ISSN: 2424-8460(O)
2251-2608(P)

Original Research Article

Research on automatic test system framework for distributed architecture aimed at controllability issues

Tianhao Wu, Hongxiao Liu, Yayu Zhai, Xiaoquan Gao
China Institue of Marine Technology & Economy, Beijing, 100081, China

Abstract: To address the controllability and observability issues in distributed testing of intelligent autonomous systems, this paper proposes an automatic test system framework based on multi-port finite state machine (np-FSM) and greedy algorithm, introducing real-time middleware to support multi-task parallel testing. Firstly, a two-level architecture of "management node - test node" is constructed, ensuring test real-time performance and accuracy through dynamic scheduling and fault recovery mechanisms. Secondly, an np-FSM model is established for the formal description of the system under test (SUT), and a greedy algorithm is proposed to optimize the selection of key ports, reducing the number of control coordination messages and thus lowering communication overhead. Finally, a distributed test platform is designed based on real-time middleware, supporting loosely coupled combination and task collaboration of resource components, and four types of test case timing constraint relationships are defined to realize scalable automated testing. Experimental results show that the framework outperforms traditional methods in terms of test sequence length, communication overhead, and real-time performance.
Keywords: distributed architecture automatic test system; multi-port finite state machine; greedy algorithm; real-time middleware

1. Introduction

With the wide application of distributed systems [1] in key fields such as aerospace and military industry, their reliability testing [2] faces challenges of multi-node collaboration, high concurrency, and strong real-time performance. Traditional standalone test systems have bottlenecks in resource allocation and task scheduling [3]. Especially in scenarios where the input timing of test commands cannot be predetermined, existing methods rely on a large number of coordination messages, leading to a sharp increase in communication costs and affecting test efficiency [4].

This paper proposes a distributed automatic test system framework for controllability issues. The main contributions include: designing a hierarchical "management node - test node" architecture to realize dynamic scheduling and fault tolerance of test cases; establishing an SUT model based on np-FSM and optimizing key port selection with a greedy algorithm to reduce the number of control coordination messages; introducing real-time middleware to support multi-task parallel testing and defining four types of test case timing constraint relationships to improve the scalability and flexibility of the test system. The subsequent structure of this paper is arranged as follows: Section 2 reviews the research status; Section 3 elaborates on distributed testing methods, including system architecture, np-FSM modeling, and greedy algorithm optimization; Section 4 proposes the automatic test system framework based on real-time middleware; Section 5 summarizes the research content and outlines future research directions.

2. Research status of distributed test systems

Christian et al. proposed a testing method to solve controllability and observability issues by exchanging coordination messages between local testers [5]. Liu improved the UIO (Unique Input Output) test sequence generation algorithm; the improved algorithm can generate the shortest-length distributed test sequence and solve control and observation problems [6]. The basic approach of this algorithm is to first add synchronization messages and control messages to the original state graph, then generate test sequences using the UIO method. However, the algorithm is relatively complex to implement. Anier et al. analyzed the timing in protocol distributed system

ISSN: 2424-8460(O)
2251-2608(P)

testing and proposed timing constraints [7].

3. Distributed testing methods

3.1. Distributed test system

The composition structure of the distributed test system is shown in Figure 1. The management node is responsible for receiving test requests, assigning tasks, and summarizing results; Test Nodes 1~4 are connected to the management node via a bus to execute specific test tasks and feed back status. The system supports dynamic scheduling and fault recovery to ensure test real-time performance and accuracy.

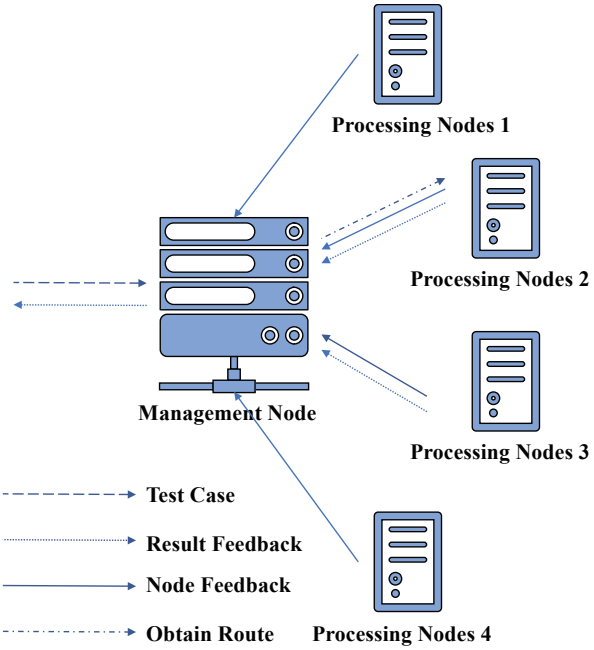

Figure 1. Structure of the distributed test system.

