Fault-tolerance by Duplication and Debugging for Distribution Real-time Systems

Zhiying Wang, Jeffrey J. P. Tsai and Chunyuan Zhang

Department of Electrical Engineering and Computer Science,
University of Illinois at Chicago,
Chicago, IL 60607, USA
E-mail: tsai@eecs.uic.edu

Abstract

Distributed real-time systems have continually being developed for many real applications. It is very important to assure the reliability of such systems. However, developing a fault-free distributed real-time system is very difficult. Duplication is one of the fault-tolerant technologies for improving the reliability of distributed real-time computer systems. In this paper, we present a dual-cluster distributed real-time system model with four states to improve the reliability of distributed real-time systems. The system employs cluster switch, output arbitration and fault detection to increase its reliability. The system’s availability is evaluated according to the model. We also introduce a dynamic approach for visualizing and debugging the timing behavior of distributed real-time systems.

Key Words: distributed real-time systems, dual-cluster system, switch, debugging, timing analysis

1. Introduction

Distributed real-time systems are increasingly being developed for use in everyday life. Examples include command and control systems, flight control systems, robotics, patient monitoring systems, and process control systems. The failure of these systems could result in huge economic damage or even cost human life. It is very important to assure the reliability of distributed real-time systems.

Duplication is one of the fault-tolerant technologies for improving the reliability of distributed real-time systems. In this paper, we present a dual-cluster distributed real-time system model with four states to improve the reliability of distributed real-time systems. The system employs cluster switch, output arbitration, and fault detection to increase its reliability. The system’s availability is evaluated according to the model. We also introduce a dynamic approach for visualizing and debugging the timing behavior of distributed real-time systems.

The paper is organized as follows: Section 2 presents the system model for fault-tolerant computing. Section 3 discusses switch and duplication results processing which are key technologies in system implementation. Section 4 analyzes the system reliability in theory and quantify. Section 5 introduces a dynamic approach for visualizing and debugging the timing behavior of distributed real-time systems.

2. System model

2.1 Architecture

In our system, we divide every service request into n steps based on the application and every step is processed by a different computer node. Thus, each service is performed by n computers. During this process, the output of step i is the input of step i+1 (1 ≤ i < n). To achieve high system reliability, we adopt the duplication technology. The entire system consists of two equivalent clusters, each of that includes n processing nodes to provide the n steps service. In addition, there is one duplication management node for controlling the whole system. The results from those two clusters are passed to the
duplication management node that outputs only one result after selection. Thus, the system forms a duplicated distributed system whose system model is shown in Figure 1.

In the model, the system has two identical clusters, each of which consists of a complete set of computers processing the same service. The inputs to the two clusters come from the same source and are sent to the two clusters separately. A system state space is used for system fault-detection and will be discussed in Section 3. One of the outputs from the two clusters is chosen as the final output of the system based on a matching and comparison policy which will be also described in Section 3.

2.2 Working Process of the System

The system works in duplication mode in order to meet the fault-tolerant requirement. In other word, there are two clusters working at the same time under normal condition.

The duplication mechanism is in master-agent mode. The master resides on the duplication management node and all the other nodes act as agents respectively. The master controls the entire system to work in line and sends out commands such as ‘system start’, ‘cluster switch in’, ‘cluster switch out’ and so on. The agents on the other nodes receive these commands, act correspondingly and give the response when completed.

The duplication management node also processes the results from the two clusters besides processing commands. First, the results of one service must be matched with the service identification number (a service ID can uniquely identify one service). After matching, one of the more reliable results from the two clusters is chosen and sent out. Those results which have not matched from another cluster will be sent out in a single mode. The length of the time interval and the way the results are managed rely on the application.

2.3 System States

The system is operated with four states: duplication (normal condition), hot-backup, warm-backup and cold-backup. These states differ in how the two clusters processing data and how the results being handled. In the duplication state, both clusters are under normal condition. They receive the same input for processing and the final output is chosen from the results of the two. In hot-backup state, all clusters are in working state. However, the backup cluster only receives the same inputs for processing and produces result. The result generated by the backup cluster is not used and can not be sent out. In warm-backup state, one cluster is in normal working condition and the other backup cluster is only ready to receive inputs but not produce any outputs. In cold-backup state, one cluster is on duty and another is shut down. The differences among these states are listed in Table 1 in which 1 or 2 represents number of cluster.

