VFAST Transactions on Software Engineering
http://vfast.org/journals/index.php/VTSE@ 2018 ISSN(e): 2309-3978;ISSN(p): 2411-6246
Volume 6, Number 1, January-December, 2018

pp. 42-50

REVIEW PAPER ON FAULT TOLERANT SCHEDULING IN MULTICORE SYSTEM

MUHAMMAD ADEEL ASHRAF[1], SARAH ILYAS[1], MUHAMMAD AHSAN[2]
[1]Department of Computer Science, University of Management and Technology, Punjab, Pakistan
[2]Department of Software Engineering, University of Management and Technology, Punjab, Pakistan
Email: adeel.ashraf@umt.edu.pk

`

ABSTRACT. *In this paper it was discussed about various fault tolerant task scheduling Algorithm for multicore system based on hardware and software. Blend of triple module redundancy and double module redundancy considering Agricultural vulnerability factor other than EDF and LLF scheduling algorithms was used to create hardware based algorithm. Most of the real time systems used shared memory as dominant part. Low overhead software based fault tolerance approach could be implemented at user space level so that it did not require any changes at application level .Redundant multithread processes were used which could detect soft recover from the errors and could recover from them giving low overhead, fast error mechanism recovery and detection. The overhead incurred by this method ranged from 0 to 8% for selected benchmarks. Another system used for scheduling approach in real time systems was hybrid scheduling. Dynamic fault tolerating scheduling gave high feasibility where task critically was used to select the fault recovery method type in order to tolerate maximum no. of faults.*

Keywords*:* Fault tolerant, dynamic scheduling, multicore processor, Earliest deadline first, Task graph, Check pointing.

1. Introduction. Fault Tolerant computing is a science and an art of system computing building that continuously to operate properly in the existence of faults. A fault-tolerant system may have a capability to accept one or more faults and the fault-types are a) Hardware faults (may be transient, intermittent or permanent hardware); b) Design errors of software and hardware, c) Operator errors, d) Externally persuade upsets or physical damage. A general method has been expanded in this field last 30 years, and many numbers of fault-tolerant machines have been designed. A smaller number deal with software, while most dealing with random hardware faults, design and operator faults to varying degrees. Different types of fault which are hardware fault, fault masking, dynamic recovery and software fault. Hardware fault is occurring due to design issue when it is building. Fault masking is a redundancy technique that entirely masks faults surrounded by a set of redundant units (modules). Dynamic fault is necessary when one copy of computation is running at a time and also included self-repair automated. This technique is more hardware efficient generally. Software fault is occurring due to software design fault.

Nowadays semiconductor technology is growing very fast and it's possibility of billions of transistors to have on a small chip. Small transistors are more susceptible to both Permanent and transient faults. Therefore, through expanding the budget of the chip, multicore design is well-liked for enhancing the structure throughput. By increasing the amount of transistors and constricting their sizes, the ratio of Software errors on processors can't be ignored; therefore, the reliability of these types of frameworks has been a prominent between the most necessary complexities as smaller transistors are more at risk to faults. The most important condition in the design process of real time systems are time limitations, energy efficiency throughput. In a single package there are two or more than two independent cores of a multi core processor. A dual core processor means processor has two independent cores and quad core means processor has four cores. Multicore processors have many advantages which are given a) Throughput is high b) Consumption of power is linear c) Processor cores can be utilized efficiently d) Per unit cost provides high performance. IBM ARM

