Al-Khwarizmi Engineering Journal, Vol. 8, No. 1, PP11 -17 (2012)

Al-Khwarizmi Engineering Journal

Design a Fault Tolerance for Real Time Distributed System

Ban M. Khammas

Department of Network Engineering/College of Information Engineering/ University of Al-Nahrain
Email:Ban_moh79@yahoo.com

(Received 7 March 2011; Accepted 23 October 2011)

Abstract

This paper designed a fault tolerance for soft real time distributed system (FTRTDS). This system is designed to be independently on specific mechanisms and facilities of the underlying real time distributed system. It is designed to be distributed on all the computers in the distributed system and controlled by a central unit.

Besides gathering information about a target program spontaneously, it provides information about the target operating system and the target hardware in order to diagnose the fault before occurring, so it can handle the situation before it comes on. And it provides a distributed system with the reactive capability of reconfiguring and reinitializing after the occurrence of a failure.

Keywords: *Soft real time, distributed system, fault tolerance*.

1. Introduction

Distributed systems have made substantial progress over the recent years in terms of functionality, scalability performance and openness so that they are an alternative even for very demanding and complex control systems.

This makes the developer of a functional and dependable system face enormous challenges largely because of loosely coupled hardware architecture with no physical shared memory, many things that straightforward in centralized systems are difficult in distributed systems. For example, synchronization processes with spread threads of control typically use shared variables on a single machine, but must do with message passing in a distributed system. The extra time delay associated with sending messages over a network increases the asynchrony of the processes and necessitates the use of special protocols to coordinate their respective actions [1].

A distributed real-time system (DRTS) is a distributed system whose correctness depends on meeting timing constraints as well as logical requirements. Using distributed computing to exploit the inherent concurrency of real-time systems leads to distributed real-time systems. For example, for a real-time system that can meet all deadlines with five processes, a distributed processing may be a solution. In this example one processor can be used to handle two periodic processes while the other processor handles the five periodic processes.

It can be seen that the distributed system can be introduced to improve a real-time system's response time and/or reliability. By decomposing a large real-time application into a set of processes, the decomposed processes can operate concurrently using interprocess communication and synchronization. Thus the response time can be improved with parallel processing. Reliability can be increased with distributed systems. This is important because real-time systems are expected to operate continuously with extremely high reliability even with the presence of a faulty processor. Fault tolerance can be achieved by detecting a faulty processor, saving and restoring the computational tasks of the faulty processor, and then distributing the recovered tasks to the remaining processors so that the DRTS can continue to operate, although with degradation of computing power.

The increase the reliability of the system offers the availability by using duplicate software and

backup computers and self-checking technique. For high availability and integrity, the replicas need to be diverse, so failures are sufficiently independent. For high performance, a sufficient number of replicas is needed in order to meet the load imposed on the replicated object [1].

Reliability and availability are essential characteristics for computer systems operation. A runtime monitoring system contributes for improving reliability and availability, respectively, by continuous failure detection and by reducing time to diagnose failures. In most systems, the savings in ground support and maintenance costs offset the initial cost of hardware and software monitoring system in a year [2].

Savings in ground support costs are achieved by reducing manual diagnostic costs. The run-time monitoring system will automatically identify failures as they happen in normal operation. Thus, support personnel spends less time and money in troubleshooting failures. This also results in reduced turn-around times and improved system availability [3].

2. Related work

Researches in this filed made substantial progress over the recent years in terms of functionality, scalability performance and openness.

Failure detectors are important building blocks for constructing fault-tolerant distributed systems. In [4] it takes a view of fault tolerance of real time computing and its attributes in automatic computing. And [5] discusses progress in the field of real-time fault tolerance in different way. In particular, it considers synchronous vs. asynchronous fault tolerance designs, maintaining replica consistency, alternative fault tolerance strategies, including checkpoint restoration, transactions, and consistent replay, and custom vs. generic fault tolerance.

