Using FTT-Ethernet for the coordinated dispatching of tasks and messages for node replication

Sinisa Derasevic, Julián Proenza, Manuel Barranco
DMI, Universitat de les Illes Balears, Spain
sinishadj@gmail.com, julian.proenza@uib.es, manuel.barranco@uib.es

***Abstract*—The *Flexible Time Triggered* (FTT) paradigm provides online flexible scheduling for distributed embedded systems but it does not present adequate fault tolerance mechanisms so as to reach a very high reliability. Adding the adequate fault tolerance mechanisms to FTT-based architectures would open room for adaptive yet highly dependable systems. In this work we present a fault-tolerant system architecture for control applications that adds a node replication scheme with voting on top of an FTT-based system. Using a previously proposed network-centric approach we show how to coordinate the execution of the different phases for a typical control application in our system architecture, i.e. we show how to trigger the execution of tasks in node replicas and the transmission of messages in the communication channel, using the underlying FTT protocol. At the end, we demonstrate how to apply this idea of coordinated dispatching to one concrete control application, ball-on-plate.**

I. Introduction

This work is done within the scope of the project called FT4FTT. The aim of the project is to provide fault tolerance to Distributed Embedded Systems (DES) that use the FTT-Ethernet communication paradigm [1]. Doing so would allow to combine the flexible scheduling features of FTT with the high reliability provided by fault tolerance mechanisms, thereby opening room for adaptive yet highly dependable systems. In this project we take advantage of the features provided by FTT to simplify the design of the fault tolerant mechanisms we are devising.

Node replication is widely used for tolerating permanent hardware faults in the nodes. We use active replication [2], i.e. all replicas of a specific node are identical hardware components executing identical software, thus, if these node replicas receive the same input data, they will produce the same output data. The same input data is provided by a consistency protocol based on [3]. The voting, performed on the outputs, compensates for the erroneous output values if at most a minority of node replicas are faulty and produce these errors. This is known as *error compensation* [4]. Our scheme treats transient hardware faults as if they were permanent ones. We make no attempt to tolerate software (design) faults, even though there is no obstacle preventing us form using design diversity for the nodes' software instead of active replication.

In this paper our target are control applications. A conceptual control system executes 3 phases carried out by a *Sensory subsystem* that collects data from the plant; a *Controller subsystem* that processes collected data and calculates, by means of a control law, the action needed to be taken; and an *Actuation subsystem* that performs the action set by the controller subsystem. In our fault-tolerant system architecture the nodes of the controller subsystem are replicated. Therefore the sensory subsystem consists of replicated sensors, each of which is connected to one of the node replicas, whereas the actuator subsystem is composed of one or more actuators connected to all node replicas. We will assume that actuators execute an additional *output consolidation* [5] phase, i.e. that each actuator collects the actuation commands from the node replicas and then typically vote to obtain the final actuation command. Actuator replication has not been considered for this work, but our system architecture does not impede its use.

For a control application running on a system based on node replication, it is necessary to coordinate through the different control phases the dispatching of the tasks to be executed and the messages they need to exchange. A possible approach to do so on top of FTT was presented in [6] for the CAMBADA robots [7]. However, it uses FTT-CAN [8] and not FTT-Ethernet, and presents the following limitations: (1) it does not support a node replication scheme with voting for tolerating faults; and (2) the network subsystem is aware of the application (FTT was modified to convey application-specific knowledge), which provokes an undesirable interdependence between both of them. Thus, the current paper proposes how to take advantage from FTT-Ethernet specific features in order to jointly dispatch tasks and messages in a fault-tolerant control system that uses node redundancy, while keeping the independence of the communication subsystem from the application.

Section II presents the system architecture and organization. Section III identifies all the tasks and messages for a typical control application in our system architecture. Section IV shows how to dispatch the identified tasks in the node replicas and the messages in the communication channel using the FTT-Ethernet specific features. Section V discusses how to apply the joint dispatching approach for the ball-on-plate control application [9]. Finally, section VI concludes the paper.