and MPCore Cell are the examples of multicore processors which are employed in the real-time embedded systems. Multicore processors are classified into two parts, which are heterogeneous or homogenous; Most of the existing multicore systems are homogeneous. The main purpose of multicore processor is used to managing the tasks in such a way that the cores can be utilized/used efficiently and effectively. To improve the execution time, the responsibility of a scheduler is that keeping all the cores in processor should busy during the implementation of real time tasks. Faults can also be categorized into different categories which are transient, intermittent and permanent faults. Transient faults are very short time faults and on the basis of duration of the causes and occurrence, it can be distinguished from others fault. These may be occurring due to other sources and external noise. Intermittent fault occurs at interims on account of some hidden deadly fault of the segments like power supply, noise and temperature vacillations. Permanent fault such as wear off of any part which require substitute from the extra part to restore the system functionality. Processes which run on instruction level have low error identification as comparison to hardware level. Multi-threaded programs runs on multi cores processors where sharing memory is common as compared to interrupt or signals generation. For this type of efficient achievement is too much difficult, so here is need to implement different type of deterministic language. For this we can do fault tolerance using redundant running of software in which fake copies which give same output for given input. For this method it uses user library at user level not need of modification at kernel level. In user space the error mechanism should be used to optimize in such a way that memory comparison replicas should perform efficiently. Performance can be increased by using the multiple threads. Two level of scheduling hierarchy soft real time with non-real time tasks is used. Real time requirement supports other tasks and decrease average deadline miss ratio if this method is used. For hardware based algorithm principle there is need to identify the tasks or assigning tasks which are fitting on cores and tolerate processing component failure due to homogeneous or heterogeneous. The transient flaws may be happen in entryway, transistor and a bit even, this is called architectural helplessness variable. This algorithm can be compared to other techniques for fault tolerant a proposed method outflanks DMR and TMR methods. These are most common topologies in computing. When there is a reliability of tasks then there is TMR methods are used as it can only be the mask fault and this type of problem from voter is a faulty. DMR methods is used when two parallel cores running and here is notice the output similarity, if there is indicate of mismatch then does not tolerate the fault and here is need to use other mechanism for error recovery.

Tasks priority algorithm on the basis of time is one of the classical scheduling algorithm. The tasks having short time period then it gives a priority to assign a first core to the tasks and given the next core to task which have second short time. Another scheduling approach which is alternative is Primary Backup (PB) that is a show up between the other methodologies, in which two tasks, is scheduled on two different cores. For result validation an acceptance test is used.

Impact on Society. Multicore processors have many advantages which are given Throughput is high, Consumption of power is linear, Processor cores can be utilized efficiently, per unit cost provides high performance.

2. Related Work. Current processors (UltraSPARC T1), which are put on advanced process technology, are particularly flat to soft errors. Keeping this type of problem in mind, Sun (company) logically planned the UltraSPARC T1 processor with the proper level of security of its on-chip memories. In common, the UltraSPARC T1 secures memory arrays with SED/DED (Single Error Correction / Double Error Detection) or parity protection. Redundant arrays are protected with parity, while non-redundant arrays are protected with Error Correcting Code (ECC). Given table lists the UltraSPARC T1 on-chip memories and its related safety system.

Table 1. On chip memory array protection

Floating point register file.	ECC
Integer register file.	ECC
L1Cache-data instruction	Parity/Retry
L1 instruction cache-tag	Parity/Retry
Instruction TLB	Parity/Retry
Data TLB	Parity/Retry

L1 data cache-data	Parity/Retry
L1 data cache-tag	Parity/Retry
L2 cache-data	Parity/Retry
L2 cach-tag	Parity/Retry
L2 cache-scrubber	Parity/Retry

Within a single memory nibble (4 bits), the Chipkill mechanism1 (UltraSPARC T1 processor's) can correct any fault controlled, and identify any uncorrectable errors contain inside any 2 nibbles. Memory scrubbing is another mechanism applied in the UltraSPARC T1 processor to guarantee for main memory consistency. 4 Memory controllers (Each of the processor of UltraSPARC T1) has a background error scrubber /scanner is used to reduce the incidence of multi-nibble errors. At the International consistency Physics conference, Sun explained that applying power and thermal management features can significantly increase both the reliability and lifetime of the device by up-to 24 times while continuing or civilizing performance of the device. Li et al presented a method called CASP in Multicore environment for independent testing of cores by hardware unit adding. CASP is a unique type of identity test where a system tests itself parallel during normal operation not including any downtime visible to the end-user. Test patterns are stored in nonvolatile storage because it uses flash memory or hard disk and also providing system level support or architecture for testing in multicore system of one or more cores, while the other remaining parts of the system should work normally. ARGUS explains fact that core of the multicore system has only four basic operation which are given 1) Selecting a sequence of instruction how they will execute. 2) Computation which each instruction will perform. 3) Passing data flow results for each instruction. 4) How the memory interaction. Checking 4 all activities and error detection can be here if any error occurs inside a core. Bower et al presented a checker called DIVA, this method detects an error in an instruction and increments a small saturating error counter for every field reconfigurable unit (FRU) used by that instruction. A hard fault in an FRU if the value above threshold error counters for that FRU and thus analyzes the fault. Bell et al have presented that faults are detected by using the redundant execution on a chip of multiprocessor with no impact on performance. Data is stored in a memory and then compare to their similar storage. If there is no identical store or mismatch in store data or address then an error signal is sent to the processor, similarly it exists in Android operating system [17-20].

