International Journal of Engineering & Technology, 7 (4.44) (2018) 99-104

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Fault Tolerant Dynamic Scheduling on Real Time Hierarchical System: Proposals for Fault Tolerant Mechanism on Safety-Critical System

Catur Wirawan Wijiutomo[1]*, Bambang Riyanto Trilaksono[2], Achmad Imam Kistijantoro[3]*****

[123]School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Jl. Ganesha 10 Bandung 40132, Indonesia
[1]School of computing, Telkom University, Indonesia
**Corresponding author E-mail: catur20@students.itb.ac.id*

Abstract

The paradigm changes from federated architecture to integrated architecture in the real time system introduces a partitioned system to ensure fault isolation and for scheduling the hierarchy scheduling at the global level between partition and local in partition. Integrated architecture based on partitioned system with hierarchical scheduling is referred as real time hierarchical system which is a solution to increase efficiency in terms of hardware cost and size. This approach increasing the complexity of the integration process including the handling of faults. In this paper the authors describe a proposal with three components for dealing with fault tolerant in real time hierarchical systems by handling fault in task level, partition level and distributed level. The contribution of this proposal is the mechanism for building fault tolerant system on real time hierarchical system.

Keywords: *Real time Hierarchical system, real time scheduling, Fault-tolerant system, Integrated Modular Avionics*

1. Introduction

The developments of microprocessor technology impact on increasing computational speed. It gives microprocessor capability to handle more tasks in a certain period. Suddenly, it is wasteful to run single task or application on modern microprocessor, to improve processor utility, researcher introduce using parallel computing to run multiple applications and virtualization to run multiple operating systems on one platform. Although this method initially applied to the needs of the data center, this development also affected the real time system with embedded processor which experienced a change in the development paradigm.

Real time system starts to be developed based on integrated system which runs several applications in one shared hardware platform. Previously real time system uses separate dedicated hardware for specific function. Each unit communicate with each other with a communication bus on dedicated specific link, this configuration is referred to as federated architecture, this architecture could create problem once the real time system grows with many dedicated hardware and communication it could become more complex. The integrated architecture is introduced to mitigate this complexity. This paradigm change is illustrated in the Figure 1, in this picture, the architectures of functions that build an application distributed using own hardware communicate with common bus changed to share a common hardware communicate within hardware via OS mechanism and run on exclusive partition.

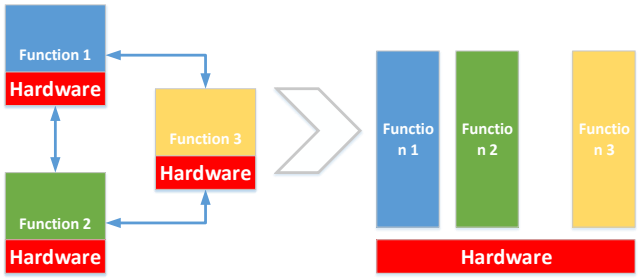

Figure 1 Paradigm Change

Despite Federated architecture drawbacks, it provides isolation against fault with exclusive dedicated hardware in each unit, faults that occur on one unit will not spread to other units so that it is naturally a fault containment. But with the increasing demands for features, capabilities and application certification, this architecture has become more complex and has led to proposals to switch to integrated architectures which several software application units run on shared hardware platform. One of the applications of this architecture is the avionics system which is referred to as integrated modular avionics [1] and automotive called AUTOSAR [2]. while reducing the complexity on hardware used and communication bus, the integrated architecture also introduces challenges related to resource sharing, scheduling and fault tolerance.

The problem of resource sharing and scheduling between real time applications on shared hardware platform needs to be made strictly related to CPU time (temporal) and memory resources (spatial). This causes integrated architecture to be implemented in a partitioned scheme where each application is run on a partition that guarantees temporal and spatial isolation with other partitions. This

approach forms a hierarchy of scheduling) which consists of global level (between partitions) and local level (between tasks in the partition). the unique approach of scheduling application in real time system on integrated architecture that form hierarchy to isolate application on shared hardware platform is called real time hierarchical system[3].