Table 1. System States

<table>
<thead>
<tr>
<th>States</th>
<th>Clusters receiving inputs</th>
<th>Clusters producing results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplication</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Hot-backup</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Warm-backup</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cold-backup</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The transition among the four states is illustrated in Figure 2. In Figure 2, the fault condition means that there is something wrong in the system that can be detected somehow, but the system can still work properly. Most faults are reported by application. On the other hand, the failure condition indicates that the system encounters a serious problem and cannot go on. In such case, the system has to quit from the running state. Failures are usually reported by the system. In general, the working system is either in the
duplication state or in the single mode (three backup states).

![System States Transition](image)

**Figure 2. System States Transition**

3. **Switch and Duplication Results Processing**

3.1 **Switch Technology**

In duplication systems, the units to be switched can be a whole machine cluster or a single computer node. To reduce the complexity of system design and increase the system reliability, we apply cluster switch technology in our system. In our system, switching includes cluster ‘switch in’ and ‘switch out’. The transition from the duplication mode to a single mode is called cluster ‘switch out’ and the transition from a single mode to the duplication mode is called ‘switch in’.

Cluster ‘switch out’ process is relatively simple. The computing nodes connected to a cluster will be switched out when they receive the ‘switch out’ command from the master on the duplication management node. The agents on the computing node free the resources that have been used and then quit. After that, the system is in a single mode.

In contrast, cluster ‘switch in’ is quite complicated due to the cluster synchronization. This type of switch has to deal with data coherence between the cluster on duty and the cluster to be switched in, context recovery on the switched in cluster, and synchronizing two running clusters.

To make the data on the two clusters coherent during a switching-in process, the cluster on-duty must stop processing service requirement and wait for the switched-in cluster to recover context data from it. The context data is static and dynamic background data using for service processing. The recovering process will not start until all the service requirements waiting for response in the on-duty cluster are finished. As a result, the switched-in cluster can read data from the on-duty cluster. After that, the two clusters can start running from the same scratch line. Note that we have to try to recover as little context data as possible since this is a real-time system and the switch in process cannot take too much time to do this.

For data recovery, service requests must be accepted according to system high reliability requirement. Therefore, buffers must be allocated in the system to store arrived service requests during the switching-in process. In our system, we apply a dual-buffer mechanism to solve this problem. There are two queues, working queue and backup queue, on both the switched-in node (the node accepting input) and the on-duty node. Under normal conditions, service requests are input and processed in the working queue.

After receiving the ‘switch in’ command from the duplication management node, the on-duty node will go through the following steps:

1. Change the input pointer to the backup queue to make the working queue empty after the request is completed;
2. Send a ‘data recovery’ command to all switched-in nodes to recover the context data when the working queue is empty and the last request is done;
3. Change the output pointer to backup queue and start to work normally; and
4. Exchange the working queue and backup queue.

At the same time, the switched-in nodes also perform the following tasks on the reception of a ‘switch in’ command:

1. Put the input pointer to the working queue and start to accept service request;
2. Start to recover the context data after receiving the ‘data recovery’ command from the on-duty node; and
3. Complete data recovery, put the output pointer to the working queue, and respond to the on-duty nodes to work synchronously.

The entire pointer exchange process, known as the dual-buffer switch-in principle, is demonstrated in Figure 3 in which two buffers correspond to working queue and backup queue respectively.
Note that the service requests kept in the working buffer and the backup buffer may vary due to time difference. The switched-in nodes may keep one or two more requests than the on-duty nodes do. Thus, it is necessary to delete these redundant requests before the queue in the switched-in nodes.

Cluster synchronization is another issue we have to cope with. This needs same context data. It is completed by data recovering in switch-in node. The context data include two parts of static and dynamic data. Static context data are unchangeable in running process. But dynamic context data are changeable with running time. The static context data are first recovered in each node. After receiving the ‘data recovery’ command, all the switched-in nodes recover the dynamic context data from the on-duty nodes. Since the data recovery on each node is different, the time when the two clusters start to work must be controlled. This task is performed by the management node. The agent on each processing node sends ‘ready’ information to the duplication management node after data recovery, and then wait for the ‘startup’ command. The master on duplication management node keeps looking up the states of all processing nodes after all processing nodes are ready. Once all processing nodes receive the ‘startup’ command, the system goes into the duplication state. A flow chart of cluster switching process is shown in Figure 4.

In switch-in node aspect of Figure 4, it first recovers static context data and starts accepting input, but do not process input. Then dynamic context data are recovered from on-duty node because dynamic context data only appear in processing node. Switch-in node sends a request to on-duty node and receives the dynamic context data and an ‘end’ signal from on-duty node. At last it reports ‘ready’ state to duplication management node and enters waiting. Once ‘startup’ command is received from duplication management node, switch-in node begins its processing and switch-in procedure is completed.

Figure 3. The Dual-buffer Switch-in Principle.

Figure 4. Flow Chart for Cluster Switch.