3. Methodology. Here we discuss different scheduling algorithms and their characteristics for fault tolerant multicore system. These scheduling algorithms are given Hybrid, Non pre-emptive EDF, Checking pointing optimization, Harvesting aware real time, Dynamic voltage and frequency selection, Dynamic fault tolerant, Fault tolerant scheduling based on task criticality, and Task graph scheduling using NABBIT. Here are details of all the above scheduling algorithms.

A. ***Hybrid Scheduling***

Hybrid algorithm is a combination of two algorithms which are DMR and TMR fault tolerance scheduling algorithms both in which by using the AVF (Agricultural Vulnerability Factor) which uses task scheduler. Task scheduler tells the fault occurrence when it's running on the core. Defined each tasks in four-tuple, Ti = (ai, AVFi, di, ci) where ai, AVFi, di, ci is arrival time, architectural vulnerability factor, deadline, and execution time respectively for the each task. For efficient and effective scheduling there should be 1) task should be ended on their time (deadline). 2) One task is assigned to a single core. Algorithm efficiency is measured by core utilization and task total execution time. Utilization of the core is calculated by a mathematical formula which is given below

$$\text{Utilization} = \frac{\sum_{i=1}^{n} \text{Pi}}{N * T}$$

Where Pi is the each core busy time in which it's executing the different tasks, N is total number of cores and T is the total running time of the task. In Hybrid scheduling algorithm to achieve the best result it can execute multiple parallel task. Parallel task execute in such a concurrently way with other tasks. It may be possibility of real time and non-real time tasks executing concurrently. Hybrid scheduling method uses 2 level scheduling policies. One is top level where it used sporadic server and other is bottom level where it used rate monotonic operating system (OS).

B. ***Implementation Method***

1st on the basis of arrival time tasks are sorted in a waiting queue. For running a task in a safe mode there is need to determine the number of cores needs for a task and then compares the architectural vulnerability factor (AVF) to architectural vulnerability factor threshold (AVFthreshold). Task will be executed in TMR mode only when architectural vulnerability factor (AVF) >= architectural vulnerability factor threshold (AVFthreshold) otherwise it will be run in DMR mode.

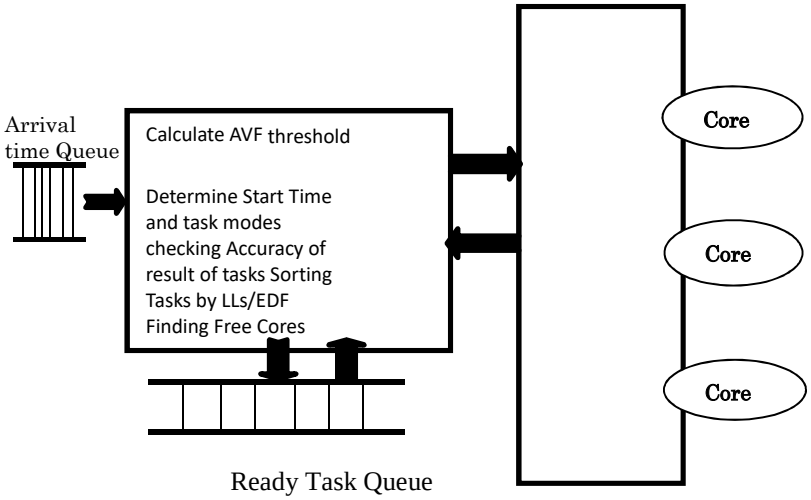

Figure1:- Fault tolerant task scheduling algorithm for multicore system

C. ***Non-preemptive Earliest-Deadine_First***

In this scheduling algorithm have different priority algorithms which are dynamic such as EDF (Earliest-Deadline-First), LL (Least Laxity), LSTF (Least-Slack-Time-First), and MLF (Minimum-Least-First), where on the basis of their specific parameters, the tasks are scheduled and prioritized. Earliest-Deadline-First (EDF) is an optimal technique with a set of preemptive task for a single processor. Total utilization (U) on a set of processor (M single core) and mathematical formula is given below