II. System architecture and organization

As shown in Figure 1, our system architecture consists of:

- *Nodes*. The nodes are the processing units, they can be used for different purposes and some of them can be replicated. Among all replicated nodes this work focuses on the ones that execute tasks related to the control application and the fault-tolerance actions, i.e. the ones whose tasks interact with the sensors and actuators, carry out the control law and do the different voting actions. Nodes communicate among themselves using FTT, which is a master/multi-slave communication paradigm where one node, called the FTT-Master, controls the communication of multiple slaves. Communication is

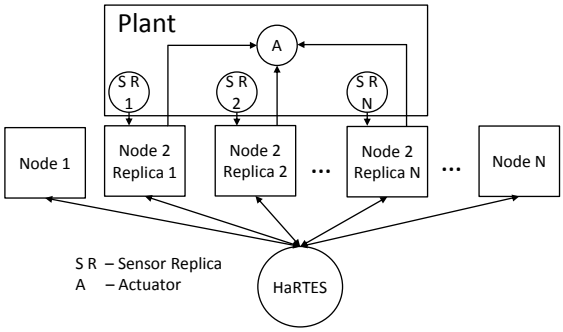

Fig. 1. System Architecture.

divided into fixed-size rounds called *Elementary Cycles* (ECs). Each EC is further divided into a synchronous window, which conveys periodic messages, and an asynchronous window for conveying aperiodic messages. Each EC is initiated by the FTT-Master that broadcasts a special control message, called *Trigger Message* (TM), which instructs which slave should transmit which message, if any, in the current EC.

• *HaRTES*. Nodes are connected to a special switch with the FTT-Master integrated inside, called *Hard Real Time Etherent Switch* (HaRTES) [10]. This switch is enhanced for preventing error propagation and for enforcing incorrect computation failure semantics of the node replicas from the point of view of the other node replicas as needed by the voting procedure. This switch transforms wrong values in the time domain into omissions and prevents wrong values in the value domain only in some cases (two-faced behaviours and impersonations). More details of how these enhancements are done can be found in [11]. According to this, the voting procedure will receive from each replica either no value, incorrect value, or correct value. Then, it will compensate faults of a minority of replicas that produce incorrect or no values. The switch is a *single point of failure* in our system. How to to solve this problem by replicating the switch is described in [12] and is out of the scope of this paper. Therefore, in this work we show only one switch, but assume it is replicated.

• *Plant*. The plant includes the entities of the controlled system which interact with the node replicas and vice versa. These entities are: *a*) *the sensor replicas*, each of which is connected to one of the node replicas *b*) *and the actuator subsystem*, which includes one or more actuators connected to all of the node replicas.

III. Control application phases

Classic control applications repeat the sequential cycle *sense-control-actuate* periodically with a period called *sampling period*. However, our system architecture with replication scheme and voting adds some additional complexity, and therefore we divide this periodic cycle further phases. As shown in Figure 2, these phases are the following ones.

• *Sense*. Each sensor replica produces a value showing the current plant state and transfers it to the linked node replica.

• *Message exchange of sensor values*. Each node replica sends, by means of FTT-Ethernet, the value it obtains from its sensor to the other node replicas. Note that, as explained later, it is expected that all replicas provide exactly the same output. Thus, it is necessary that replicas exchange the sensor values (and then vote on them) in order to provide the control algorithm executed in each replica with the same input.

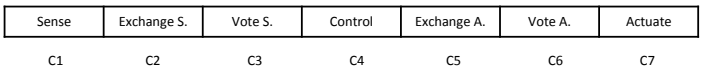

Fig. 2. Phases of the Control Application Cycle.

• *Voting on sensor values*. Each node replica votes on all sensor values, i.e. the sensor values received from the other replicas and the local sensor value obtained from the attached sensor replica. As a result, replicas produce a *consensus* sensor value. Depending on the value measured by the sensors, different kinds of matchings are possible by the voting procedure [13]. Among them the most relevant ones for control applications are the *exact match*, when bit-by-bit identical data is expected, and the *numeric match*, when small differences are acceptable.

• *Control*. Each node replica uses the consensus sensor value and calculates an actuation value. In our case study we consider that a PID controller [14] is used as control algorithm.

• *Message exchange of actuation values*. Each node replica sends its obtained actuation value to the other node replicas by means of FTT-Ethernet.

• *Voting on actuation values*. Each node replica votes on all actuation values, i.e. the actuation values received from the other node replicas and the locally calculated actuation value. As a result, replicas produce a *consensus* actuation value. For this case the match used by the voting procedure is *exact match*, since all of the node replicas are expected to produce exactly the same value (active replication).