In on-duty node aspect of Figure 4, when it receives ‘recover context data ’ request from switch-in node, it first changes the input pointer of buffer queue of itself. Then it sends the dynamic context data and ‘ end’ signal to response the request of switch-in node. At last it also reports ‘ready’ state to duplication management node. Once ‘startup’ command is received from duplication management node, on-duty node continues its processing. Because synchronization and coherence of dynamic context data of two node, on-duty node must wait till switch-in node has received all dynamic context data.

In general, switch-in procedure can be completed from few milliseconds to hundred milliseconds according to the number of dynamic context data.

3.2 Duplication Results Processing

The duplication management node performs
two tasks to deal with the results from the two clusters. One is to choose a better result according to some and send it out after matching two results of the same service request from the two clusters. Another is to produce statistics, such as service quantity, processing time etc., and produce report information to operator.

3.2.1 Results Matching, Comparison and Selection Process

Matching is the first process after results being sent to the duplication management node. In order to avoid two results to one service caused by both clusters in working state, an identification number is given to each service request. Assignment of identification number is the key of matching process. Certainly, buffers must be allocated to store results. The matching tactic is depending on the system. Those results that have stayed in the buffer for a long time but not matched with its counterpart should be sent out in a limited time. However, this condition should be reported to the duplication management node since it can reflect the system states.

If two results from the two clusters for one service are the same, outputting either one is acceptable. When the results are different, a better result will be selected and sent out. A quality checking mechanism designed to select a better one from two results has been built into the system. Based on this mechanism, all the processing nodes have to evaluate their own processing quality regarding each service request after completion of processing. The evaluation is sent to the duplication management node along with the service result. The final output is determined by the quality evaluation of all processing nodes. Further more, the duplication management node must give the statistic duplication-quality, which is accumulated in some time intervals and reported to operator. The operator can then know the state of the entire system based on this information.

The following are general selection rules used in our system:

1. If two results are the same, the result from the on-duty cluster is generated. In this situation, the duplication-quality is the best;
2. A weighted summation of the quality of all processing nodes of for each cluster is calculated respectively if the two results differ. (In implementation, each processing node can be given different weight.) The final result derived from the cluster with a better quality. Under such a condition, the duplication-quality should be relatively worse; and
3. If error occurs in software or hardware of the processing nodes (the detection process is described later), the duplication-quality should be the worst.

Following is an example: assuming that the duplication-quality has four levels (0 to 3) with 0 being the best and 3 the worst. The gathered value is simply the summation of all processing nodes quality. Figure 5 indicates the comparison process.

![Figure 5. The Comparison Process](image-url)

Note that the comparison should be performed under duplication condition. If the system is in a single state, this process is not needed. Also, the duplication node only gives the duplication quality (identify the single state).

3.2.2 Statistics

The statistical analysis is dependent on the actual system. The results sent to the duplication management node may include all types of information on every processing node. The duplication management node can do statistical analysis from the data, such as proportion of type of
service, the states of nodes and so on. These statistics should be gathered when a time unit is over. Such statistical information is sent to operator screen for monitoring the state of the whole system.

3.3 Fault Detection

In the model, the way to detecting system faults relies on setting the system state table in the system communication space. In this table, the states of all the nodes are set by flags including the hardware states, the software states and so on. Faults of software or hardware including OS and application program fault are used to control the whole system.

In addition, there is a counter for every processing node. The counter is used to detect if the node is still alive because every processing node changes the counter value continually. If the counter value is not changed for some time, the node is considered to be dead.

The master on the duplication management node keeps watching the state table. The master process will inform the management process of the corresponding condition if there are errors. The management process will then send out system commands, such as cluster switch out, to adjust the system state.

4. Analysis of the Reliability of the System

First of all, we make two assumptions on the system. 1) Every processing node has two states: normal state and faulty state, and the failure probability of every node is independently distributed. 2) The lives of all processing nodes submit to exponential distribution, and the time from failure to restoration also obeys exponential distribution, where \( \lambda \) represents the probability of failure, \( \mu \) represents the probability of recovery.

In simple calculation, it is often assumed that the probability of failure-detection is 100 percent. Certainly, it is too difficult to achieve this in reality. Also, it is not allowed if the time needed to detect fault is too long, but this time exists impersonally. So the analysis to system reliability must consider the probability and the time of failure detection.

Now we assume that the probability of success of failure detection is \( C \), the average time of system failure detection is \( (1/\beta) \), then the probability of system failure detection is \( \beta \). We build a model based on Markov process as shown in Figure 6.