$$U \ll \frac{M}{2M-1}$$

With M cores it can be considered as boundary for processor (multicore). Task AD (Absolute Deadline) is calculated when task will be included in ready list and it will release. Assigning the core it is only possible when it will be idle, before assigning the core it will check that either task can be scheduled on this idle core or not. If assignment is possible then core status shows it busy else it will remain idle for the next coming task which can be scheduled on it. Here is an example of quad-core processor where Earliest-Deadline-First (EDF) scheduling of tasks on it. From given figure which shows that T2 has lowest absolute deadline so it can be scheduled on P1 first and thus T1 is scheduled on P2, T3 on P3. Competition between T4 and T5 is observed because both will be released on the same time, but the AD (absolute deadline) of T5 is before than T4, therefore here is a need of T5 is scheduled on 1st. Scheduling of the other tasks will be on the idle core same like the above.

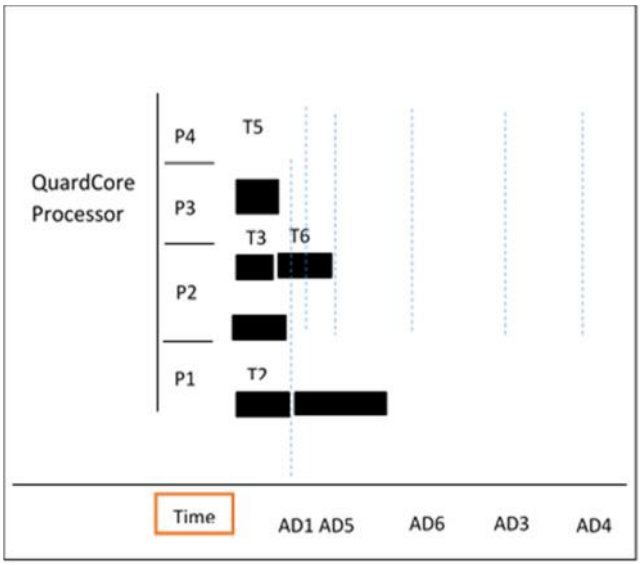

Figure 2:- EDF Task Scheduling of a task set on a quad-core processor

Recovery and re-execution are more convenient when it used checkpoints and can be utilized with soft real time system while redundancy techniques are more convenient with tight deadlines for hard real time systems. Total execution time is increased due to re-execution and checkpoints.

*D. **Ckecking Pointing Optimization***

There is Context switching between applying infrequent and frequent checkpoints for a task in a system. In the presence of faults, re-execution time is decreased due to frequent check pointing whereas task time of execution is increased. . In the presence of faults, there are lower time overheads in infrequent check pointing whereas if fault is identified then cost amount of the re-execution will increase. The below formula shows that m is a optimal number of check points where k are faults to be considered.

$$m = || \sqrt{\frac{K * C}{Cs}} ||$$

Where Cs is saving check points and Cr recovering from checkpoints. With rollback recovery (Cc) the worst time for check pointing of a task can be given by mathematical formula which is given below

$$Cc = (C + m + Cs) + K * (Cs + Cr) + \frac{K * C}{m + 1}$$

Where $(C + m + Cs)$ this equation is used for check pointing where there are faults. This equation $(Cs + Cr) + \frac{K*C}{m+1}$ of recovery fault cost for a single fault.

*E. **Harvesting Aware Real Time Algorithm***

Power management with scheduling of the tasks is an important issue in real time embedded system. To lessen the energy consumption in this paper, author's proposed a technique which is called harvesting aware real time scheduling algorithm, while attainably plan the understanding of alternating tasks within their deadline. This can be done by frequency selection and Dynamic Voltage, running the task inside the speed it must consumes energy according to their need to complete the task on their due date (deadline).