In [6] it shows a simple algorithm and easy to implement. It also increases utilization speed and efficiency of scheduling. It can also be concluded that appropriate use of redundancy is important since too much redundancy increases reliability but potentially decreases the schedule ability. Too little redundancy decreases reliability but increases schedule ability. Also, designing, managing redundancy incurs additional cost, time, and memory and power consumption. Thus this algorithm can be efficiently used for fault tolerance in case where multiprocessors are used to run real-time applications. In [7] it proposes a reliability monitoring scheme for active fault tolerant control systems using a stochastic modeling method. The reliability index is defined basically on system dynamical responses and a safety region; the plant and controller are assumed to have a multiple regime model structure, and a semi-Markov model is built for reliability evaluation based on the safety behavior of each regime model estimated by using Monte Carlo simulation. Moreover, the history data of fault detection and isolation decisions is used to update its transition characteristics and reliability model. This method provides an up-to-date reliability index as demonstrated on an aircraft model. In thesis [8], it deals with techniques for tolerating effects of transient and intermittent faults. Re execution, software replication, and rollback recovery with check pointing are used to provide the required level of fault tolerance at the software level. Hardening is used to increase the reliability of hardware components. These techniques are considered in the context of distributed real-time systems with static and quasi-static scheduling.

3. The Architecture of the Proposal System

This research is done in special environment chosen to implement and test the work. It is programmed in Visual C++ (MFC) language, the computers are connected by Ethernet Network with 100 Mbps speed and the topology is star connection.

The (FTRTDS) has a distributed unit and a central unit. The Distributed Fault Tolerance Unit (DFTU) is distributed on all computers in the system and it is run with system boot of that computer as it will be explained in 3.1, and Central Fault Tolerance Unit (CFTU) which it is run manually by users (or by RTDS administrator) in one computer chosen for. It needs backup software for each part of distributed system, to use it when it necessary, as it will be explained later.

3.1. Distributed Fault Tolerance Unit (DFTU)

To give the CFTU full capability to diagnosis and control all faults on all computers in a network, there must be a program work as a service to it on each network's computer. This program is called DFTU. Its jobs are monitors the software and hardware of distributed system and diagnoses the error and tries to correct it before any fault occurred. And send a message to inform

the CFTU if the fault or error is done successfully or not.

The DFTU is run in each computer automatically with computer system startup as a service. The term "service" in Windows NT or Windows XP is used to denote both a special kind of win32 process and Windows NT kernel-mode device drivers. In fact, a component of the operating system known as the "service controller" (or "service control manager", or SCM) is used to load and control both types of services. In that context, a service is more or less a program that gets executed by NT (as opposed to getting directly executed by a user) and that responds to special requests to start, pause, or stop execution. Services have some special capabilities beyond those of the typical win32 process. For one thing, you can tell NT to start your service when the system loads, before any users have logged on. That makes services a good choice for software that needs to start automatically and run constantly in the background, whenever the system is up [9].

3.2. Central Fault Tolerance Unit (CFTU)

CFTU is run on one computer chosen by the administrator. Its jobs are monitoring the DFTU work, showing the state of RTDS and handling the fault that DFTU cannot handle.

It has several functions to do its job. The first one is called connection function which has two steps. The first step is called 'Identification step'. Its job is identifying each computer available in the network. In this phase, the CFTU make a connectionless service with all computers using mailslot technique which is a connectionless technique.

Mailslots are a simple way for a process to broadcast messages to multiple processes. One important consideration is mailslots broadcast messages using datagram. A datagram is a small packet of information that the network sends along the wire. Like a radio or television broadcast, a datagram offers no confirmation of receipt; there is no way to guarantee that a datagram has been received [9]. Because of that the CFTU repeat this step twice to insure that all the computers in its network send their names. At the end of this step computer's names and numbers are being collected; then the second step begins.

The second step is called 'Connection step' the named pipes will be used instead of mailslots for interprocess communications.

Named pipes are a simple way for two processes to exchange messages. They are like telephone calls: you talk only to one party, but you know that the message is being received. This type of connection will be used to increase the reliability of the systems because named pipe is an example of connection-oriented communications, the transmission of the data in this type is across a path that stays established until one of the nodes drops the connection. This type of logical connection guarantees that all blocks of data will be delivered reliably. While mailslots are an example of a connectionless communications, the transmission of the data in this type is across a network in which each packet is individually routed to its destination, based on information contained in the packet header. The path the data takes is generally unknown because there is no established connection between the computers that are communicating. Connectionless services can drop packets or deliver them out of sequence if each of the packets gets routed differently [10].