• *Actuate*. Each node replica sends its actuation value to an actuator device. Typically an output consolidation of the received values [5] is done in the actuator afterwards, and finally the actuator device performs its actuation.

Note that the sum of the Worst Case Execution Time (WCET) of each phase (C_i in Figure 2) has to be less than or equal to the sampling period (T), in order not to have a negative impact on the control performance, i.e. $\sum_{i=1}^{7} Ci < T$.

IV. Using FTT-Ethernet for coordinated dispatching of tasks and messages

In this work we propose how to jointly dispatch both tasks related to the control application and the voting mechanism, and the periodic messages related to these tasks, i.e. the messages used in the *message exchange of sensor values* and the *message exchange of actuation values* phases.

Specifically, to trigger the periodic messages we propose using the service already provided by the Trigger Message (TM). For that, it is necessary to build a static off-line schedule of these messages and, then, to include the results of this schedule within the *Synchronous Requirements Table* (SRT). Figure 3 shows an example of the SRT after including these results. Note that all messages' time parameters, i.e. period, deadline and offset (release instant delay) have to be expressed in EC multiples. Also note that for these periodic messages we do not use the main feature of FTT, which is *flexibility*, i.e.

FTT-ID	Source	Destination	Period (#EC)	Deadline (#EC)	Offset (#EC)	Description
1	Node Replica 1	Other Node Replicas	Sampling Period	D1	C1	Node Replica 1 sends its sensor value
2	Node Replica 2	Other Node Replicas	Sampling Period	D1	C1	Node Replica 2 sends its sensor value
...						
N	Node Replica N	Other Node Replicas	Sampling Period	D1	C1	Node Replica N sends its sensor value
N+1	Node Replica 1	Other Node Replicas	Sampling Period	D2	∑Ci, i=1-4	Node Replica 1 sends its actuation value
N+2	Node Replica 2	Other Node Replicas	Sampling Period	D2	∑Ci, i=1-4	Node Replica 2 sends its actuation value
...						
2N	Node Replica N	Other Node Replicas	Sampling Period	D2	∑Ci, i=1-4	Node Replica N sends its actuation value

Fig. 3. Synchronous Requirements Table (SRT).

the ability to change the communication parameters on the fly. This is because we need to guarantee that the messages related to the fault-tolerant execution of the control application are schedulable. However, this does not prevent us from including in SRT other traffic that uses the flexibility features of FTT.

For dispatching the tasks we also propose to use the TM, but unlike the previous works [6] and [7], we do not modify the TM to contain the information about the dispatching of tasks. This is because we do not want the network subsystem to be aware of the application executed on top of it, but only to know which messages are transmitted and when, thus being consistent with the FTT protocol. Instead, we include an internal counter, *IC*, in each node replica for this purpose. This counter increases each time a TM is received, counting elementary cycles (ECs). When it reaches the sampling period number of ECs it is reset. Specific values of this counter are used for dispatching different tasks in the node replicas. Additionally, note that the TM includes a sequence number the Master increases with each newly sent TM. Thus, we propose to provide each replica with a function that establishes a correspondence between the IC and the TM sequence number. In such a way, if the IC value is affected by a fault, the replica can use the TM sequence number for correcting it.

Note that how to enforce that each replica (node) actually dispatches a given task once it receives the corresponding TM is already solved in [6]. Thus, here we advocate using the same strategy for this specific issue. Similarly, the way in which the tasks that are not related to control/fault-tolerance must be scheduled to accommodate the tasks triggered by the TM is an issue that is out of the scope of this paper.

Figure 4 shows a possible specification for dispatching tasks, within a replica, based on the IC (the internal counter). Each node replica starts the *sense* phase when its own counter reaches the value 0, starts the *vote on sensor values* phase when the counter reaches the value $C1 + C2$, and so on. Note that each IC is expressed in EC multiples.

One important aspect of our system that has to be taken into account is the EC length, as it determines the system time granularity. The smaller the EC length, the bigger the time granularity is and, then, message and task temporal properties can be expressed more precisely. However, shortening the EC increases the communication and computing overhead, since more TMs are going to be sent per unit of time.

Configuring the appropriate EC length depends on the

Description of task	Trigger value for the counter – IC (#EC)
Sense	0
Vote on Sensor Values	∑Ci, i=1-2
Perform Control Logic	∑Ci, i=1-3
Vote on Actuator Values	∑Ci, i=1-5
Actuate	∑Ci, i=1-6

Fig. 4. Node Replicas Execution Specification.