The distributed test system can be abstractly summarized as a "distribution-collection" process, with the specific workflow as follows: Firstly, test requests carrying a large amount of test data are routed to the management node. Secondly, the management node divides the test data into different test tasks using a screening algorithm and distributes the tasks according to the operating status of processing nodes in the entire system. Thirdly, each task is routed to a designated processing node and test processing begins. Fourthly, processing nodes synchronize test results and their current status to the management node. Finally, after collecting results from all processing nodes, the management node feeds them back to the user either individually or uniformly. As shown in the figure, each test node maintains timing synchronization through communication strategies. If a test node fails to receive test cases due to downtime or network anomalies, the test case execution engine of the management node immediately adjusts the test case scheduling method to ensure the real-time performance and accuracy of test case processing as much as possible. When the faulty test node recovers, it immediately requests synchronization signals from the intranet to synchronize the status of all test nodes in the current system, thereby restoring the state of the entire system.

Currently, relevant researchers have proposed several testing methods for distributed systems. The general solution is to introduce control coordination messages and observation coordination messages into test sequences, using these messages for synchronization to make the test system execute according to the pre-specified test sequence. Based on timing constraints, this study defines four types of task control timing relationships, reducing the number of new coordination messages added during testing for a given test sequence.

3.2. Test model for distributed systems

Since the test system based on the collaborative distributed test architecture consists of several testers, the formal description of the IUT (Implementation Under Test) in the collaborative testing method proposed in this paper uses a multi-port finite state machine (np-FSM). A multi-port finite state machine is a 6-tuple, denoted as A: A = (S, I, O, δ, λ, S_0), where:

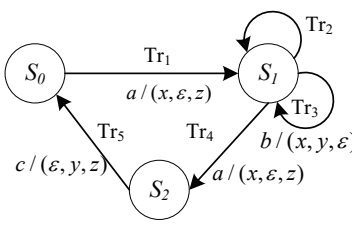

Figure 2. Three-port finite state machine.

ISSN: 2424-8460(O)
2251-2608(P)

In the figure, circles represent states, and arrows between circles indicate state transitions. Input a at state S_0 triggers a transition from S_0 to S_1, with outputs x on port 1, ∅ on port 2, and z on port 3. The SUT can be modeled using this multi-port finite state machine, where components in the SUT are regarded as ports of the finite state machine. A global test sequence is established based on the system model, and the mapping of the global test sequence on port P is called the local test sequence of port P. Through a local test sequence generation algorithm, the global test sequence can be mapped to each port of the IUT. In distributed testing, $Tester_p$ executes the local test sequence on port P. If each tester observes the expected input-output sequence, the IUT is considered consistent with the specification; otherwise, the IUT is deemed to have errors. Testers execute local test sequences according to a specific algorithm to complete the distributed system test.

3.3. Solution to controllability issues in distributed system test models

The controllability issue refers to a scenario in distributed testing where a tester cannot determine when to send the next input to the IUT if it has not sent the previous input or received the IUT output of the previous input. For example, $Tester_k$ cannot confirm when to send X_{i+1}. Additionally, controllability issues can arise in other situations, such as when the tester sending X_{i+1} is unaware of whether the IUT has received X_i.

This paper combines a greedy algorithm to calculate the port pairs with the most frequent controllability issues and selects these ports as candidates for $Tester_m$, thereby minimizing the number of channels established in the entire test system and reducing unnecessary communication and resource deployment waste. The idea of the greedy algorithm is presented below:

Algorithm 2-1: Greedy Algorithm
Input sets $C_1, \ldots, C_j$ Let $C = \varnothing$ and $CL = \{C_1, \ldots, C_j\}$ While $CL \neq \varnothing$: For all elements e in the union of sets in CL, let count represent the number of sets in CL that contain e Select the element e that maximizes count Add e to C and remove all sets containing e from CL End while Output C

The final output of the greedy algorithm is the set of port pairs that maximize count. $Tester_m$ is selected from this set, which reduces unnecessary communication and resource deployment—especially when the IUT has a large number of ports, the effect is more significant. In the local test sequence generation algorithm, combining the above algorithm with the following rules enhances the scalability and flexibility of the algorithm:

The solution to the controllability issue is to add control messages (denoted as C) to the test sequence:

Rule 1: If $\delta(S_h, X_i) = S_h'$ and $\delta(S_m, X_j)$ depends on S_h', then after sending X_i, $Tester_h$ sends C_1 to $Tester_m$;