In this step, it uses the computer's names collected from first step to connect with each other by named pipes connection. Two pipes will be opened with each computer; one for sending and one for receiving to increase the speed. A thread will be created for each connection because the use of the single thread limits CFTU responsiveness. The reason for that limitation of responsiveness is that only one DFTU can be served at a time, even though the CFTU may be idle while waiting for the child process to complete. For example, the DFTU (client) must open a connection to the DFTU (server), send a request, wait for a response, and terminate the connection. Other DFTU (clients) cannot connect during this time. Thread can turn on this synchronously with concurrent DFTU (server) processing, long-lived client/server connections, and higher availability for client requests.

4. The Whole Demonstration of the proposal Fault Tolerance System Work

Before the RTDS's module is run, some information is needed to be known by CFTU, like their names, location, priority and so on. They are stored, by administrator before the run step, in a special file, called Information file.

As mentioned in 3.1, the DFTU runs automatically in each computer in the distributed system platform but the CFTU is run manually by the administrator and fill the information about the

system just for the first time it is run, i.e. not for each run the file information must be filled, but it can be updated.

When the CFTU is run the connection function begin first to connect the CFTU with each DFTU in the whole distributed system as described in 3.2 see algorithms (1, 2 and 3). Then create a new thread for Collection function whose job is collecting the hardware information about the whole distributed system, like memory space, hard disk space and processor information. The collection Function must run in a thread because it need to keep running when the RTDS run to collect the changing information about the system when it is running.

```
1.  Mailsloat connection
2.  If (success)
3.      Send computer name
4.  Else
5.      Go step 1
6.  End if
7.  Create a thread for read connection
8.  While()
9.      connect named pipe with CFTU for read
10.     if (fail)
11.         Error_No=Get Last Error()
12.         Call Error Handle (Error_No)
13.         Go to step 9
14.     End if
15.     Call connect write pipe function
16.     wait for any message from CFTU
17.     Enter Critical section //to not be interjection
                                  // by another thread
18.   If (connection cut off)
19.         Call Error Handle (Error_No)
20.         Leave Critical section
21.         Go to step 9
22.   End if
23.   Execute the message
24.   Leave Critical section
25.   Go to step 16
26. End While

Connect write pipe function ()
1.  connect named pipe with CFTU for write
2.  if (fail)
3.      Error_No=Get the Error
4.      Call Error Handle (Error_No)
5.      Go to step 2
6.  End if
```

Algorithm (1) DFTU Read and Write Functions.

```
1.  Create monitoring thread for each process
2.  If (process fail)
3.      Call Error Handle (Error_No)
4.      Save the error and the time
5.      if the same error repeat for 3 times in small
        period
6.      write to CFTU to handle this error
7.   End if
```

Algorithm (2) DFTU Distributed Process Monitoring Function.

```
1.  Mailsloat connection
2.  If (success)
3.      Read(ComputerNames, ComputerNO)
4.  Else
5.      Error Message
a.      If (continue)
6.      Go step 1
7.  Else
8.      End the program
9.  End if
10. End if
11. For (1 to ComputerNo)
12.     Create a thread for read connection
13.     Create a thread for write connection
14. End for

   --------Inside Read thread-----------------
1.      While()
2.      connect named pipe with DFTU for read
3.      if (fail)
4.          Error_No=Get Last Error()
5.          Call Error Handle (Error_No)
6.          Go to step 2
7.      End if
8.      wait for any message from DFTU
9.      Enter Critical section //to not be interjection
  by another thread
10.     If (connection cut off)
11.         Error_No=Get Last Error()
12.         Call Error Handle (Error_No)
13.         Leave Critical section
14.         Go to step 9
15.     End if
16.     Execute the message
17.     Leave Critical section
18.     Go to step 8
19.     End While

   --------Inside Write thread-----------------
1.      While()
2.      connect named pipe with DFTU for write
3.      if (fail)
4.              Error_No=Get Last Error()
5.          Call Error Handle (Error_No)
6.          Go to step 2
7.      End if
8.      End While
```

Algorithm (3) CFTU Connection Function .