![Figure 6. The System State Transition of Markov Process.](image)

\[
A = P_0 + P_1 = \frac{1 + 2 \frac{\lambda}{\mu}}{1 + \frac{\lambda}{\mu} + 2 \left( \frac{\lambda}{\mu} \right)^2 + 2(1 - C) \frac{\lambda}{\beta}}
\]

Some interesting points in Equation 1 have to be mentioned here. When \( C \) (the probability of success for failure detection) is a constant, the higher the \( \beta \) (the probability of system failure detection), the higher system reliability. When \( \beta \) is a constant, the higher the \( C \), the higher system reliability. In other words, the probability of success for failure detection and the probability of system failure detection must be improved in order to increase the system reliability [5].

5. System Debugging

Debugging distributed real-time systems is difficult because of timing constraints imposed on them and their non-deterministic execution behavior. In a distributed real-time system, the correctness of the system depends not only on the logical results of the system behavior, but also on the time at which the results are produced. Because of the non-deterministic character, the system may not exhibit the same execution behavior upon repeated execution of the same program with the same inputs.
A proper operation of the whole system, which consists of the real-time target processes and the real-world process, depends on the capability of the target processes to comply with certain timing constraints. Thus, one of the most important problems, especially for distributed real-time systems, is to determine if timing constraints have been violated.

Much research has been focused on the timing analysis of distributed real-time systems [2]. A dynamic approach for visualizing and debugging the timing behavior of distributed real-time systems is presented in [3,4]. The main concern of this approach is to find the causes of a timing constraints violation according to the imposed timing constraint and the recorded system execution trace. In this approach, the execution traces containing the run-time information of an on-line target program are first recorded during the execution of the target program using the non-interference monitoring system [2,5]. Now we extend this approach to our duplication real-time system for debugging.

We first collect all running data by the non-interference monitoring system. The collected data is then interpreted and analyzed on off-line using a visualization and debugging method which has presented in the previous research [3,4]. The visualization method is designed to graphically display the correlated data for a better understanding of target program’s timing behavior. The debugging method is designed to locate and fix the program segments where the timing constraints are violated. The timing behavior of a target program is visualized as two graphs—the Colored Process Interaction Graph (CGIP) and the Dedicated Colored Process Interaction Graph (DCGIP). The CGIP depicts the timing behavior of a target program by graphically representing the inter-process relationships during their communication and synchronization. The DCGIP is derived from the CGIP by imposing a timing constraint on it. Unlike the CGIP which shows the entire timing behavior of a target program, the DCGIP reduces debugging complexity by focusing only on the portion of a target program’s timing behavior which has a direct or an indirect effect on a timing constraint. With the help of the CGIP and the DCGIP, a timing analysis method is presented for computing the system-related timing statistics and analyzing the causes of timing constraint violation.

A target program is considered to be a set of processes running on the target system. Each process is identified by an integer known as the process identification number. A process is considered to be a sequence of events, and various events can occur in different process at the same time.

At any time between a process’s creation and termination, it is in one of the three states—running, ready, and waiting. They correspond three basic causes of timing constraint violations—computation, scheduling, and synchronization respectively.

Timing constraints are further classified as intraprocess timing constraints and interprocessor timing constraints in distributed real-time systems. An intraprocess timing constraint is a TC whose relevant processes are all on the same target node. An interprocessor timing constraint is a TC(timing constraint) whose relevant processes disperse on more than one target node.

Figure 7 is a example CGIP in which there are two duplication node i and j, and each node has 4 processes. Here we suppose that node i matches two results and outputs. This figure shows the beginning and one real-time application in 25ms cycle. This figure can be transferred to DCGIP which is used by VDS (visualization and debugging
system) for debugging of a real-time system [3,4]. We can do timing analysis for debugging system using this figure. For example, we can get timing statistics, such as the global total waiting time, the global total ready time, local total execution time, the local total waiting time and the local total ready time etc.

6. Conclusion

With the widely used computer systems, the problem of system reliability becomes more and more prominent since the lost is greater and greater if systems fail. System reliability is usually at the price of system cost and system performance. The duplication technology used in this paper may be a better tradeoff. Although duplication technology has a long history, and more and more new technologies will emerge with the development of the computer technology, the duplication technology can still be given a new life.

In this paper, we build a real-time system model that has been widely applied in reality. The technologies described in this paper have been implemented in a four-computer system with two computers for each cluster [6]. The results in our experiment are quite good. Certainly, in the actual application system, some of our methods should be modified according to the requirement of users.

We also can use the dynamic approach for visualizing and debugging the timing behavior of our distributed real-time application systems. The main concern of this approach is to find the causes of a timing constraints violation according to the imposed timing constraint and the recorded system execution trace. Those accelerate the system debugging and guarantee system correctness.

References