Fault Tolerant Approach for Distributed Real-Time and Embedded System using Reflection

Archana Kale and Ujwala Bharambe

Abstract—**Fault tolerance (FT) is a crucial design consideration for mission-critical distributed real-time and embedded (DRE) systems, which combine the real-time characteristics of embedded platforms with the dynamic characteristics of distributed platforms. Traditional FT approaches do not address features that are common in DRE systems, such as scale, heterogeneity, real-time requirements, and other characteristics. This paper describes reflection approach applied in DRE system for fault tolerance. We have proposed an algorithm using combination of replication and reflection technique to be applied in DRE system for fault tolerance.**

Index Terms-- **fault tolerance, reflector, replication, active replication, distributed real-time embedded systems**

I. Introduction

Distributed Real-time Embedded (DRE) systems are a growing class of systems that combine the strict real-time characteristics of embedded platforms (e.g., constrained resources and deadline criticality) with the characteristics of distributed platforms (dynamic environments). As these systems increasingly become part of critical domains, such as defence, aerospace, telecommunications, and healthcare, *fault tolerance (FT)* becomes a critical requirement that must coexist with their real-time performance requirements [2]. DRE systems have several characteristics affecting their fault tolerance:

DRE systems typically consist of many independently developed elements, with different fault tolerance requirements. This means that any fault tolerance approach must support mixed-mode fault tolerance (i.e., the coexistence of different strategies) and the coexistence of fault tolerance infrastructure (e.g., group communication) and non-fault tolerance infrastructure (e.g., TCP/IP). DRE systems stringent real-time requirements mean that any fault tolerance strategy must meet real-time requirements with respect to recovery and availability of elements and the overhead imposed by any specific fault tolerance strategy on real-time elements must be weighed as part of the selection of a fault tolerance strategy for those elements. DRE applications are increasingly component-oriented, so that fault tolerance solutions must support component infrastructure and their patterns of interaction. DRE applications are frequently long-lived and deployed in highly dynamic environments. Fault tolerance solutions should be adaptable at runtime to handle new elements [1][4].

There have been different types of approaches for fault tolerance:

- **Graph based scheduling/distribution heuristics**
  These types of heuristics[6] [7] [8] combine real-time constraints, distribution constraints, algorithm specifications (operations and data dependencies) with architecture specifications (processor and communication link). The distribution constraints assign a set of processors to each operation (along with value of execution duration) of the algorithm graph and produce static distribution schedule followed by real time distributed executives. Synchronization between processors is stated by algorithm specifications.
- **CORBA-Based Fault tolerance approaches :**
  DRE systems with hard real-time requirements [11] have been developed with the CORBA middleware for run-time support to automate many distributed computing tasks. QoS requirements of DRE systems, particularly dependability and predictability, are addressed by the OMG's Fault tolerant [9] and Real-time CORBA [10] specifications. But this approach has several challenges[12] i.e. Non-determinism, Expensive Replication, Inablity to meet requirements of fault tolerance and Semantic Incompatibilities Between features of CORBA versions, Lack of Standards to Handle Byzantine and Partial Failures, Lack of Standard End-to-end QoS Configurability etc.
- **Middle ware based Semi active replication & Replica Communicator:**
  Further modifications have been made which describe extensions and solutions to achieve fault tolerance. Semi-active replication[12] ,Model engineering[13], Self configurative replica manager[1], Meta object architecture[14] and reflection. Reflection been proposed during the last decade as a fruitful paradigm to separate non-functional aspects from functional ones, simplifying software development and maintenance whilst fostering reuse.

Fully reflective databases are not feasible due to the high cost of reflection. So we have proposed a fault tolerant solution for DRE system which is combination of both Replication and

Archana Kale, Asst.Professor, archiekk@yahoo.co.in,
Ujwala Bharambe[1] Lecturer,ujwala.b@gmail.com
Dept. of IT, Thadomal Shahani Engineering College, Bandra(w), Mumbai-50.

reflection. We have considered basic architecture of distributed systems and proposed one new innovative technique for fault tolerance using replication and reflection.

II. CHALLENGES IN PROVIDING FAULT TOLERANCE IN DRE SYSTEMS

We first motivate our work by describing the fault-model and general approach under which our system operates. In providing fault tolerance for any DRE System there are following challenges, specifically [1]:

- Communicating with replicas in large scale, mixed mode systems
- Handling dynamic system reconfigurations
- Handling peer-to-peer communications and replicated clients and servers.

A. *Fault-Model and Fault Tolerance Approach*

A fault model describes the types of failures we expect our system to deal with. By being specific about our fault model, we make clear the types of failures the system is designed to handle.

