analysis method [11]. The formula for calculating the simple moving average is given as follows:

$$\text{SMA} = \frac{\sum_{i=n-L}^{n} X_i}{L}. \tag{1}$$

The parameter value at the ith time point is represented by X_i. The window size of the moving average, which is the number of time points considered, is denoted by L, and n represents the current time point. The simple moving average can smooth short-term fluctuations, reflecting the average utilization over a period of time. However, it can also be used to calculate the ratio of the current value to the previous window's SMA to identify exceptional fluctuations. We use the SMA method to evaluate CPU utilization, memory utilization, and software performance.

In operation, some details of fault detection have to be considered. The first is to avoid false alarms, which is why we are saving the data in the database to consider the whole rather than using stream processing to make real-time judgments. We found that some metrics will occasionally produce data out-of-standard range, but in fact, it may be a collection error rather than a real error report. In the real application, only continuous out-of-standard value will be considered as fault. Second, we use stream processing to judge whether the data flow software is working properly in real time. The data flow is considered to have stopped working when no messages from it are received by FADD within 1 s. The algorithm is simple, consumes little computational resources, and works efficiently and quickly. Immediately after determining that the data flow has stopped, a fault diagnosis is initiated to reduce the time spent on fault analysis.

C. Fault Diagnosis

When the online computing software fails to operate normally, it is crucial to rapidly and accurately analyze the root cause of the fault. The root causes analysis module in the analysis layer of FADD has designed and implemented a fault analysis method based on a directed acyclic graph, called Fault DAG, aimed at locating the root cause of faults.

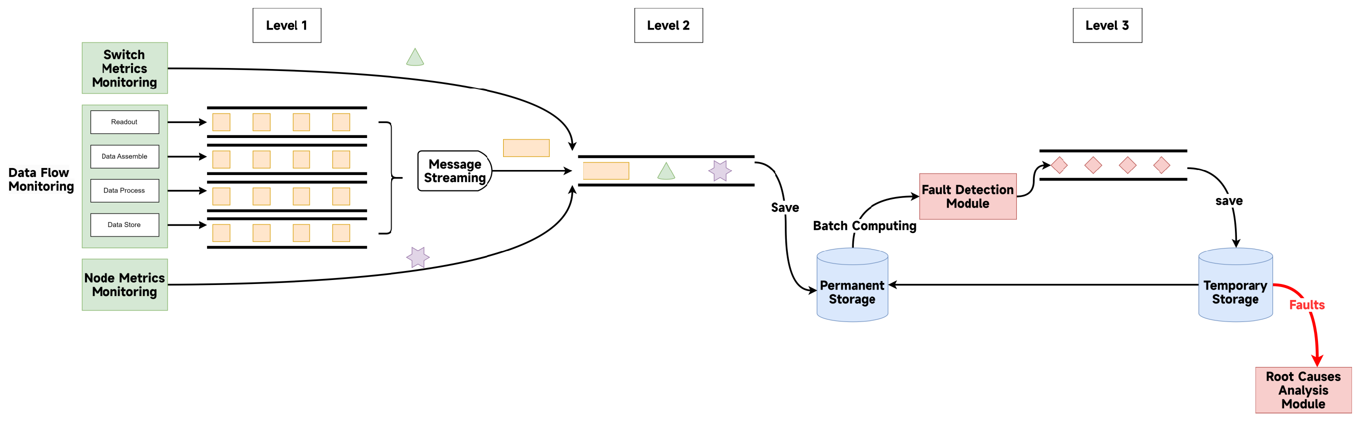


Fig. 5. Three-level message design.

The analysis of faults can have different scopes. If the scope is too fine, for example, a specific computing node's CPU temperature is too high, it can be challenging for operations personnel to determine what caused the high CPU temperature. If the scope is too coarse, it may not provide enough information for operations personnel to recover from the fault. FADD, informed by LHAASO's operations, targets the analysis on the nexus of component failure and fault type, clearly delineating the component and fault for personnel.

The root causes analysis module acts as an expert system, consisting of system structure graph, fault dependence graph, and Fault DAG, as shown in Fig. 6. When a fault occurs, the fault detection module identifies the failure component and failure phenomenon, combining this information with the system structure graph to form the Fault DAG. By algorithmically analyzing the fault dependence graph designed based on operational experts' experience, the root cause of the fault is determined and informed to operations personnel. Operations personnel use the system notification as a reference for fault resolution.