Although integrated architecture provides resource efficiency, the use of such architecture in hierarchical scheduling manner has the need for fault tolerance. Several fault tolerant scheduling studies have handled partitioned scheme but are limited to recovery of task [4] and backup partitions [5] for a single processor platform, so that there are still opportunities to develop method for real time hierarchical systems in some special cases or wider scope.

Fault tolerant on the real time hierarchical system is the focus of this paper, although the relationship between components is more integrated and uses sharing platforms but it is proper that the real time task on the system needs to be functional within a constraint deadline, so that the fault tolerant method commonly used in conventional real time systems cannot be fully implemented. This paper give insight on some proposal on fault tolerant scheduling method for real time hierarchical systems that can handle potential faults at each level of hierarchy.

2. Related Work

Some research related to the fault tolerant method on partitioned systems. The recent one is conduct by Hyun that add recovery job to the partition [4]. The research proposed a new framework for real time hierarchical system with the fault tolerant property and add scheduling analysis to monotonic fault tolerance rate in the periodic resource model. There is an assumption that the "fault manager" can detect faults that occur on the component. The default model is offered when the fault task is detected at the end of the execution of the task. The task process is recovered using a task recovery scheme such as re-execution, forward recovery, checkpointing. The Periodic task defines its own recovery scheme and uses backup tasks based on available time.

In addition, research from Jin developed a resource model to support primary and backup partition [5]. With the aim of deploying backup partition and being able to handle failover from primary partition when fault software is detected. in this research the task in a partition is divided into Context Dependent task (CDT) and Context Independent Task (CIT). CIT must always run because this task must know the trace of the current situation while the CIT is only active when the primary file is. Fault is detected by event deadlines and the absence of heartbeat. In the backup partition tasks are divided into Context Dependent Task (CDT) and Context Independent Task (CIT). The combination of CIT and CDT in the backup partition is the total workload in the backup partition. In this fault tolerant mechanism, there are modes that state the conditions, including Primary (no error) (in the backup partition only CDT that runs), Recovery (error detected), Backup (error happen and primary stop) (Workload backup partition has been executed).

This paper based on the scheduling real time problem form Liu Layland [15] model which discusses multi task scheduling on the hard-real time system up to Guasqe [16] which states that arbitrary scheduling at the global level can guarantee the task level below it on real time hierarchical system. The author tries to see the correlation between the oldest and most recent papers and the following papers such as periodic resource for compositional real time guarantees [17]and compositional real time scheduling framework [18]. Furthermore, based on the scheduling model in the paper and the fault tolerant method studied, several proposals can be designed for the fault tolerant scenario in the real time hierarchical system

3. Real Time Hierarchical System

Real time hierarchical system is used in hard real time embedded computer that run safety-critical function. such as space & defense, process control and automotive fields. The high level of safety demands that not only the computational output is correct but also meets the computational time required[6], [7].

Real time hierarchical system designed to run several tasks sets where each task has deadline timing constraint. Based on the impact of failure, the task can be divided as soft real time system and hard real time system. soft is when the condition does not meet the deadline usually appear form of a lag or inconvenience can be tolerated while hard is when failure to meet the deadline will cause disaster and loss.

3.1. Real Time Scheduling

Real time scheduling problem is a specific scheduling problem is to ensure the T tasks can be scheduled in N resources or processors with strict timing constraint. In general, a task in real time system can be described in the form of Gantt chart which describes the nature of a task

Main parameters can be listed, those include:

- a_i arrival time is when task arrive in CPU and ready to execute
- s_i start time is when task is executed
- c_i computation time is when CPU run task
- f_i finishing time is when CPU stop running task
- d_i deadline is constraint that need task to be finish.

Derived parameters:

- o_i Offset is the time of arrival task to start time
- l_i Laxity or slack time, is the remaining time between the time of completion and the deadline stating the maximum time that an activation of the task can be postponed before the deadline.

on that parameter we can create tuple τ_i

$$\tau_i = \{a_i, s_i, c_i, f_i, d_i\}$$

The nature of the task can be periodic, aperiodic and sporadic. The difference is time interval between instance of task. For periodic tasks the time interval is always the same and is often modeled in terms of T_i so that this periodic Task is quite typical and added to the previous tuple model.

$$\boldsymbol{\tau_i = \{a_i, s_i, c_i, f_i, d_i, T_i\}}$$

whereas aperiodic will have different interval for each instant but have a maximum interval which is not found in sporadic. In addition, there is a pre-emptive nature which states that a task can be delayed or not implemented.

