Handling Aperiodic Overloads in Real Time Operating System

*Note: Sub-titles are not captured in Xplore and should not be used

1st Charles Arsenal Okere
Department of Electronic Engineering
Hamm-Lippstadt University Of Applied Science
Lippstadt, Germany
charles-arsenal.okere@stud.hshl.de

***Abstract*—Many industrial applications with real-time demands are composed of mixed sets of tasks with a variety of requirements. The use of dynamic real-time operating systems (RTOS) becomes more and more common in modern manufacturing, aircraft, industrial automation systems, and telecommunication systems. Tasks must be completed with correct results by specified deadlines in such systems. The real-time task that occurs at any random time is known as an aperiodic real-time task. Between two aperiodic real-time tasks the time interval maybe even be zero. Soft real-time tasks are generally aperiodic real-time tasks. It is also possible that these tasks may occur frequently or there might be a large time interval between two aperiodic real-time tasks. However, in practice, the workloads of real-time systems fluctuate often due to task arrivals, peripheral device failures and so on. The newly accepted tasks may generate system overload, causing some of the tasks already in the system to miss their deadlines. In this work, we look at the topic of aperiodic overloading and try to figure out how to deal with it in a real-time operating system.**

I. Introduction

Real-time scheduling is the kernel technique that significantly impacts RTOS performance [1]. Real-time applications require complex constraints such as precedence, end-to-end deadlines, jitter, and dispersion [2]. If one fails, the consequences can be catastrophic, including injury to humans or property [3]. On the other hand, soft real-time systems can endure frequent failures with relatively small consequences [4].

Online scheduling allows for the efficient reclamation of time saved as a result of early completion and the management of overload situations [5]. Feasibility tests determine if a particular set of tasks can be efficiently scheduled using the algorithm's criteria. Schedules for distributed systems with intricate restrictions like precedence, jitter, and end-to-end deadlines can be created using table-driven scheduling [6]. This method is suitable for critical, demanding, real-time systems [7] [8].

Overload is described as a situation in which there is inadequate CPU time to finish all jobs, causing some deadlines to be missed. Traditional online scheduling algorithms like EDF or FPS perform poorly in overload situations [5]. Offline schedulers suffer from overload due to their lack of flexibility [9].

Scheduling periodic and aperiodic jobs has been a significant challenge in designing real-time computer systems. Periodic jobs repeat themselves after a predetermined time, referred to as the period [10]. Only when specific events occur are event-driven actions taken. In an intelligent parking situation, a job that looks for cars can't start until a car comes in [11].

Multi-processor systems are primarily utilized for computationally intensive applications. The use of real-time systems to conduct periodic and aperiodic activities has become increasingly widespread [12]. Aperiodic tasks are dynamic tasks that recur at any arbitrary moment and have a soft deadline. The time slice of the periodic tasks is changed to allow for the execution of these tasks [13].

A sporadic task is a discontinuous sequence of disconnected stages arranged sequentially. In contrast to routine activities, which repeat at regular intervals, sporadic occupations have a maximum rate of occurrence [14]. A collection of required and sufficient model parameter relationships for real-time job execution has been established [15]. This study shows which solution to the problem of aperiodic overloads has worked best in the past.

The remainder of the paper will be written in the following manner. Section 2 summarizes prior research on periodic, aperiodic, sporadic, offline, and online-based real-time scheduling. Section 3 is devoted to the technique, which details how several algorithms approach aperiodic problems. Section 4 provides the discussion in which we examine the optimal solution for dealing with aperiodic overloads and alternatives. Finally, Section 5 concludes the document by summarizing its significant ideas.