An online computing system contains a large number of various types of components, such as computing nodes, switches, and software processes. These components have complex dependence relationships with each other, which mainly contain calling relationships and deployment relationships. The calling relationship is mainly between software, and in LHAASO, it mainly refers to the process relationship during the data flow process. Deployment relationship refers to the connection between hardware and hardware or between hardware and software, for example, a computing node is connected to a port of a switch, or a process is deployed on a certain computing node. The system structure graph is constructed based on the deployment relationship and calling relationship of the online computing system, and the algorithm for constructing the system structure graph is as follows.

1) Form three sets of computing nodes, switches, and software processes, denoted as N, S, and P, respectively.
2) Define a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edges.
   a) $V = P \cup N \cup S$, which means V includes all software processes, computing nodes, and switches.
   b) E is a subset that contains directed edges of the form (N_j, P_i) and (S_k, N_x).
3) The rules for adding directed edges are as follows.
   a) For all $P_i \in P$, $N_j \in N$, if software P_i runs on computing node N_j, then $(N_j, P_i) \in E$.
   b) For all $N_x \in N$, $S_y \in S$, if there is a network cable connecting computing node N_x to switch S_y, then $(S_y, N_x) \in E$.

During the running of the online computing system, the Fault DAG is formed by combining the anomalies with the system structure graph. The process of generating the Fault DAG is as follows: there is a system structure graph $S = (V, E)$, with V denoted as the set of nodes and E denoted as the set of directed edges. If there is a fault phenomenon, A occurs in component $B \in V$ and the event A occurs at time t. Define the Fault DAG $F = (V', E')$, where $V' = V \cup A$ and $E' = E \cup (A, B, t)$. That is, a new node A and a directed edge pointing from A to B with weight t are added to the graph S. The process of adding nodes and edges cuts off when the online computing system stops working.

The fault dependence graph is compiled from the duty logs of the LHAASO online computing system over the last few years and from the experience of operation and maintenance specialists. The nodes of the fault dependence graph consist of system components and the corresponding fault phenomena of the system components, and the directed edges are the reflection of the fault dependence relationships. For instance, a case that frequently causes system operational anomalies is a computing node responsible for data processing with a high CPU temperature (the deeper cause being the failure of the rack cooling system in which the computing node is located, but this is not considered in FADD). Based on this, the fault dependence graph for this case is designed as shown in Fig. 7, which reflects the failure dependence that causes the online computing system to cease operation. In implementation, FADD store graph data by saving node and edge information in the database.

When a system failure occurs, the system combines the real-time generated fault phenomena with the fault nodes to construct a real-time Fault DAG until the online computing system fails to work. Subsequently, this Fault DAG is