*F. **Dynamic Voltage and Frequency Selection***

In this paper a technique is proposed which should be low complicated nature and task refers on mapping, power management and scheduling for multicore real time system like embedded system with harvesting energy. This technique (Dynamic voltage and frequency selection) totally based on utilization of the CPU. For proposed utilization based algorithm it should combine this method (Dynamic voltage and frequency selection)

to the energy harvesting. This algorithm provides efficient and effective use of energy harvested on the multicore system.

G. ***Dynamic Fault tolrent Algorithm***

This algorithm based on resource allocation time and utilization which used task criticality. Maximum no. of faults in order to tolerate, utilization of the task is used select dynamically the type of fault recovery. Task is divided into two categorize which are non-critical and critical based on criticality parameters of the task. In faulty condition critical tasks will be replicated on different cores to increase on time completion probability. Non-critical are scheduled on one core (single core) and check pointing with a technique which is called fault tolerant rollback recovery. Scheduling (feasibility) rate of dynamic fault tolerant scheduling (DFTS) is higher than the other scheduling fault tolerant.

H. ***Fault tolrent Scheduling Based on task Criticality***

When resources are available then this algorithm selects a technique assign for each task. There should be a care during scheduling is that no one task will be assigned until all the other tasks with higher priorities are assigned (scheduled) and previous faults occurred are tolerated. Criticality threshold is calculated by the scheduler for each task which is currently at 1st position of ready list as ideal core is existing in the system. RAT (resource allocation time) is defined as the time in which one ideal core for on task can be calculated by the following formula.

$$delayi = RAi - Ri$$

Where delay is a time which is wasted in ready list and fault tolerant applies and allocation of the resources for each task. Non-critical task becomes critical when increasing delay. Hardware replication techniques which are apply on critical whereas check pointing techniques are applied on non-critical by scheduler.

I. ***Task Graph Scheduling using NABBIT***

Some information from user side for a task graph such as sink task, task key, successors, and predecessors. Task graph structure captures by task scheduling algorithm after getting the information. On the basis of work stealing this algorithm is built on NABBIT (Task Graph). Keys are assigned to the task in task graph and execution is controlled by hash map. 'INSERT TASK IF ABSENT routine' is used to insert a task on hash map and a call 'GETTASK' later for a created task. To proceed further the runtime have status, join, and notify array for every task. 'INITANDCOMPUTE' function is used in hash map where insertion and creation of the sink task can be done when execution task begins. 'TRYINITCOMPUTE' call is used in 'INITANDCOMPUTE' for immediate predecessors forward and the task starts. 'TRYINITCOMPUTE' function is used in recursive fashion, task graph is expanded by execution and reaches with no dependencies its status is updated as completed and notifying the successors using its notify array. Status of the task is changed into 'COMPLETED' after the final successor in a notify array.