For our solution, we assume that all faults are fail-stop at the process level. When an application process fails, it stops communicating and does not obstruct the normal functioning of other unrelated applications. Network and host failures can be seen as a collection of process failures on the element that has failed.

We tolerate faults using replication strategies. In these schemes, we use multiple copies of an application called replicas to deal with failures of the applications. There are two types of replication strategies: active replication and passive [16] replication. We use the active replication strategy where all replicas need to be deterministic in their message output, and each replica responds to every input message. Our solution ensures that only a single request or response is seen regardless of how many actual replicas are used. Though it uses active replication, we maintain only one leader (the one which has the minimum load) replica that responds to the messages and shares its state with any other non-leader replicas, so they can take the leader's place in case of a failure. We call all replicas as active replicas because when any request comes from the client, all these replicas will perform that job independently and only the leader and the reflector replica will receive all the results from all active replicas. Depending upon system operational modes (our proposal) leader will send reply to client.

Fault Detection: For designing fault tolerance system, fault categorization is required. We have considered specifically only process level faults in DRE system, which can be categorized as-

1. *Execution domain faults:*
   Caused within the software other than algorithm logic; such as memory leakage, segmentation fault, divide by zero error, spin in an infinite loop, deadlock, and live lock etc.
2. *Logic Domain faults:*
   Usually caused by the logic of the underlying algorithm itself, that defines the computational logic.

We have considered the execution domain faults in DRE Systems. When such a fault occurs, rather than recovering that process we consider takeover of the system by other non-faulty process.

B. *System Model*

We assume a conventional client / server model where servers process client requests and return the results of this processing. Servers encapsulate data (their state) and code (describing the services they offer to clients) [3]. When a service request is received, an "execution point" appears within the server. This execution point travels through the code, processes the received request, possibly modifies the server's state, and possibly produces a reply that is returned to the client. We also assume that every job will have single process and every process will have some intermediate values (value set). In this section, we don't make any assumption about the nature of servers, but we assume that server replicas are "distributed" so that they do not fail simultaneously. Our notion of server is very similar to those of "replication entities" or "distributed processes" commonly found in works on distributed algorithms.

C. *Basic Concept*

In this paper we try to apply replication and reflection to DRE system for fault tolerance.

Reflection can be defined as the property by which a component enables observation and control of its own structure and behavior from outside itself.

A reflective system is basically structured around a representation of itself —or *meta-model* — that is causally connected to the real system [2]. This approach divides the system into two parts: a *base-level* where normal computation takes place and a *meta-level* where the system computes about itself (*meta-computation* or *metalevel software*). (See Fig.1)

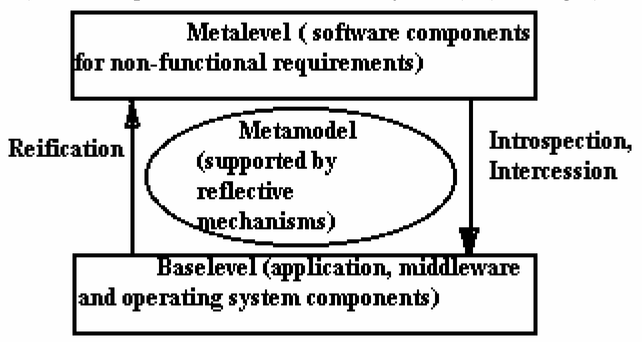
Metalevel (software components for non-functional requirements)
Reification
Metamodel (supported by reflective mechanisms)
Introspection, Intercession
Baselevel (application, middleware and operating system components)

Fig. 1. Organization of a Reflective System

The meta-model results from the interactions between the base-level(application), and the meta-level (fault-tolerance). These interactions (see Figure 1) are classified as follows:

1. **Reification**: initiated by the base level to provide information to the meta-level.

2. **Introspection**: initiated by the meta-level to obtain information from the base-level.
3. **Behavioral intercession**: initiated by the meta-level to modify the behavior of the base-level.
4. **Structural intercession**: initiated by the meta-level to modify the state of the base-level.

III. FAULT TOLERANT SOLUTION

We are considering n total nodes in distributed system. Total connectivity has to be √n + 1 i.e for every node √n other nodes will maintain replication and one node will maintain reflection.