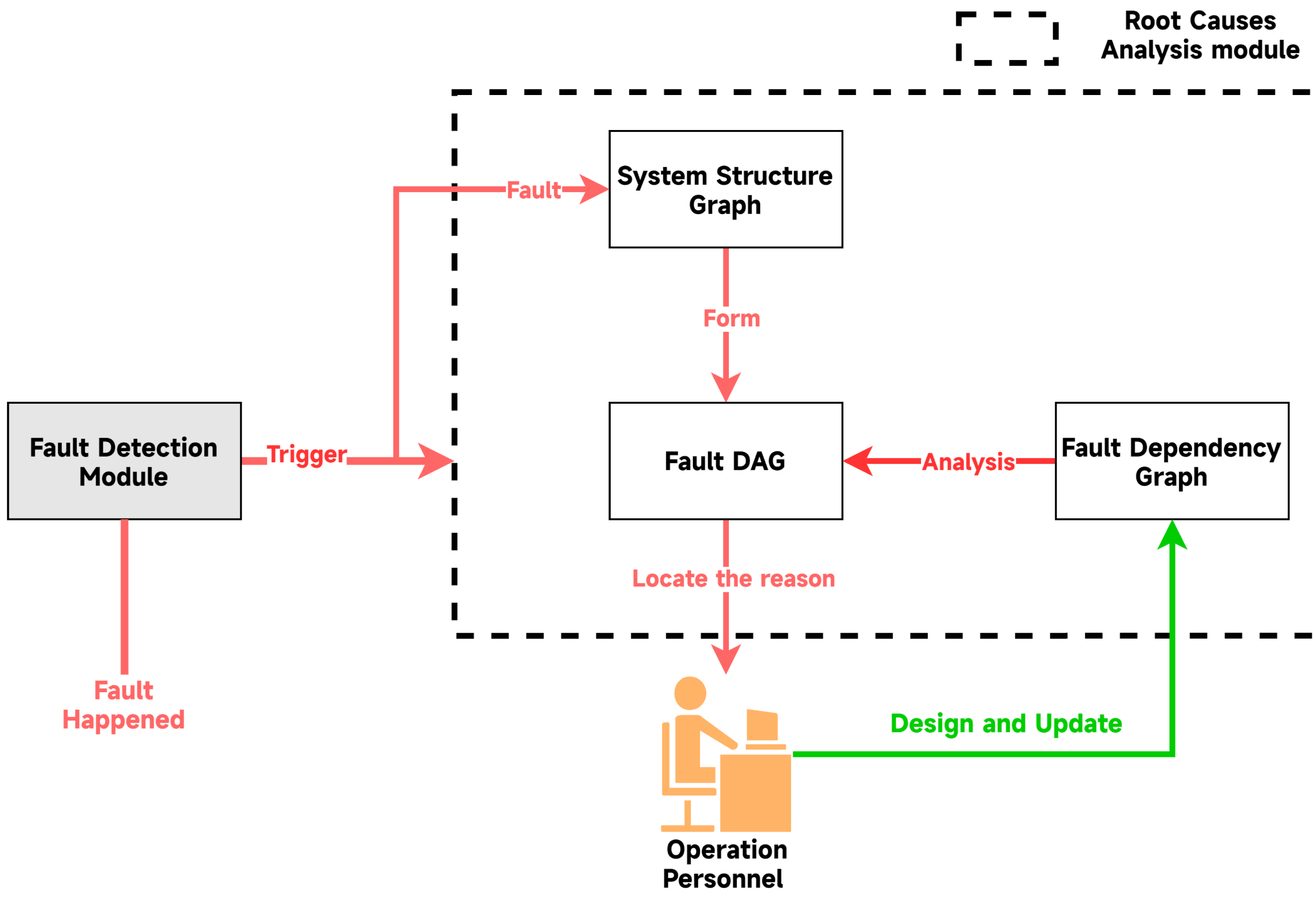


Fig. 6. Workflow of fault diagnosis.

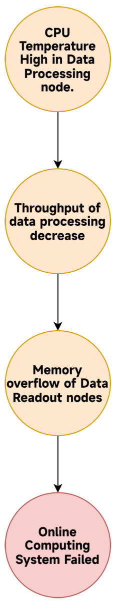

CPU Temperature High in Data Processing node.
Throughput of data processing decrease
Memory overflow of Data Readout nodes
Online Computing System Failed

Fig. 7. Example of a fault dependence graph.

algorithmically matched with the fault dependence graph to determine the root cause of the fault occurrence. The algorithm utilizes a priority queue, a data structure that arranges elements according to specific priorities (in FADD, the hierarchy and time of the nodes). In this context, each "element" corresponds to a fault phenomenon, which is also the node in Fault DAG. By updating and maintaining the priority queue, it is possible to obtain the node with the deepest hierarchy and earliest time that matches the fault dependence graph within the fault DAG.

Finally, the root causes analysis module outputs which component failed and what kind of failure it was. This happens within a matter of seconds.

IV. Application in LHAASO

A. Notification and Display

Notifications are sent in conjunction with LHAASO's actual on-call requirements. Alarms occur when errors are detected by the monitoring module. The online computing system at LHAASO is tightly interconnected, and the faults in a single component can cause many correlated error reports, so also considering their independence from each other, only the important alarms are reported. After the root causes analysis module determines the root cause of the fault, the system gives a notification of which factor of which component caused the fault. These notifications make it easy for the duty personnel to judge the phenomenon and the root cause of the fault. Alarms are immediately triggered when the data flow stops working, and they are retracted if the fault is recovered and the state of the data flow returns to normal.