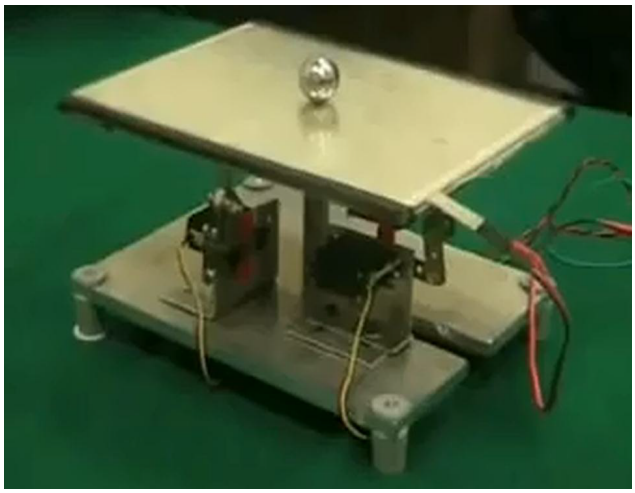

Fig. 5. ReTiS Ball-on-Plate.

number of phases, their WCETs and the sampling period. It has to be ensured that the sum of the phases' WCETs, expressed in EC multiples, is less than or equal to the sampling period. The ideal EC length would be the greatest common divisor of the phases' WCETs.

Finally, it is worth noting that since the TM is used for dispatching tasks and messages, our system relies on a successful transmission/reception of this message. Indications of how the transmission/reception of the TM can be guaranteed are given in [12].

V. Ball-on-Plate example

Next we present a feasibility study on how to configure the control application called *Ball-on-Plate* in our system architecture. The main element of this application is a touchscreen panel that serves as a plate for a ball. This panel is mounted on two servo-motors, one for moving the panel in the x axis and the other for moving it in the y axis. The node, which can be a PC, controls the plant by reading the ball position coordinates from the sensory subsystem (touchscreen) and sending a command to the actuation subsystem (servo-motors). The goal of the system is to keep the ball in the center of the touchscreen. One of these systems developed by the The Real-Time Systems Laboratory (ReTiS) is shown in Figure 5.

In order to configure this application we use the data gathered from the (unfortunately unpublished) experiments conducted at the University of Aveiro on both a ball-on-plate demonstrator and a current HaRTES switch implementation. For this study we assume we have the minimum number of node replicas required for being able to perform a majority voting, which is 3. Likewise we assume that we have 3 replicas

of the sensor, despite the technological problems in order to triplicate the sensors providing the position information in a touchscreen. Such a triplication would be easier to achieve if the position of the ball was obtained from a (redundant) vision system, but we abstract away this technological aspects here.

Current technology used for implementing the HaRTES switch restricts the minimum size for the EC to 1ms, which is what we decided to use in this study. The sampling period of the ball-on-plate demonstrator is $20ms$. The restriction that we defined in Section III has to hold, i.e. the sum of all the phases has to be less than or equal to 20 ECs ($\sum_{i=1}^{7} Ci \leq 20$). Now, according to the data gathered from the experiments we present WCETs of all the control phases and express each control phase in EC multiples:

- *Sense*. This phase lasts between 1 and 2 ms (C1 = 2).
- *Message exchange of sensor values*. This phase lasts 1 elementary cycle (C2 = 1).
- *Voting on sensor values + Control*. These two phases last less than 1ms (C3+C4=1). Since both of these phases are tasks needed to be dispatched in the node replicas and can fit in one EC, we have decided to merge them into a single task (phase).
- *Message exchange of actuation values*. This phase lasts 1 elementary cycle (C5=1).
- *Voting on actuation values + Actuate*. These two phases last more than 2 and less than 3ms (C6+C7=3). Again, we merge these two phases into one for the same reasons used for merging control with voting on sensor values phases. Output consolidation and actuation action will be finished before the next actuation command arrives, and will be performed immediately after receiving the actuation command.

Each message exchange phase fits into a single EC, and we only use a very small percentage of this EC, since the number and the payload of the messages we are exchanging in each phase is very low. This unused space can be utilized by the message exchange for other applications that can coexist with our control application. This also applies to all the ECs we are not using for exchanging messages, i.e. for the phases in which the tasks are executed in the node replicas.