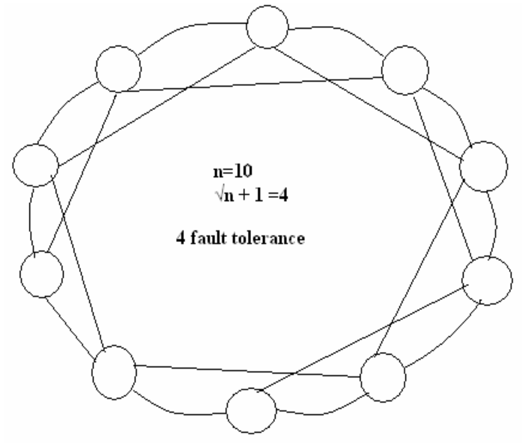

Fig. 2. Topology

The basic idea behind this topology is that in all n nodes [(√n+1)-1] nodes will maintain virtual active replicas and only one will maintain reflection .The use of multiplicity of √n + 1 nodes is to achieve fault tolerance of √n at every step. .

The basic algorithm is described in next section.

A. Basic Algorithm

System specification:

Consider DRE system with sites Si= {1…n}

Connectivity:

N nodes are √n+1 connected.

System Operational modes:

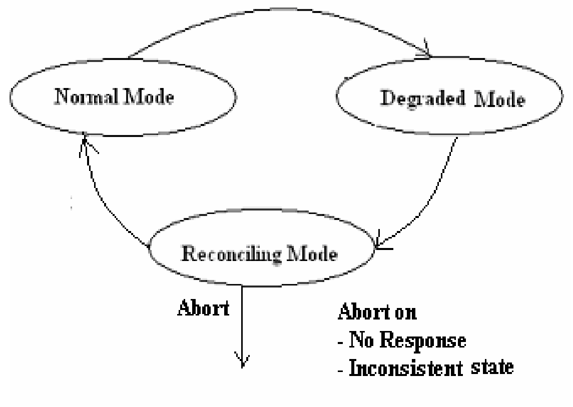

Fig. 3. System Operational Modes

The system modes of operation can be described as the three phases depicted in Fig.3. Parameters involved in the system can be represented by a "Value set". Value set is intermediate stable snap shot or cut of the process in execution. We here by represent Value set in generic way by using subset of alphabets {x,y,z,w}. We are considering group communication system (GCS) as FIFO causal order multicast system. Consistency criteria vary for different modes across all replicas and it depends upon the number of entities in value set. Partial overlap consistency criteria for different operational modes are described below:

Normal mode: The system is said to be in normal mode if at least 75% overlap consistency should be maintained across all replicas and at least one replica has completed its execution (100%). The system can send reply to the client in the normal mode

P1= {x,y,z,w}
P2= {x, y,z ---}
P3= {x, y, z,-}

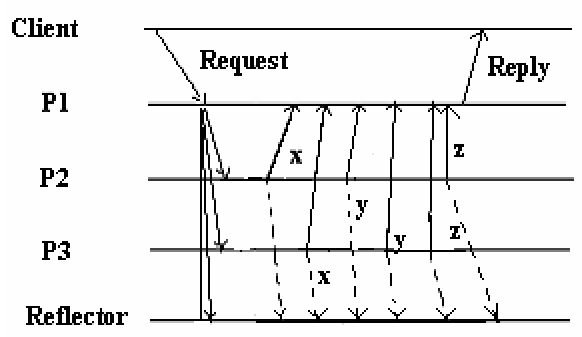

Fig.4. Normal Mode

Degraded Mode:

If there is any type of failure or there is any inconsistency then the system is said to be in degraded mode

P1= {x, y, ----} P1= {x, y, ----}
P2= {x------} OR P2= {x------}
P3= {x, y, a, ---} P3= {} no responce

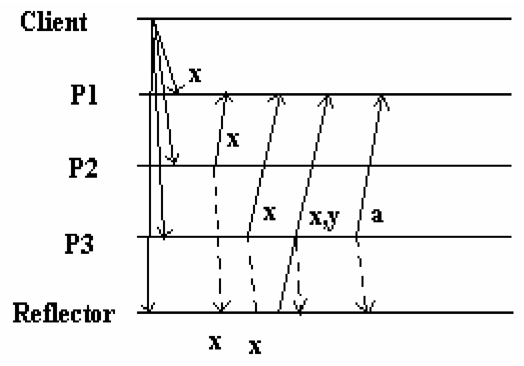

Fig.5.Degraded Mode

In above case the system will go in the degraded mode when P1 and P2 values reach the expected value z and the value of P3 i.e. a is inconsistent with z or P3 fails to respond.

Reconciliation Mode:
Abort either by primary or by reflector on two basic conditions- No response and Inconsistent state.

P1= {x, y, z,-}
P2= {x, y, ---}
P3= {x, y, I,--}

From the degraded mode the system goes into reconciliation mode where the abort message is sent to abort the inconsistent process (P3 in the above example) and a request is sent to another processor from the Queue for which it is mandatory to maintain only 66% consistency to meet its deadline.