II. Literature Review

Assume that the periodic tasks were scheduled manually using a conventional offline scheduler. At run time, scheduling is accomplished by using the slot shifting approach, which permits aperiodic activities to be completed if they do not

Identify applicable funding agency here. If none, delete this.

cause an offline task to miss its deadline. Jan and El formulated overload handling as a general binary optimization issue and provided a method for addressing it in their work [14]. They formulate overload handling as a broad binary optimization problem and propose a solution. Numerous industrial applications that require immediate attention are composed of various occupations with various requirements. These might be straightforward time constraints such as period and deadline, or they can be more complex, for example, to indicate application-specific or non-temporal constraints, reliability, or performance. Damir and El developed a technique for dealing with a heterogeneous collection of tasks and constraints, including periodic tasks with sophisticated and straightforward constraints, soft and hard aperiodic tasks, and sporadic activities [16]. Rather than developing a custom solution for a specific set of constraints, they propose an EDF-based runtime method and the use of an offline scheduler to reduce the complexity of sophisticated constraints.

Damir El. proposed an approach for dealing with a diverse set of tasks and constraints in another paper [16], including periodic tasks with sophisticated and straightforward restrictions, soft and hard aperiodic, and sporadic activities. They recommend employing an offline scheduler to manage routine activities' complex scheduling and resource constraints, which they then transform into a basic EDF model with start and end periods and deadlines. Real-world industrial applications, such as distribution, end-to-end deadlines, and jitter management impose complex constraints on real-time systems. Most scheduling algorithms concentrate primarily on a small set of constraints and criteria. Gerhard and El. [17] proposed a method for effectively handling soft real-time jobs in offline scheduled real-time systems using a total bandwidth server. The offline scheduler starts by resolving problematic limitations, reducing their complexity, and ensuring bandwidth availability. The set schedule is broken down into individual jobs on single nodes, each with only a start time and a deadline. The earliest deadline is used first, and total bandwidth server scheduling is used to execute these at run-time.

Modern industries are moving away from traditional industrial networks and toward virtual machine (VM)-based networks as hardware virtualization and cloud computing improve. In a multicast TSN, Qinghan, and El. [18] study an online scheduling approach for dealing with dynamic VM migrations. The network notion is introduced first, followed by the formalization of the VM migration problem. While earlier research has mainly employed offline approaches, a new online scheduling framework called Minimal Distance Tree Construction-Heuristic Breadth-First Search (MDTCHB) has been developed to respond quickly to VM migrations. The framework has two steps: (1) offline scheduling and (2) online rescheduling.

The industrial revolution is progressing, and autonomous technology and embedded Internet of Things (IoT) technologies are at the forefront. Real-time systems and real-time computing are critical in autonomous technology. Embedded IoT devices must respond in real-time while still meeting all requirements. Sehrish et al. [19] suggested a custom-designed adaptive and intelligent scheduling method to effectively execute and manage complex and soft real-time tasks in embedded IoT systems. The proposed scheduling method distributes CPU resources equitably among soft real-time jobs that may be starved in overloaded instances while focusing on the execution of high-priority, demanding real-time tasks as its primary goal. Intricate real-time systems with hardware-in-the-loop simulations are rigid real-time systems that require quicker simulation than real-time. A method for executing challenging real-time systems with simulations that are slower than real-time is proposed. This provides a predicted execution of parallel simulations in intricate real-time systems [20]. Michael et al. The simulation path results are saved until the real-time subsystem requires them, at which point one is chosen and supplied to it to continue the system's execution.

Scheduling periodic and aperiodic jobs to satisfy their time restrictions has been a significant concern in designing real-time computer systems. In most cases, work scheduling algorithms in such systems must meet periodic task deadlines while also providing fast response times for aperiodic jobs. A new scheduling algorithm was proposed by Tein–Hsiang at El. [10]. The new approach generates a periodic server with the highest priority but not always the shortest period. If no aperiodic tasks are waiting, the server is suspended from decreasing overhead, and it is activated as soon as the next aperiodic job arrives. As a top priority, one aspect of real-time task scheduling is the intelligent modeling of input jobs to produce logically accurate output within the deadline and use the least amount of CPU power. Shabir and El provide a resource-aware method for minimizing the hyper-period of input tasks using device profiles, allowing tasks with any period value to be admitted [11]. The suggested work is compared to similar existing techniques, and the findings show significant power usage reductions.