Tasks in the real time system can be scheduled both statically and dynamically. In the case of static scheduling algorithms, the task assignment to the processor and task execution schedule are set from the beginning before the task is executed. the advantage of static scheduling is that the schedule that is run is certain that all tasks can meet the deadline.

Some real time systems require dynamic scheduling algorithms due to changes in resources and tasks. In dynamic scheduling when the task arrives, scheduler sets and determines how tasks are scheduled without disrupting the pre-existing task scheduling.

3.2. Partitioned System

Federated architectures have guaranteed system isolation and faults. Examples of avionics use the distributed function of avionics packaged in a self-contained units that called Line-Replaceable Unit (LRU) [8] In addition, automotive traditional design is based on the concept Federated architectures which integrates hardware and software on Electronic Control Unit (ECU) independently based on loosely interconnected functions [2] This component is connected by certain bus and works together by exchanging messages. Although quite effective, this design is not efficient when adding units need each bus or connection for each function that must be added to each additional unit.

The Integrated architecture uses a high integrity partitioned environment that is used to put several functions with different levels of safety in a shared hardware platform. This provides benefits in terms of weight and power because computing resources can be used more efficiently. In terms of implementation, avionics itself is divided into open Integrated Modular Avionics (IMA), which use open interface standard which is available in public domains while closed IMA uses customizable proprietary interfaces [8].

Integrated architecture development adds to the responsibility of the system integrator because several system functionalities will reach a higher level of integration than federated architecture. Such integration level provides some additions to system optimization but also adds an additional layer to the complexity of integration. Software components can be supplied from a variety of sources, which are then integrated into the hardware text platform that is the same or physically distributed and can be moved from one CPU to another without functional loss with integrated architecture so that some functions can be combined in one integrated ECU/LRU or can be distributed in several integrated ECU/LRU.

Regarding the complexity and high level of integration transition from federated systems to IMA, the system integrator must be able to fulfil the responsibilities for integration tools and processes including. Increased interface definition and management, resource allocation management and system configuration analysis. Resource allocation issues related to partitions with different critical levels, hierarchical schedules, partition allocation for cores and generation of global schedules. Real time system need to use applications with different critical levels that each component with different dependability and real time constraints are integrated into the shared computing platform. The functional critical and different laying on one sharing platform are non-functional things such as: decrease in cost, volume, weight and power consumption.

3.3. Partitioned System Implementation

In terms partitioned system, virtualization software techniques provide partitioned architecture with temporal and spatial isolation. The purposes of a partitioned system are isolation of the function and fault containment. The partitioned system for the real time system itself can be obtained using virtualization techniques with hypervisor which will provide an environment that divides physical resources in a virtual CPU to execute some independent execution environment as it should in native hardware.

Time partitioning and isolation of memory resources in ARINC 653 have similarities with virtualization techniques. So, there is a convenient to combine these two technologies. The current implementation has two approaches and this approach has directions that are opposite to each other [9].

The first approach is using the current hypervisor technology implementation of the hypervisor in virtualization to adapt the technology to a safety-critical environment to meet the ARINC 653 standard by applying DO-178B.

The second approach uses proprietary RTOS technology for safety-critical application. for example, by adding the concept of virtualization on ARINC 653 RTOS.

As for these two approaches, almost all vendors real time operating systems (RTOS), such as Windriver and LynxWorks, use the second approach by adding existing RTOS solutions with the hypervisor's ability to RTOS products. Windriver with VxWorks 653 is adding the ARINC 653 extension based on virtualization on the VxWorks RTOS. This is because the focus for conventional virtualization solutions is different from the real time system, including:

- Virtualization is used for CPU utilization needs to run multiple OS
- Virtualization technology generally focuses on improving performance throughput
- Hypervisor generally do not provide communication between partitions.
- The DO-178B requirement is not fulfilled in existing virtualization technology.