When RTDS begins running, the DFTU creates a thread for each module that runs in its computer for monitoring function. Its job is monitoring that module job and is used from collection function information to tolerate the error before the fault occurs if possible and if not, it try to tolerate the fault by a different thing depending on the collection function information, like if module fail, it try to reload it again if there is no problem in this computer. But if the reload fail again or there is a problem in this computer, like there is no memory space, it informs the CFTU. The CFTU tries to reload it on another computer in that distributed system, (See Figure 1).

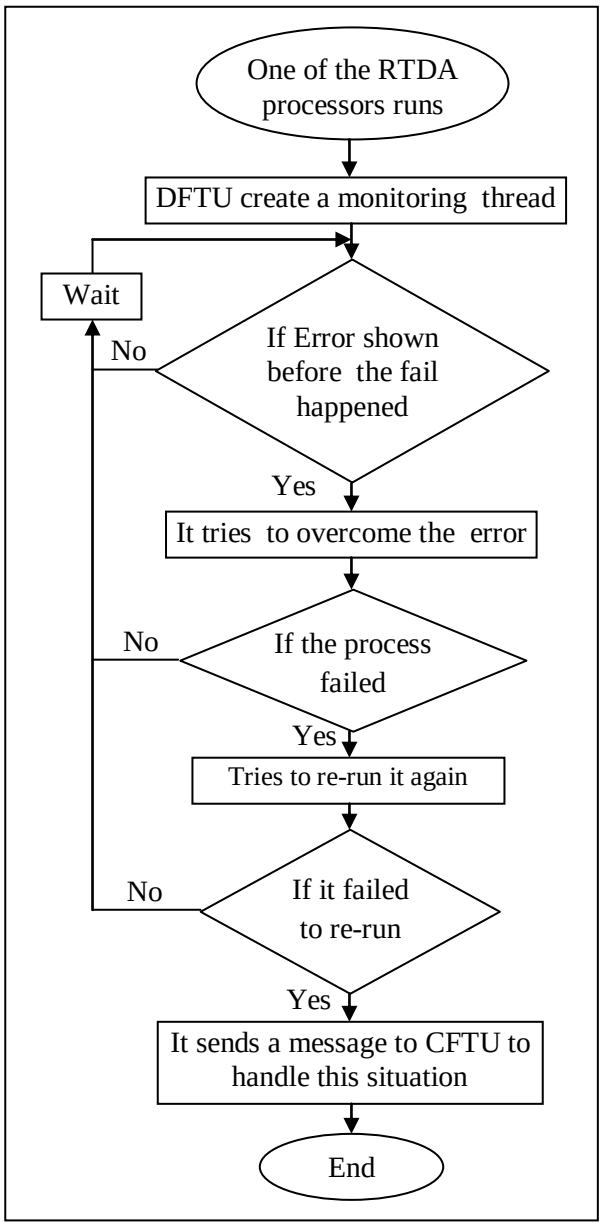

Fig.1. Monitoring Function .

The CFTU takes a handle of any bad situation, when the DFTU fails to control that bad situation, like computer breakout. In this situation CFTU decides to run the modules that were running in that broken computer in another suitable computer depends on the previous collection information.

5. Backup CFTU Service

After the CFTU creates monitoring thread it calls the backup service function. It chooses one DFTU depending on file information (filled by the administrator as mentioned in section 4) and declares it as CFTU recovery (CFTUR). It takes an action when CFTU is damaged.

When CFTU is damaged, all the DFTU detects that because the pipes are broken. Only the one which is chosen by backup service function will take a handle of that situation. When the CFTUR detects CFTU damage, it tries to reload it from its computer taking in consideration the last state, till it handles the error. See algorithm 4.

If the connection failed (i.e the computer is damaged or wire break, ect) it take no farther action till the administrator repairs the damage.