We have calculated that the maximum communication overhead is less than 10%. The computing overhead is assumed to be low since decoding of TM is not a so demanding operation. This means that our EC duration choice is admissible. Also, the sum of all the phases $\sum_{i=1}^{7} Ci = 8 \leq 20$, thus satisfying the main requirement that we imposed at Section III. This example illustrates the feasibility of our approach, showing how to define all the parameters in our fault tolerant system architecture for a concrete control application example.

VI. Conclusion

Distributed embedded control systems based on the Flexible Time Triggered (FTT) paradigm are provided with flexible scheduling, but they lack of appropriate fault-tolerance mechanisms to reach a very high reliability. In particular, since nodes are one of the most unreliable parts of a system, we advocate using a system architecture that includes node replication.

In this paper we propose such an architecture on top of FTT. The main focus of this work is our mechanism for coordinating throughout the different control phases the dispatching of both the tasks to be executed and the messages to be exchanged. We take as a basis a network-centric approach previously proposed to dispatch tasks in a non node-replicated architecture based on FTT-CAN. In our approach, however, we use FTT-Ethernet as the underlying communication technology and, then, we overcome some limitations of that previous work, in order to support node redundancy and keep the independence of the communication subsystem from the application. Finally, we demonstrate the feasibility of our approach showing how to configure a control application on top of it.

In future work we plan to provide a more detailed and formal explanation of how to configure an application that uses this approach. Also we are currently working on how to integrate it with the solutions being proposed in the context of the FT4FTT project.

VII. Acknowledgements

This work was supported by project DPI2011-22992 (Spanish *Ministerio de Economía y Competividad*), by FEDER funding, and by the Portuguese government through FCT grant Serv-CPS PTDC/EEA-AUT/122362/2010. Sinisa Derasevic was supported by a scholarship of the EUROWEB Project (http://www.mrtc.mdh.se/euroweb), which is funded by the Erasmus Mundus Action II programme of the European Commission.

References

[1] P. Pedreiras and A. L, "The Flexible Time-Triggered (FTT) paradigm: An approach to qos management in distributed real-time systems," in *Parallel and Distributed Processing Symposium, Proceedings. International*. IEEE, 2003.

[2] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Distributed Computing Sys., Proc. 20th Int. Conf.*, 2000, pp. 464–474.

[3] G. Rodriguez-Navas and J. Proenza, "A proposal for flexible, real-time and consistent multicast in FTT/HaRTES switched ethernet," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, Sept 2013, pp. 1–4.

[4] P. A. Lee and T. Anderson, "Fault tolerance principles and practice, volume 3 of dependable computing and fault-tolerant systems," 1990.

[5] D. Powell *et al.*, *A generic fault-tolerant architecture for real-time dependable systems*. Springer, 2001.

[6] M. Calha and J. Fonseca, "Adapting FTT-CAN for the joint dispatching of tasks and messages," in *Factory Communication Systems, 2002. 4th IEEE International Workshop on*, pp. 117–124.

[7] V. Silva, R. Marau, L. Almeida, J. Ferreira, M. Calha, P. Pedreiras, and J. Fonseca, "Implementing a distributed sensing and actuation system: The CAMBADA robots case study," in *Emerging Technologies and Factory Automation, ETFA 2005. 10th IEEE Conference on*.

[8] L. Almeida, P. Pedreiras, and J. Fonseca, "The FTT-CAN protocol: why and how," *Industrial Electronics, IEEE Transactions on*, 2002.

[9] S. Awtar, C. Bernard, N. Boklund, A. Master, D. Ueda, and K. Craig, "Mechatronic design of a ball-on-plate balancing system," *Mechatronics*, vol. 12, no. 2, pp. 217–228, Mar. 2002.

[10] R. Santos, "Enhanced Ethernet Switching Tecnology for Adaptibe Hard Real-Time Applications," Ph.D. dissertation, Universida Aveiro, 2010.

[11] A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, and P. Pedreiras, "Towards preventing error propagation in a real-time ethernet switch," in *ETFA, 2013 IEEE 18th Conference on*.

[12] D. Gessner, J. Proenza, M. Barranco, and L. Almeida, "Towards a flexible time-triggered replicated star for ethernet," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conf. on*.

[13] W. Torres-Pomales *et al.*, "Software fault tolerance: A tutorial," 2000.

[14] W. S. Levine, *The control handbook*. CRC press, 1996.