4. Analysis and Scheduling techniques.

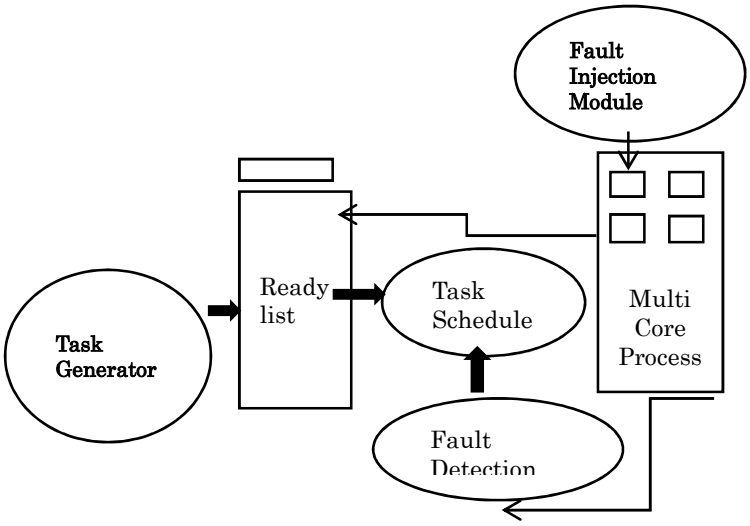

Figure 3:- The block diagram of a system

Table 2. Scheduling algorithms comparison

Scheduling Algorithm	Characteristics
Hybrid Scheduling	This task scheduling algorithm can promise the right execution of running task and decline aggregate time of task execution by expanding the cores while AVF of each task.
Non-Pre-emptive EDF Scheduling	EDF is an optimal algorithm for single processor system with a set of pre-emptive independent tasks.
Check Pointing Optimization	Check pointing decreases re-execution time in the presence of faults, while task execution time is increased.
Harvesting Aware real Time Scheduling Algorithm	This algorithm decreases the energy consumption while attainably plan the arrangement of intermittent tasks.
Dynamic voltage and frequency selection Algorithm	This is the low-multifaceted nature and task assignment mapping, scheduling.
Dynamic fault tolerant	Scheduling feasibility rate of the DFTS algorithm is higher than other fault tolerant scheduling methods.
Fault tolerant scheduling based on task criticality	Task criticality is used to select the type of fault recovery method in order to tolerate the maximum number of faults.
Task graph scheduling using NABBIT	This algorithm can efficiently recover from arbitrary faults that are proportional to the amount of work lost.

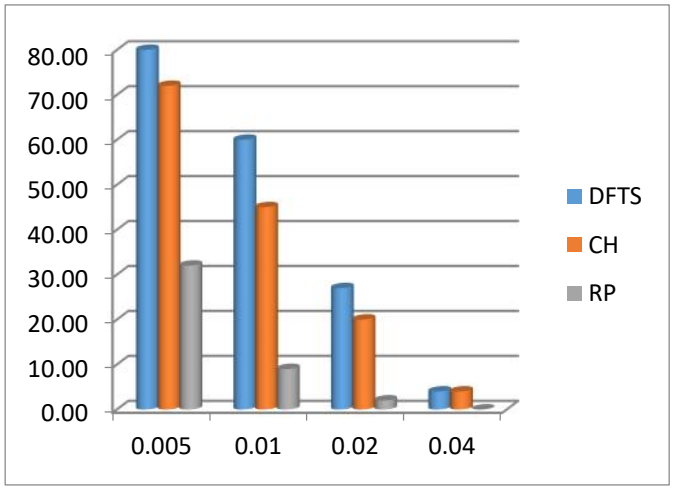

Figure 4:- Feasibility rate of DFTS compared to simple check pointing and replication for quad-core processors regarding doubled fault arrival rates

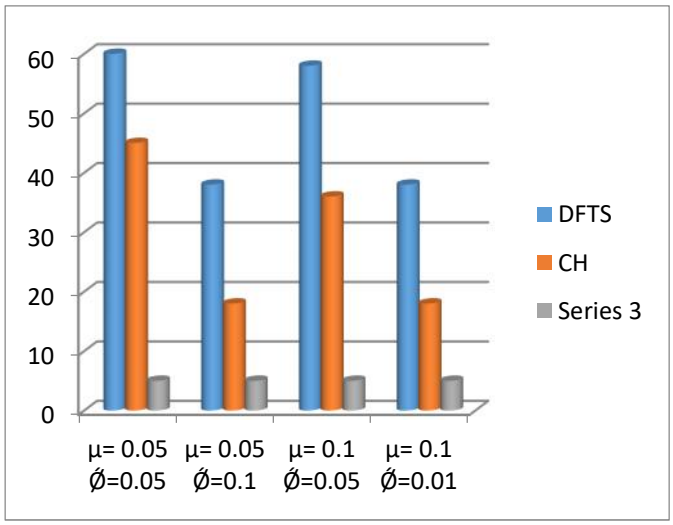

Figure 5:- Feasibility rate of DFTS compared to simple check pointing, and replication by varying the amount of checkpoint saving and recovery cost with constant fault rate (k = 0.01).