Rule 2: If $\delta(S_m, X_j) = S_m'$ and $\delta(S_h, X_{i+1})$ depends on S_m', then after receiving X_j, $Tester_m$

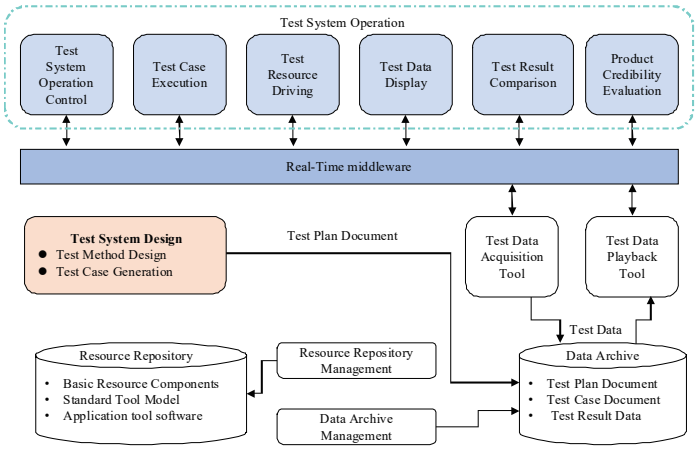

Figure 3. Automatic test system framework based on distributed architecture.

ISSN: 2424-8460(O)
2251-2608(P)

sends C_1 to $Tester_h$.

4. Automatic test system framework based on distributed architecture

To meet the multi-task parallel testing requirements of ship-shore integrated interaction systems, this project proposes an automatic test system framework based on a networked distributed architecture, whose composition structure is shown in Figure 3.

(1) Basic Resource Components: Provide functions required for testing independent and controllable computing platforms in the form of standard components, including test system operation control, test case execution, test resource driving, test data display, test result comparison, and product credibility evaluation. Data interaction between resource components is realized through real-time middleware.

(2) Real-Time Middleware: Provides real-time data interaction capabilities between distributed test nodes and resource components. The underlying layer of the real-time middleware adopts a reflective memory mechanism (VMIC) to improve the real-time performance of data transmission and realizes loosely coupled communication through a topic-based publish/subscribe mechanism.

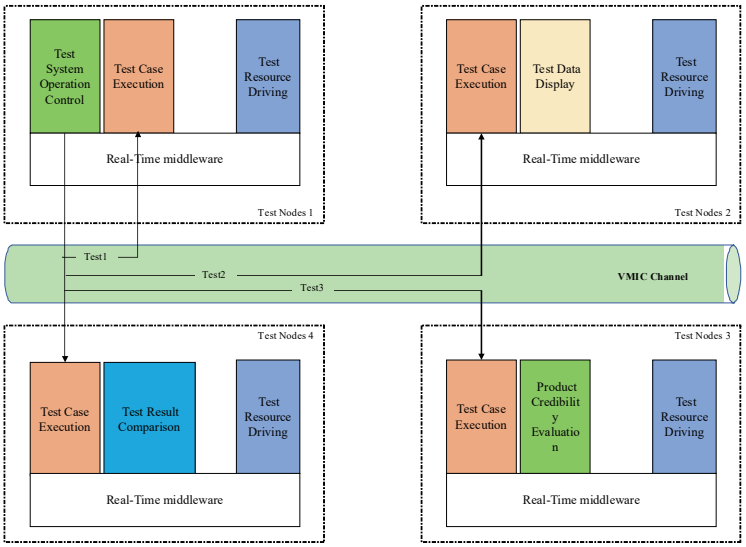

Figure 4. System structure diagram under multi-task parallel testing requirements.

(3) Test System Design Tools: Support the design of test schemes and test cases using various basic resource components. The test system design results (test scheme files and test case sets) are stored in the data archive.

(4) Resource Repository and Management Tools: Provide storage and management functions for basic resource components, standard comparison models, and application tool software.

(5) Data Archive and Management Tools: Provide storage and management functions for test scheme files, test case sets, and test result data.

(6) Test Data Acquisition Tools: Realize data acquisition during testing by calling basic services provided by real-time middleware, and store the acquired data in the data archive.

(7) Test Data Playback Tools: After the execution of test cases, support the playback of collected test result data to realize offline comparison functions.

A notable feature of this framework is that it provides various basic testing capabilities in the form of resource components, supporting flexible construction of test systems by selecting basic testing capabilities according to test tasks. Meanwhile, by adopting real-time middleware based on reflective memory technology, test tasks can be collaboratively carried out among various distributed test nodes. With the excellent real-time transmission capability of the reflective memory mechanism and the interoperability and composability provided by the real-time middleware software, the rapid deployment and operation of the test system can be realized, while ensuring the real-time performance of the test task execution process. Figure 4 describes the system structure diagram for realizing multi-task parallel testing based on the above framework.