P3 is then pushed to the bottom of Queue at primary and reflector as least preferred node for selection in later executions.

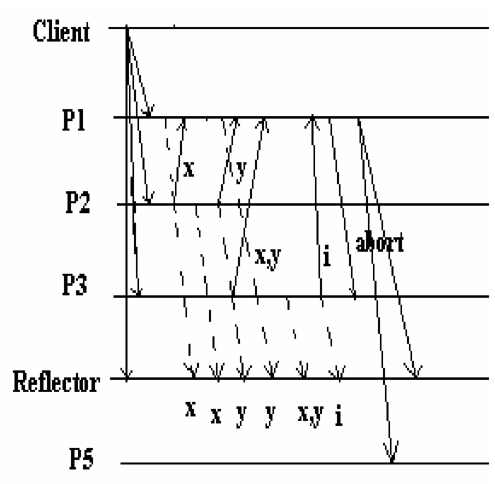

Fig.6. Reconciliation Mode:

- Site Si (S1-----Sn). Every site will maintain Queue consisting (n-1) sites.
- Queue is ordered initially based on hop distance and later changes dynamically depending on reachability and response time.
- When request arrives for Job J at site Si (primary) then

```
If
    Si load > lb (Load threshold)
    Then send request to reflector
        Reflector will choose less loaded site and forward that request.
Else
    Si will remove [|(√n+1)|-1] sites from Queue and send request message to [|(√n+1)|-1] replicas and one reflector .
```

- All sites Sj who receive request from site Si will start execution and intermediate results are sent to Si and reflector.
- Site Si will receive intermediate values (value set) and it calculates Overlap consistency accordingly.
  - If there is no response from any site, then abort message is sent to the node and the queue reconstructed and the node is put in the end.
  - If system is in normal mode (75 % consistency maintained) & any one of the processes finishes its work then Si will send reply to the client
  - If system is in degraded mode (less that 75 %) then hold the results and wait for further reply from replicas.
  - If System is in Reconciliation mode then it will enter in normal mode only when the old replicas maintain 75% consistency and the new replica maintain at least 66 % consistency and there by protecting deadline

Reflector: When it receives reply from other sites it will calculate overlap consistency but if process is giving inconsistent values or it is giving no response then it will send abort message. Reflector will also maintain load and history of each node.

Meta Model:

1. Reification: All members in the group will send all intermediate values to meta level of reflector.
2. Introspection: Reflector will send request periodically to all members to send the information regarding load to decide to threshold.
3. Behavioral intercession: Load threshold will get decided by reflector based on analysis and that threshold will be sent to each member of the group.
4. Structure intercession: If there is no response from any site or the overlap consistency is inconsistent then reflector can send abort message to that site.

B. *Failure Scenarios*

If Site Si (primary) fails
Then reflector will choose another site as primary and will send value set to new primary.
If reflector fails
Then Si (primary) will choose another site as a reflector and send the value set to new reflector.
If any other sites fails
Then primary will choose another site for further processing and system will enter in reconciliation mode. The failed site is pushed to the bottom of Queue at primary and reflector as least preferred node for selection in later executions.

IV. CONCLUSION AND FURTHER WORK

This paper proposes a new algorithm for fault tolerance in DRE Systems using combination of reflection and replication. This algorithm is based on partial overlap system consistency criteria on which the system operational modes are based (Normal, Degraded and Reconciliation.).The proposed algorithm has √n concurrent executions and one reflector per real time process execution.

The following table gives comparison of various techniques for fault tolerance:

Approach	Advantage	Disadvantage
Graph based scheduling/distribution heuristics	-Redundant hardware is not required. -Due to use of active redundancy computation Explicit replication is not required.	-Complex
CORBA-Based Fault tolerance approach	- Support for highly available systems - End-to-end predictable behavior for requests	-Excessive overhead for embedded systems. -Overly complex & difficult to Integrate.
Middle ware base Semi active replication	-Provides support for active & passive replicas. -Cross version mapping.	-Weaker consistency model. -Single point failure at middleware as replication manager and synchronizer.
Replica Communicator	-No overhead on non replica clients. -Self configuration of replica communication. -Multitier solution for various DRE systems.	-Low performance. -High consistency overhead. -Single point replica communicator failure.
Our Approach	-$\sqrt{n}$ concurrent replica executions and one reflector. -High availability. -Partial overlap based consistency.	-Dynamic reconfigurations. -Need of frequent updates of node status.