Multi-processor systems have multiple processors and are typically utilized for computationally heavy applications. Real-time systems need task execution to be completed within a predefined time frame. Ayaz et al. propose a scheduling technique that 1) executes aperiodic tasks at full speed and migrates periodic tasks to other processors if their deadline is earlier than that of aperiodic tasks–reducing response time–and 2) executes aperiodic tasks at a reduced speed by identifying the appropriate processor speed without affecting response time–reducing energy consumption [12]. We demonstrate the suggested algorithm's efficiency through simulations and show that it outperforms the well-known total bandwidth server approach. Furthermore, Aicha et al. proposed a unique scheduling technique for efficiently computing deadlines for activities and communications while adhering to relevant limitations [13]. This method consists of two phases: the first phase defines different time slots, each of which is defined by energy and frequency parameters to address the issue of energy availability; and the second phase calculates the deadlines necessary to ensure the feasibility of a real-time system by taking into account the invocation of aperiodic task execution

and task precedence constraints. This article aims to discuss the real-time scheduling challenge for multi-core computers powered by renewable energy obtained from the environment.

Paolo et al. approach a controller implementation that tolerates overflow events and responds appropriately, which can be easily integrated into current controller implementations, mainly commercial off-the-shelf control systems [21]. The primary strengths of this strategy are its ease of application and a high degree of deployment flexibility. It does not require a stochastic model of the system's temporal evolution, nor does it rely on forecasting future delays. For a real-time system (RTS) that processes both sporadic and periodic jobs (multiple-period), Xi Wang et al. provide a novel modular modeling framework for describing task parameters that adhere to discrete-event system (DES) concepts and techniques [22]. A job is represented by a synchronized automaton with parameter-specific modular models. As a result, the DES model of the RTS is synchronized with the DES model of these tasks. The supervisory control theory solves priority-free conditionally-preemptive (PFCP) real-time scheduling by identifying all possible safe execution sequences.

Formal performance verification of real-time constraint systems is both necessary and challenging. Real-time systems frequently comprise numerous components, such as processors or buses, on which multiple tasks are concurrently executed. Sophie et al. propose a novel compositional approach for securely obtaining quantitative information about real-time systems [23]. Their approach is based on a novel model describing intermittent overload at a system's input. They demonstrate how to get safe quantitative information about the response time of each activity from such a model. Experiments validate this strategy using a real-world case. This article offers an overview of the fundamentals of real-time aperiodic overload management. This article aims to explain how various approaches or algorithms can be used to manage aperiodic jobs in a real-time operating system.

III. Methodology

This section will go through different algorithms and their implementation strategies for dealing with aperiodic job scheduling offline and online in real-time. To accomplish so, we first discussed a method for adapting complicated aperiodic tasks based on time-triggered systems' offline generated schedules. It is based on slot shifting [17] [2], a technique for combining offline and online scheduling by repurposing underutilized resources. Then, we present an EDF-based 0 (N) acceptance test to determine if aperiodic tasks can be included in offline scheduled tasks. The approach avoids explicitly handling resource reservations for guaranteed tasks and their impact on offline scheduled activities at runtime. As a result, flexible task rejection and removal strategies and aperiodic overload management can be used [24] [7].

The method's central concept is based on the standard first-to-deadline guarantee, but it is modified to function on top of the offline schedule. EDF is predicated on a fully available CPU; we must evaluate the impact of offline scheduled tasks on their viability. In pseudo-code, we now give the acceptance test and technique for finishing time calculation. Read the following comments in order.