1. Reload the backup CFTU on the CFTUR //to handle the situation till we cane reload it in the same computer
2. If (Error_No == Database Lose) // i.e the software destroy
3. connect named pipe with DFTU of the same CFTU for read
4. If connection (success)
5. connect named pipe with DFTU of the same CFTU for write
6. Reload CFTU on its origin computer
7. Destroy the backup CFTU
8. End If

Algorithm (4) CFTUR Repair Function

6. Conclusions

The diagnosis capabilities of the system should be tailored to the needs of different users and applications. This can be achieved through a variation of the diagnosis techniques used to construct the results of reasoning.

This proposal design decrease the response time by using of hybrid technique of fault tolerance between distributed and central techniques led to increase the performance of the plant because of increase the speed of diagnosis the problem and handle it. It increases the reliability of the system by offering the availability by using duplicate software and backup computers.

It is easier to re-use in new circumstances than a conventional program because of using the diagnostic systems employing functional reasoning which is more adaptable. Given the soaring costs of software production it may be that the economics of software ownership.

For example, if it is required to modify an existing system for diagnosis of similar plant, only a need to update the data base that contains the structural description of the system is needed, when the set of components is the same for both the old and the new systems. When the two systems have many structured similarities, the modification becomes even simpler. Hence, diagnostic systems employing functional reasoning are very flexible and adaptable.

It decreases the number of failure by monitoring the whole system (software and hardware). This let the CFTU augury of the bad situation. So it takes handle before the failure occurs.

3. References

[1] B. Charron-Bost, F. Pedone, and A. Schiper (Eds.), "Replication", LNCS 5959, pp. 19–40, 2010. c_Springer-Verlag Berlin Heidelberg 2010.

[2] Andrew S. Tanenbaum, “Distributed Operating Systems”, Prentice-Hall, 2010.

[3] Sérgio Ricardo Rota and Jorge Rady de Almeida Jr.,"Run-Time Monitoring for Dependable Systems: an Approach and a Case Study", Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems 2004.

[4] Paul Ezhilchelvan, "On the Progress in Fault-Tolerant Real-Time Computing", Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems 2004.

[5] P.M. Melliar-Smith and L.E. Moser, "Progress in Real-Time Fault Tolerance", Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems 2004.

[6] A. Ch risty Persya and T.R. Gopalakrishnan Nair, " Fault Tolerant Real Time Systems", International Conference on Managing Next Generation Software Application (MNGSA-08), Coimbatore,2008.

[7] Hongbin Li, Qing Zhao and Zhenyu Yang, "Reliability Monitoring of Fault Tolerant Control Systems with Demonstration on an Aircraft Model", Journal of Control Science and Engineering, Vol. 2008, Article ID 265189, pages 135-145.

[8] Viacheslav Izosimov, "Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems", Linköping University, Department of Computer and Information Science, Sweden, 2009.

[9] Editors of Windows Developer’s Journal, “Windows NT Programming in Practice”, Miller Freeman, 1997.

[10] Microsoft Developer Network (MSDN) Library, April 2001.

تصميم مصحح الأخطاء لنظام حقيقي موزع

بان محمد خماس
قسم هندسة الشبكات/ كلية هندسة المعلومات/ جامعة النهرين
البريد الالكتروني:Ban_moh79@yahoo.com

الخلاصة

هذا البحث صمم مصحح أخطاء لنظام حقيقي مرن موزع. هذا النظام مصمم ليكون غير معتمد على الآلية الخاصة والخواص للنظام الحقيقي المرن الموزع الذي يعمل معه. هذا المقترح مصمم ليكون موزع على كل حاسبة من النظام ومسيطر عليه من قبل وحدة مركزية.
وبالاضافة لجمع المعلومات آنيا عن النظام، هو يجهز بمعلومات عن نظام التشغيل والأجزاء الصلبة المستخدمة مما يمكن من تشخيص الأخطاء قبل وقوعها، فيتم أتخاذ اللازم.ويزود النظام الموزع بالقابليةِ التفاعليةِ لإعادة التَشكيل وإعادة التَشْغيل بعد حدوثِ فشل.