On the paper [9] there is an opportunity to build ARINC 653 on the existing hypervisor. One of them is using Xen, Xen is an open source hypervisor with low level system software that manages virtual machines. Xen runs directly above hardware and provides a virtual view of the machine to the domain. One domain called domain 0 has special privileges to access hardware and set another domain. There is enough research related to the implementation of Xen for the real time system [10], [11] although it is not yet widely used in safety-critical applications.

4. Fault Tolerant Computer System

Fault, error, and failure are words commonly used in research related to fault tolerant computing system [12]. According to Laprie fault is damage or defect in hardware or software that can cause the system to state incorrect. an error is part of the system state which can cause a fault that ends in failure. Failure is a state of a system that cannot function properly or in other words also called a system that has failed.

When designing fault tolerant system, a fault tolerant method is applied to handle one or several types of faults. An efficient method to increase the fault tolerant is to use redundancy by providing backups during failures. This redundancy in the real time system is generally classified in two scheme hardware-based redundancy and time-based redundancy [13]

Hardware-based redundancy or spatial-based redundancy develops in several techniques such as hardware replication, stand-by sparing, DMR (Dual Modular redundancy) and TMR (Tripe Modular Redundancy) [14]. In general, this method works by copying the execution to additional hardware, if the primary hardware is experiencing a fault, then the hardware backup can handle the function of the primary. although it is effective in handling permanent faults, this technique has a shortage of hardware overhead cost and additional power consumption.

Software-based redundancy or time-based redundancy which is pretty much used, including rollback recovery and re-execution [13], is to make serial copy-executions on the same hardware where the copied task is executed. in the absence of additional hardware, additional costs of hardware are not an issue. the weakness of the technique is that it is not effective for permanent faults or transient faults with long duration. with the addition of time when this method is run can cause the task to fail to meet the time constraint.

On the purpose of measuring system performance against faults, two criteria are defined: fault coverage and computational integrity. Default coverage is measured based on qualitative and quantitative parameters. Quantitative parameters specify the type of fault that can be handled by the system. The requirement that the system can handle faults if the system can do recovery when the type of fault occurs. In other words, quantitative parameters provide a measurement of how well the protection mechanism works for each type of fault. Computational integrity is measured based on three main parameters: (1) computational accuracy, (2) reconfiguration time which is the time needed to isolate fault and recovery, and (3) protection in critical states.

4.1. Partition Resource Model

Partition resource model is model for partitions behaviour on real time hierarchical system. This model is then used for mathematical verification and simulation. The model resource partition in this paper is derived from the periodic resource model [17] Periodic resource models used on this paper because of the nature of the partition which limits the task to a temporal limit can be equalized with periodic resources.

The periodic resource model can be equalized as

$$\boldsymbol{M} = \{\boldsymbol{W}, \boldsymbol{R}, \boldsymbol{A}\}$$

W is workload that will use resources, R is a resource, and A is the algorithm for scheduling policy that defines how workload will uses resources. Workload itself is a collection of tasks that can be derived from commonly found task models. in this paper the definition of task is used in equation

$$\boldsymbol{W} = \{\boldsymbol{\tau_1}, \boldsymbol{\tau_2}, \ldots, \boldsymbol{\tau_n}\}$$

The scheduling model for can be equalized as $\boldsymbol{M} = \{\boldsymbol{W}, \boldsymbol{R}, \boldsymbol{A}\}$ is called schedulable if a periodic set of workloads W can be scheduled with algorithm A with resource R.

The periodic model itself describes a partitioned resource that guarantees the allocation of $\boldsymbol{\Theta}$ units every $\boldsymbol{\Pi}$ unit period, partitioning resources can be modelled as

$$\boldsymbol{\Gamma} = (\boldsymbol{\Pi}, \boldsymbol{\Theta})$$

5. Fault Tolerant Hierarchical Scheduling

The fault tolerant scheduling on the real time hierarchical system becomes our motivation to design a fault tolerant algorithm which the author calls a fault tolerant hierarchical scheduling (FTHS). FTHS is a fault tolerant scheduling algorithm for real time hierarchical system with the composition of $A_1, A_2, A_3$by adding extra level from usual hierarchical system. each hierarchical level, level 1 in the task scope same as local level, level in 2 partitions same as global level, level 3 which is added that is system scope/distributed. this FTHS can be stated in the scheduling model $M = \{W, R, A\}$ with A is a set of fault tolerant feasible algorithms, which are $A_1, A_2, A_3 \in A$.