1) for $(i = 1; dl\,(G_i) < dl(A); i++)$;
2) $ft = getFT\,(ft\,(G_i), C(A))$;
3) if $(ft \leq dl(A))\{$
4) for$(j = i + 1; j < n; j++)\{$
5) $ft = \mathrm{getFT}\,(ft, c_i\,(G_j))$;
6) if $(ft > dl\,(G_j)) \Rightarrow$ not feasible!
7) $\}$
8) $\mathrm{insert}\,(A, G_i)$;
9) else reject A
10) $g\,\mathrm{etFT}\,(ft_p, remc)\,\{$

$$SC_r = \mathrm{start}\,(I_c) + S_c\,(I_c) - ft_p$$

11) while (remc $> SC_r)\,\{$ if $(SC_r\,(I_c) > 0)$ remc = remc $-SC_r$; while (remc $> SC_r)\,\{$

$$\begin{aligned} if\,(SC_r\,(I_c) > 0) \\ \text{remc } &= \text{remc} - SC_r \\ C++; \\ ft_p &= \mathrm{start}\,(I_c) \\ SC_r &= S_c\,(I_c) \end{aligned}$$

$C++;\ ft_p = \mathrm{start}\,(I_c)\ SC_r = S_c\,(I_c)\,;\ \}$

12)

$$\mathrm{return}\,(ft_p + remc)$$

Find the final task in the set $G_i, (G_i)$, with a deadline before dl, and locate it (A). Based on the completion time of G_i and A's execution demand, calculate A's completion time. If $\mathrm{A}f$ is completed ahead of schedule, then all tasks in G_i i with deadlines should be completed after $\mathrm{dl(A)}$. Next, get the current task's completion time based on the previous task's completion time. According to our findings, adding A to G_i will result in a non-feasible set, meaning that not all of the jobs in G_i will be completed by their deadlines. Therefore, we can either reject G_i task A or another task. Insert A into the previously guaranteed firm aperiodic tasks G_i if all jobs can be completed on time even if A is introduced. If A does not meet its deadline, then it should be rejected. The time it takes to complete a job is estimated by considering the time it took to complete a predecessor task and the execution demand of the target task. Take into account the amount of time left in the current interval, which is the interval that includes the completion time of the preceding task (the earliest possible start time for the current task) $f^t\,(T_i) = f^t\,(T_{i-1}) + c\,(T_i)$ in the absence of offline jobs.

Second, a novel technique was present for configuring feasible software task scheduling with precedence restrictions on a multi-core processor fueled by renewable energy obtained from the environment [13]. To deal with the energy availability issue, we first create several time slots, each of which is described as a time interval characterized by well-calculated energy and frequency fluctuation intervals. Then we calculate

the effective deadlines for each time slot, which will be at most equal to the periods. This action affects both tasks and messages. The maximum value between cumulative execution time requested by aperiodic and periodic tasks that must be performed before the considered task in the same core, and (ii) the maximum time required by this task to receive all messages coming from its successors if they exist, is used to calculate each task's deadline. Because energy consumption is linearly related to frequency, the computing activities are done using the least frequency of each time slot to guarantee the estimated deadlines' validity even if the instantaneous charging rate of energy is the smallest [25] [26].

Model of Aperiodic Tasks Each aperiodic task $\tau_e^1, e \in \{1, \ldots, k\}$ is defined by: I the WCEC c_e^1, (ii) the execution time T_e^1, which is dependent on the frequency of the core M_x on which τ_i^0 is running and the WCEC, and (iii) the needed energy E_e^1.

$$T_e^1 = \left\lceil \frac{C_e^1}{f^x} \right\rceil$$

And,

$$E_e^1 = Csf_3 \times (f^x)^2 \times C_e^1$$

An aperiodic task can appear at any time and in any manner. As a result, for each aperiodic task τ_e^1, record the maximum number of occurrences on the time interval $[1, t_1]$.