In terms of notifications, we have developed a generic notification module called message publisher interface shown

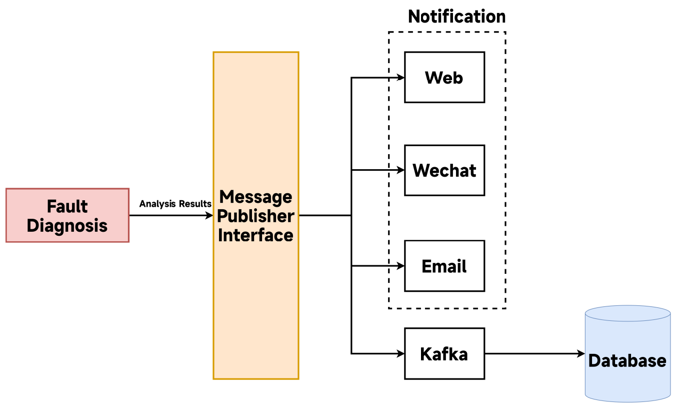
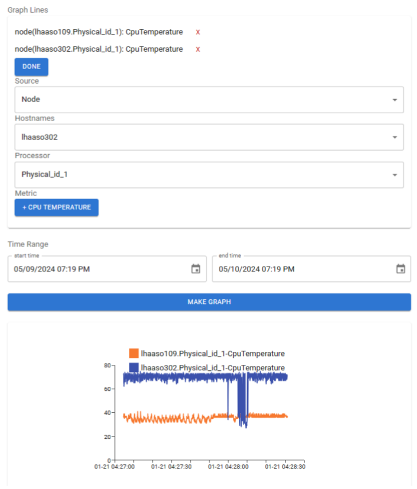


Fig. 8. Message publisher interface.


Graph Lines

node(lhaaso109.Physical_id_1): CpuTemperature x

node(lhaaso302.Physical_id_1): CpuTemperature x

DONE

Source: Node

Hostnames: lhaaso302

Processor: Physical_id_1

Metric: + CPU TEMPERATURE

Time Range

start time: 05/09/2024 07:19 PM

end time: 05/10/2024 07:19 PM

MAKE GRAPH

🟧 lhaaso109.Physical_id_1-CpuTemperature
🟦 lhaaso302.Physical_id_1-CpuTemperature
80
60
40
20
0
01-21 04:27:00
01-21 04:27:30
01-21 04:28:00
01-21 04:28:30

Fig. 9. Web page of FADD.

in Fig. 8, which supports alerting by multiple tools, including web, WeChat, and email. In addition, for archiving purposes, the module also sends and saves the messages to the database using Kafka at the same time.

The software's user interface is designed for on-duty personnel, featuring a multidimensional display of monitoring information and status information. The page mainly includes configuration for selecting and displaying dimensions, including source, host names, and metrics to be displayed. After the selection is completed, the page will draw the charts. The web page is shown in Fig. 9.

B. Reports

Regular operation reports are an important requirement for experiment operation management. The operation report of LHAASO includes on-duty time, on-duty personnel, online computing system operation status, and start and end times of operation, especially records of faults occurring during operation and their repair. This software is well-suited to meet such requirements. First, the software can accurately determine the start and end times of each operation by analyzing the occurrence of time slices in the data flow. Second, it can help maintenance personnel provide fault information. In addition, because the collected data are permanently stored, it is suitable for retrospective analysis at different timescales, statistical analysis of fault frequency, and other information.

V. Future Work

FADD is still under development and there are many areas where FADD needs to be further developed or researched.

In terms of monitoring, we plan to expand the scope of monitoring. By collecting data on White Rabbit switches of the time synchronization system, the power supply system, detector status, and other aspects, we can achieve the monitoring of the entire LHAASO experimental system, not just the online computing system. This will provide valuable assistance for comprehensive fault analysis.

Improvements in fault monitoring and diagnostics are still possible. The first is the use of unsupervised modeling to monitor anomalies. Our current choice of mathematical statistics is a reasonable method when the sample size is small. However, with FADD continuously collecting information and processing it, we can train the model using enough samples to make the process of monitoring it smarter. Second, failure root cause analysis can be aided by the use of neural networks. We have already noted the advantages of graph as a data structure in terms of dependence representation. Utilizing neural networks to assist expert systems is an approach worth trying in the future.

VI. Conclusion

The online computing system of LHAASO guarantees the operation of the experiment, data acquisition, and online computing. When the online computing system of LHAASO fails, it is an urgent need in the operation of LHAASO to analyze it quickly and give the root cause of the failure to the duty personnel.