5.1. First Component (A1)

The First Component of The fault tolerant mechanism is to handle possible faults at the task level that can apply the fault tolerant method. The task dependencies on this partitioned system can be placed on one partition or connected to each other in different partitions with a communication mechanism between tasks as shown in the Figure 2

One of the fault tolerant techniques considered is DFTS [19]by measuring the critical level based on the task utility against the period and laxity available [4]

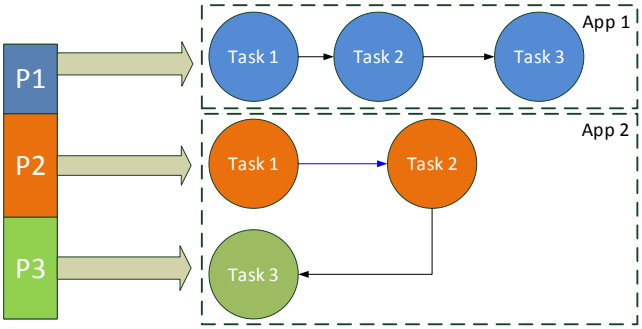

Figure 2 Task within Partition

on this first proposal the utility function is modified by measuring the level of criticality of tasks in the task set by calculating utilization not based on tasks but based on the resources provided on the partition where the task is located.

$$U_t i = \frac{t_i}{R_i}$$

If the value of utilization is closer to 1 and near threshold, then it will be classified as critical. because the value of task utility means that there is almost no laxity or spare time. A task with high utility would not allow job recovery if a fault occurs due to a narrow spare. In addition to utilization also sought a threshold value by calculating the laxity properties that exist in all tasks to become additional properties of task classification. After obtaining task classes based on utilization and threshold then used task replication in critical tasks and recovery job in non-critical tasks by utilizing available laxity. That shown on Figure 3

Data: τ_i
Result: S_{task}, a_{ft}
initialization;
while $\tau_i.t = end$ **do**
  read task;
  if *task.weight* $>$ *threshold* **then**
    set backup.task;
  else if *task.weight* $<$ *threshold* & *task.weight* $>$ *lowThreshold* **then**
    set recovery;
  else
    set re-execute;
  end
end

Figure 3 Algorithm on first component

In the last case it occurs if laxity is not available then the task fault is ignored and raise the flag message to partition, this kind of fault will be handled at the partition level by activating partition backup on second component.:

5.2. Second Component (A2)

Each SBC has several applications arranged in partitions that have several tasks. It is possible to communicate between tasks both in partitions and between partitions at the task level as described below in Figure 4 and Figure 5, in figure 4 application is distributed each partition within a SBC. Each application is running on partition with each have primary and backup. In figure 5 the backup component distributed in another SBC this backup component can be communicate with the primary partition.