During the execution of test tasks, scheduling the operation of test cases among distributed test nodes is an important means to meet multi-task parallel testing requirements. However, due to the characteristics of order constraints and time constraints on input parameters of the near-space UAV independent and controllable computing platform, test cases exhibit strong correlation. Through sorting, the relationships between test cases are summarized into the following four scenarios:

(1) Sequential Relationship: A test case can only be started after one of its predecessor test cases is executed, as shown in Figure 5a;

ISSN: 2424-8460(O)
2251-2608(P)

(2) Synchronous Relationship: Multiple successor test cases are started simultaneously, as shown in Figure 5b;

(3) Concurrent Relationship: A test case can only be started after all its predecessor test cases are executed, as shown in Figure 5c;

(4) Acyclic Relationship: No cycles or single nodes are allowed.

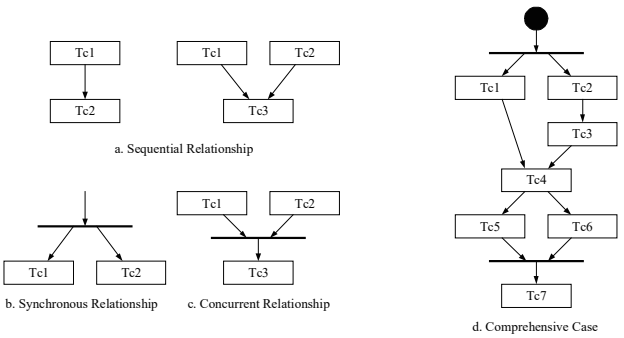

Figure 5. Test case relationship diagram.

The above four relationships define scenarios that may occur in different test requirements. The relationships in a test case set may be a single type or a combination of multiple types, as shown in Figure 5d: Test cases Tc1 and Tc2 can be executed simultaneously; Tc3 can only be executed after Tc2 is completed; Tc4 can be executed once either Tc1 or Tc3 is completed; after Tc4 is executed, Tc5 or Tc6 can be executed in any order, but must be triggered after Tc4 is executed; Tc7 needs to be executed after both Tc5 and Tc6 are completed.

5. Conclusion

Aiming at the problems of low efficiency and poor consistency in environment setup, execution efficiency difficult to match system scale, and insufficient testing accuracy in key scenarios in existing distributed system automated testing, this paper conducts research on the design and implementation of an automated testing framework. A distributed automated testing method adopting a "core module + extended plug-in" architecture is constructed, which consists of key modules such as test case management and load-aware execution engine. It has the advantages of strong architecture adaptability, high execution efficiency, and excellent testing accuracy, supporting the upgrading of distributed test toolchains, optimization of development iteration processes, and improvement of distributed system test evaluation systems.

About the author

Tianhao Wu(1999.06-), Male, Han, Household Registration: Beijing, China, Education Background: Master's Degree, Professional Title: Engineer, search Field: Reliability Engineering.

References

[1] Duan Shangze. Software Design of Automatic Test System Based on Distributed Network [D]. North University of China, 2024. DOI:10.27470/d.cnki.ghbgc.2024.001252.

[2] Ding Rui. Research and Application of Testing Technology for High-Reliability Decentralized Control Systems [D]. North China Electric Power University (Beijing), 2022. DOI:10.27140/d.cnki.ghbbu.2022.001278.

[3] Zhong Deming, Liu Bin, Ruan Lian. Research on Real-Time Task Scheduling Method for Standalone Platform of Embedded Software Simulation Testing [J]. Measurement & Control Technology, 2004, (04):52-53+56. DOI:10.19708/j.ckjs.2004.04.021.

[4] Pan Tao, Liang Faliang, Yue Long. Research on Timing Test Case Generation Algorithm for Embedded Software of Communication Equipment [J]. Information Technology, 2025, (04):168-173+179. DOI:10.13274/j.cnki.hdzj.2025.04.026.

[5] Christian Conte, Colin N. Jones, et al. Distributed synthesis and stability of cooperative distributed model predictive control for linear systems [J]. Automatica, 2016, 69:117-125.

[6] Liu W, Zeng H, Miao H. Multiple UIO-based test sequence generation for distributed systems [J]. Journal of Shanghai University (English Edition), 2008, 12(5): 438-443.

[7] Anier A, Vain J, Tsiopoulos L. DTRON: a tool for distributed model-based testing of time critical applications [J]. Proceedings of the Estonian Academy of Sciences, 2017, 66(1).