From the actual on-duty needs of LHAASO, we designed and developed the FADD. The goal of FADD software is to quickly analyze the root causes of faults in the event of a failure by comprehensively monitoring the hardware and software of the online computing system. One of the highlights of FADD is that it captures the operational quality of the online computing software in a more detailed way, which allows for in-depth performance study and monitoring of the data flow software. In addition, by deploying the collection program in a distributed cluster, additional states of the hardware and software can be monitored. The collected data are permanently stored in the distributed database Cassandra. The status parameters are analyzed using two methods, threshold, and SMA, to determine whether a failure has occurred. We use a continuously running stream processing to determine if the data flow software is abnormal, and in the event of an abnormality, the fault diagnosis module automatically starts the analysis.

We use directed acyclic graphs to describe the system, the system's fault relationships, and the dependencies between faults, forming a system structure graph, a Fault DAG, and a fault dependence graph, respectively. The system structure graph reflects the interdependencies between hardware and software of the online computing system; the Fault DAG reflects the dependencies between components and their faults; and the fault dependence graph reflects the experience of the experts. The root cause of a failure can be determined using

Fault DAG and fault dependence graph. The results are notified to the on-duty personnel in a web visualization and a variety of ways.

At present, FADD has been applied in the operation of LHAASO. We believe the software can effectively reduce the time of personnel to analyze the causes of faults and reduce the burden of the duty personnel of the online computing system of LHAASO.

References

[1] C. Zhen et al., "Introduction to large high altitude air shower observatory (LHAASO)," *Chin. Astron. Astrophys.*, vol. 43, no. 4, pp. 457–478, Oct. 2019, doi: 10.1016/j.chinastron.2019.11.001.

[2] L. Collaboration, "An ultrahigh-energy γ-ray bubble powered by a super PeVatron," *Sci. Bull.*, vol. 69, no. 4, pp. 449–457, Feb. 2024, doi: 10.1016/j.scib.2023.12.040.

[3] The LHAASO Collaboration, "LHAASO collaboration. Very high-energy gamma-ray emission beyond 10 TeV from GRB 221009A," *Sci. Adv.*, vol. 9, no. 46, Nov. 2023, Art. no. eadj2778, doi: 10.1126/sciadv.adj2778.

[4] X. Lu, M. Gu, and K. Zhu, "Online real-time distributed data process of LHAASO ground shower particle array," *Nucl. Techn.*, vol. 43, no. 4, pp. 76–84, Apr. 2020, doi: 10.11889/j.0253-3219.2020.hjs.43.040402.

[5] Z. Yu, K. Zhu, M. Gu, F. Li, and M. Zhang, "Control and monitoring software of LHAASO DAQ," *Radiat. Detection Technol. Methods*, vol. 6, no. 2, pp. 227–233, Jun. 2022, doi: 10.1007/s41605-022-00327-3.

[6] H. Wang et al., "GRANO: Interactive graph-based root cause analysis for cloud-native distributed data platform," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 1942–1945, Aug. 2019, doi: 10.14778/3352063.3352105.

[7] A. Kazarov, G. L. Miotto, and L. Magnoni, "The AAL project: Automated monitoring and intelligent analysis for the ATLAS data taking infrastructure," *J. Phys., Conf. Ser.*, vol. 368, Jun. 2012, Art. no. 012004, doi: 10.1088/1742-6596/368/1/012004.

[8] P. Liu et al., "FluxRank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation," in *Proc. IEEE 30th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Berlin, Germany, Oct. 2019, pp. 35–46, doi: 10.1109/ISSRE.2019.00014.

[9] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "AutoMAP: Diagnose your microservice-based web applications automatically," in *Proc. Web Conf.*, Apr. 2020, pp. 246–258.

[10] M. Raza, J. Tahir, C. Doblander, R. Mayer, and H.-A. Jacobsen, "Benchmarking apache Kafka under network faults," in *Proc. 22nd Int. Middleware Conf. Demos Posters*, vol. 9, New York, NY, USA, Dec. 2021, pp. 5–7, doi: 10.1145/3491086.3492470.

[11] W. Mu, A. Zhang, W. Gao, and X. Huo, "Application of ARIMA model in fault diagnosis of TEP," in *Proc. IEEE 9th Data Driven Control Learn. Syst. Conf. (DDCLS)*, Nov. 2020, pp. 393–398, doi: 10.1109/DDCLS49620.2020.9275054.

[12] J. Carpenter and E. Hewitt, *Cassandra: The Definitive Guide*. Sebastopol, CA, USA: OReilly Media, 2020.