Conclusion. Various tolerant algorithms which are applicable on multicore system processor are discussed. Scheduling technique is chosen on the basis of system requirements. Above discussed all the algorithms have some restriction and in future there is need to design scheduling algorithms which are hard real-time which will decrease the time and energy by utilizing speed in a way that task response time is equivalent or less than to other existing technique regardless of the reality on the lesser energy cost consumption.

REFERENCES

[1] Gotoda, S., Ito, M., & Shibata, N. (2012, May). Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)* (pp. 260-267). IEEE Computer Society.

[2] Han, C. C., Shin, K. G., & Wu, J. (2003). A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. *IEEE Transactions on computers*, *52*(3), 362-372.

[3] Tan, P., Shu, J., & Wu, Z. (2010). A hybrid real-time scheduling approach on multi-core architectures. *Journal of software*, *5*(9).

[4] Zhang, Y., & Chakrabarty, K. (2003, November). 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*(pp. 320-327). IEEE.

[5] Shiravi, S., & Salehi, M. E. (2014, May). Fault tolerant task scheduling algorithm for multicore systems. In *Electrical Engineering (ICEE), 2014 22nd Iranian Conference on* (pp. 885-890). IEEE.

[6] Saifullah, A., Li, J., Agrawal, K., Lu, C., & Gill, C. (2013). Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, *49*(4), 404-435.

[7] Davis, R. I., & Burns, A. (2011). A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, *43*(4), 35.

[8] Agrawal, S., & Yadav, R. S. (2009). A Preemption Control Approach For Energy Aware Fault Tolerant Real Time System. In *International Journal of Recent Trends in Engineering*.

[9] Bertogna, M., & Baruah, S. (2010). Limited preemption edf scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, *6*(4), 579-591.

[10] Kurt, M. C., Krishnamoorthy, S., Agrawal, K., & Agrawal, G. (2014, November). Fault-tolerant dynamic task graph scheduling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 719-730). IEEE Press.

[11] Harper, R. E., Lala, J. H., & Deyst, J. J. (1988, June). Fault tolerant parallel processor architecture overview. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995,* (p. 62). IEEE.

[12] Sellers, F. F., Yue, H. M., & Bearnson, L. W. (1968). Error detecting logic for digital computers.

[13] Gizopoulos, D., Psarakis, M., Adve, S. V., Ramachandran, P., Hari, S. K. S., Sorin, D., ... & Vera, X. (2011, March). Architectures for online error detection and recovery in multicore processors. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011* (pp. 1-6). IEEE.

[14] Seo, E., Jeong, J., Park, S., & Lee, J. (2008). Energy efficient scheduling of real-time tasks on multicore processors. *IEEE transactions on parallel and distributed systems*, *19*(11), 1540-1552.

[15] Baker, T. P. (2005). An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel & Distributed Systems*, (8), 760-768.

[16] Lakshmanan, K., Kato, S., & Rajkumar, R. R. (2010, November). Scheduling parallel real-time tasks on multi-core processors. In *2010 31st IEEE Real-Time Systems Symposium* (pp. 259-268). IEEE.

[17] Hussain, M., Al-Haiqi, A., Zaidan, A. A., Zaidan, B. B., Kiah, M., Iqbal, S., ... & Abdulnabi, M. (2018). A security framework for mHealth apps on android platform. *Computers & Security*, *75*, 191-217.

[18] Hussain, M., Zaidan, A. A., Zidan, B. B., Iqbal, S., Ahmed, M. M., Albahri, O. S., & Albahri, A. S. (2018). Conceptual framework for the security of mobile health applications on android platform. *Telematics and Informatics*, *35*(5), 1335-1354.

[19] Zaidan, A. A., Zaidan, B. B., Al-Haiqi, A., Kiah, M. L. M., Hussain, M., & Abdulnabi, M. (2015). Evaluation and selection of open-source EMR software packages based on integrated AHP and TOPSIS. *Journal of biomedical informatics*, *53*, 390-404.

[20] Hussain, M., Al-Haiqi, A., Zaidan, A. A., Zaidan, B. B., Kiah, M. M., Anuar, N. B., & Abdulnabi, M. (2016). The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks. *Pervasive and Mobile Computing*, *25*, 1-25.