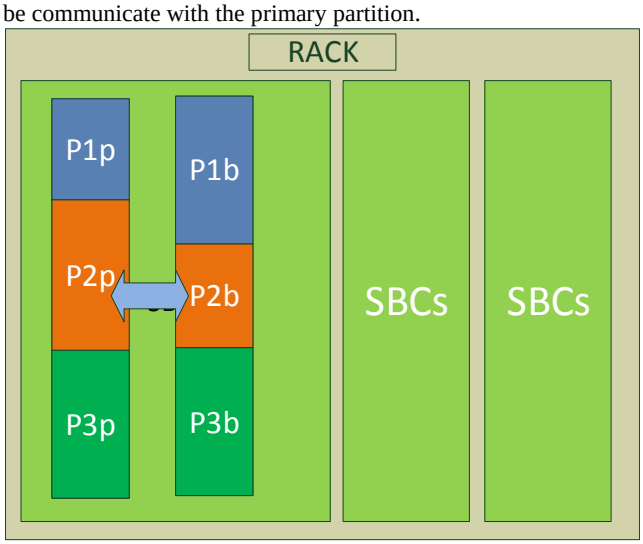

Figure 4 Partition within SBC

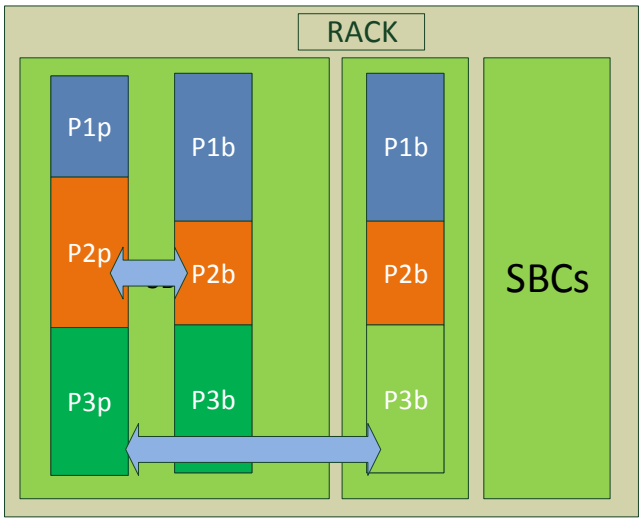

Figure 5 Partition in different partition

The first proposal of the fault tolerant mechanism in this study was to populate partitions into primary and backup application with different scheduling strategies. The primary partition is run with As Soon as Possible (ASAP) and the backup partition is run with As Late as Possible (ALAP). backup partitions can be placed on one SBC as in the Figure 4or different SBC as in the image Figure 5, so that this mechanism is a fault tolerant at the SBC level or partition.

$$C_p = \{\tau_p, R_p, ASAP\}$$

$$C_b = \{\tau_b, R_b, ALAP\}$$

On partitioned system $\Gamma(\Pi, \Theta)$, the ASAP strategy is to arrange scheduling for a task to be done as quickly as possible on the Θ partition unit guarantee, for example for $\Gamma(3.10)$ task unit Π has a minimum offset and produces a supply function. This strategy means that the remaining partition guarantees are obtained for spare time or laxity which can be used for fault tolerant requirements for backup partitions.

In the partition backup, the ALAP strategy is used by delaying if possible the task in the partition unit guarantee. so, the offset task will be very large, and the laxity task will be minimal. This is applied to the partition backup with the intention of optimizing the spare time at the beginning with an assumption that the optimistic fault task does not occur at the beginning of the task running.

As for there are still fault scenarios that are not expected and cannot be handled. for example, faults that occur on both primary partitions and backup, so that the spare time used for backups has run out or backup tasks also fail. So that the handling of faults like this needs to be done by handling at the next level.

5.3. Third Component (A3)

At the system/distributed level, a minimum of dual redundancy can be used which allows resource redundant utilization in integrated distributed system (integrated system with distributed architecture) as shown in Figure 6. In this figure there is another rack that provided to mitigate fault. Even though dual redundancy does not necessarily provide the ideal fault masking, this method is chosen because of minimum physical redundancy method and cost effective. There is some consideration on using this redundant rack that using is as idle rack (passive backup) that backup a whole rack or using it as another resources that can be used (active backup)

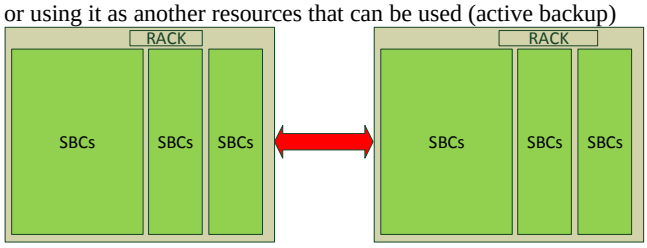

Figure 6 Distributed SBCs

5.4. Integration Configuration

The algorithm scheme is A_1, A_2, A_3 intended to handle faults at each level of real time hierarchical system but the implementation in integrated architectures requires a configuration to be able to perform dynamic resource settings when a fault occurs. The assumption used is that the operating system has provided fault management and fault detection mechanisms such as health management in ARINC 653[9] or signals on POSIX. The mechanism can be used as information and signals that can be forwarded to the scheduler to dynamically change scheduling based on the three proposed algorithms. This is illustrated in the Figure 7