Use of $\sqrt{n}$ concurrent replica executions and one reflector in the proposed purely distributed approach is suitable for providing fault tolerant deadlines. The possibility of deadline failure exists in two conditions - a) None of the processes on any replica is complete in the prescribed time and b) the replicas do not complete 66% tasks in the said time. The theoretical probability of such a situation is very low but this needs to be practically tested further for feasibility.

Dynamic reconfiguration of queue for selection avoids repeated selection of highly loaded and non-responsive nodes for further real-time processes executions there by making deadlines criteria easier to meet.

V. References

[1] Paul Rubel, Aniruddha Gokhale, Aaron Paulos ,Matthew Gillen Jaiganesh Balasubramanian Priya Narasimhan,, " fault tolerant approaches for distributed real-time and embedded systems" to be published

[2] Paul Rubel, Joseph Loyall, Richard E. Schantz, Matthew Gillen,"fault tolerance in a multi-layered dre system: a case study" *journal of computers*, vol. 1, no. 6, september 2006

[3] François Taïani, Jean-Charles Fabre, Marc-Olivier Killijian, "Towards Implementing Multi-Layer Reflection for Fault-Tolerance" *,DSN-2003 The Internatl. Conf. on Dependable Systems and Networks* — San Francisco USA June 22nd - 25th, 2003.

[4] Jaiganesh Balasubramanian, Dr. Aniruddha Gokhale, Dr. Douglas C. Schmidt, Dr. Nanbor Wang," Investigating Lightweight Fault Tolerance Strategies for Enterprise Distributed Real-time EmbeddedSystems".Available:http://www.omg.org/news/meetings/workshops/RT_2006_Workshop_CD/01-1_Balasubramanian.pdf

[5] M. Baleani, A. Ferrari, L. Mangeruca, M. Peri, S. Pezzini, and A. Sangiovanni-Vincentelli. Fault-tolerant platforms for automotive safety-critical applications. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'03*, San Jose, USA, November 2003. ACM.

[6] A. Girault, C. Lavarenne, M. Sighireanu and Y. Sorel, "*Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems*," In Proc. of the 21st International Conference on Distributed Computing Systems(ICDCS), Phoenix, USA, April 2001.

[7] http://pop-art.inrialpes.fr/~girault/Projets/FT/ visited 2/1/08

[8] Y.sareal Massively parallel computing system with real real time constraint the " algorithm architecture adequation "methodology .In M*assively Parallel computing System s Conference,* Iachia,Italy ,May 1994

[9] Object Management Group, Fault Tolerant CORB A Specification, OMG Document orbos/99-12-08 edition, December 1999.

[10] Object Management Group, Real-time CORBA Joint Revised Submission, OMG Document orbos/99-02-12 edition, March 1999.

[11] Douglas C. Schmidt, "R&D Advances in Middlewarefor Distributed,Real-time,and Embedded Systems," Communications of the ACM special issue on Middleware, vol. 45, no. 6, June 2002.

[12] Balachandran Natarajan, Aniruddha S. Gokhale Douglas C. Schmidt

[13] Chris D. Gill," Towards Dependable Real-time and Embedded CORBASystems",Available:http://www.cs.wustl.edu/~schmidt/PDF/WDMS02.pdf

[14] SumantTambe,AniruddhaGokhale,Jaiganesh Balasubramanian,Krishnakumar Balasubramanian,Douglas C. Schmidt "Model-Driven Engineering of Fault Tolerance Solutions in Enterprise Distributed Real-time and Embedded Systems",Available:www.**omg**.org/news/meetings/workshops/RT_**2006**_Workshop_CD/04-3-Tambe.pdf

[15] Jean-Charles Fabre" A Metaobject Architecture for Fault Tolerant Distributed Systems"

[16] Douglas C. Schmidt," R&D Advances in Middleware for Distributed Real-time and Embedded Systems: Communications of the ACM special issue on Middleware, vol. 45, no. 6, June 2002.

[17] Budhiraja, N., Marzullo, K., Scneider, F., Toueg, S.: 8. ACM Press, Frontier Series. In: The Primary-Backup Approach. (S.J. Mullender Ed.) (1993)

VI. Biographies

Archana Kale has received B. E degree in Computer Engineering from pune university with first class in year 1992 , M.Tech (CSE) from IIT Bombay in year 2001. Currently working as H.O.D and Assistant Professor in department of Information Technology at Thadomal Shahani Engineering college Bandra (W) and has a teaching experience of 14 years.

Ujwala Bharambe has received B.Tech IT degree from S.N.D.T university with first class in year 2001. Currently pursuing M.E in Computer engineering, university of Mumbai. She has more than 5 years of experience in teaching. Currently working as lecturer in department of I.T at Thadomal shahani engineering college.