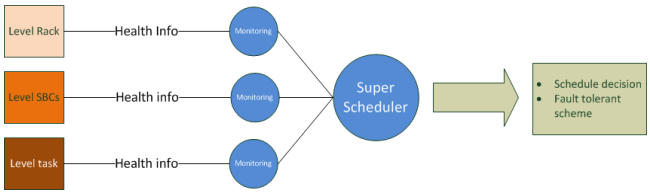

Figure 7 Configuration step

6. Conclusion

These Components and integration configuration are the mechanism of fault tolerant system and the main contribution on this paper. This architecture provides fault tolerant system on real time hierarchical system. the integrated architecture nature is considered by handling fault in each level

The proposed algorithm of A_1, A_2, A_3 is a fault tolerant hierarchical scheduling (FTHS) proposed to be able to handle faults at the task level, partition and system/distributed in the real time hierarchical

system. Even though the fault tolerant [4], [5], [19], [20] scheme has been formulated but still requires a fault management strategy that is connected to scheduler to handle change of task schedule and conduct resource mapping when a fault occurs. This give dynamic behavior to the fault tolerant system of real time hierarchical system.

At the next research stage, the author plans to do a simulation to prove whether the algorithm A_1, A_2, A_3 are a set of algorithms that meets schedulable analysis, schedule feasibility, utility bound and compositional guarantee for real time hierarchical system. As for the fault tolerant performance parameters, it will be based on a certain fault scenario that will be calculated based on error coverage and computational integrity. Regarding implementation in the RTOS environment, the author has access to VxWorks 653 and RT-Xen with target platform which is one of the research value-added opportunities.

Acknowledgement

I would like to give my acknowledgement to Telkom University and LPDP.

References

[1] P. J. Prisaznuk, "Integrated modular avionics," in *Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National*, 1992, pp. 39–45.
[2] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proc. IEEE*, vol. 98, no. 4, pp. 603–620, 2010.
[3] A. Guasque, P. Balbastre, and A. Crespo, "Real-time hierarchical systems with arbitrary scheduling at global level," *J. Syst. Softw.*, vol. 119, pp. 70–86, 2016.
[4] J. Hyun and K. H. Kim, "Fault-tolerant scheduling in hierarchical real-time scheduling framework," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, 2012, pp. 431–436.
[5] H.-W. Jin, "Fault-tolerant hierarchical real-time scheduling with backup partitions on single processor," *ACM SIGBED Rev.*, vol. 10, no. 4, pp. 25–28, 2013.
[6] H. Kopetz, "Real-time systems: design principles for distributed embedded applications," 2011.
[7] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer Science & Business Media, 2011.
[8] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *Digital Avionics Systems Conference, 2007. DASC'07. IEEE/AIAA 26th*, 2007, p. 2--A.
[9] S. H. VanderLeest, "ARINC 653 hypervisor," in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, 2010, p. 5--E.
[10] S. Xi, J. Wilson, C. Lu, and C. Gill, "Rt-xen: Towards real-time hypervisor scheduling in xen," in *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, 2011, pp. 39–48.
[11] S. Xi *et al.*, "Real-time multi-core virtual machine scheduling in xen," in *Embedded Software (EMSOFT), 2014 International Conference on*, 2014, pp. 1–10.
[12] J.-C. Laprie, "Dependable computing and fault-tolerance," *Dig. Pap. FTCS-15*, pp. 2–11, 1985.
[13] Y. Zhang and K. Chakrabarty, "Fault recovery based on checkpointing for hard real-time embedded systems," in *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, 2003, pp. 320–327.
[14] E. Elnozahy, R. Melhem, and D. Mossé, "Energy-efficient duplex and tmr real-time systems," in *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, 2002, pp. 256–266.
[15] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[16] A. Guasque, P. Balbastre, and A. Crespo, "Real-time hierarchical systems with arbitrary scheduling at global level," *J. Syst. Softw.*, 2016.
[17] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, 2003, pp. 2–13.
[18] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, 2004, pp. 57–67.
[19] M. H. Mottaghi and H. R. Zarandi, "DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors," *Microprocess. Microsyst.*, vol. 38, no. 1, pp. 88–97, 2014.
[20] A. A. Bertossi, L. V Mancini, and F. Rossini, "Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 9, pp. 934–945, 1999.