Lastly, an offer was suggested for solving an overload management problem as a generic binary optimization problem [14]. Our solution is based on slot shifting [17] [2] to allow for offline and online scheduling integration. Active jobs are planned using the basic EDF, supplemented by an overload detection and resolution mechanism. A penalty includes a value for assured tasks that missed their deadline, such as rejection due to overload.

The constraints are listed by increasing length in Overload Detection and Resolution, as defined above. Even assuming that $[Di, fti] \leq 0$ for $1 \leq I \leq n$, the problem is difficult to solve when all limitations except the last one are trivially satisfied. Our solution is built on heuristics that use the unique characteristics of this situation. One of these properties is that each constraint has fewer variables than before. The rejection algorithm iterates through the ordered constraints, solving each one separately. The initial $y-1$ $y-1$ limitations are trivially satisfied because the task set was free of overload before the new task τy arrived. All xi variables are initially set to 0, indicating that no jobs are to be eliminated. Each constraint is overcome by setting some of the variables to 1 or keeping them untouched. While solving a constraint with $xj = 1$, we never alter this variable again when solving the following restrictions. A single constraint is solved in two steps unless it is trivially satisfied by the existing variable settings. We find the one with the lowest Vi^1 from the variables $xi = 0$ on the left-hand side of the limitation, such that $xi = 1$ would solve the constraint. Following that, we collect variables from the left-hand side that will not solve the constraint on their own but may be able to solve it jointly. Starting with the one with the lowest value density $\left(Vi^1/Ci\right)$, the collection is based on a greedy strategy. Finally, the best single choice's V_i^1 is compared to the collection's summed V_i values to determine what the final choice should be.

IV. Discussion

In the technique section, several potential strategies for dealing with difficulties in real-time operating systems caused by aperiodic overloads were examined. The most commonly used algorithm on the market is slot shifting. Slot shifting is a strategy for using idle resources to combine offline and online scheduling. When used with a distributed schedule with broad task limitations, it successfully manages aperiodic activity. Slot shifting in an offline schedule gathers information about unused resources and leeway and uses that information to make extra work possible, that is, without breaching constraints on previously planned activities. All offline planned tasks are assured to be completed before their respective deadlines due to spare capacity and slot shifting intervals. They're designed to run as late as possible, but they can be completed sooner during run-time, i.e., the time frame in which they're executed can be adjusted within the limits of their feasibility window. Aperiodic activities make use of resources that aren't being utilised by other offline activities. Intervals and spare capacity are examples of indicators that give data on the quantity and location of available resources. As a result, we'd like to incorporate aperiodic jobs without jeopardizing the viability of the current offline activities. In addition, we recommend using an EDF-based 0 (N) acceptance test to see if an aperiodic job should be included. EDF (Earliest Deadline First Scheduling) is a dynamic priority scheduling strategy in real-time operating systems that prioritizes tasks by moving them to the front of the queue. When a scheduling event happens, the queue is searched for the process that has the shortest deadline in order to meet the event's requirements. Another feature of EDF is that it does not make any assumptions about how often tasks are completed. As a result, it is independent of working hours and can be used to plan jobs that occur on a regular basis.

V. Conclusion

Several unique ways for scheduling firm real-time jobs on an overloaded server were discussed in this paper. We've also gone through three important real-time overload handling techniques, which cover a wide spectrum of execution circumstances. The central focus of this research has been on an EDF-based slot shifting technique that is more suitable for dynamic priority scheduling in real-time systems. This moves tasks to the front of the queue, prioritizing them. This would assist the procedure in completing in a short amount of time. When a scheduling event happens, the queue is searched for the process with the shortest deadline to meet the requirements of the event. Another property of EDF is that it makes no assumptions regarding the frequency with which tasks are completed. As a result, it is unaffected by working hours and can be used to schedule jobs that occur frequently. EDF

scheduling system that ensures the execution of high-criticality jobs while reducing the number of tasks that are dropped.

REFERENCES

[1] Bouyssounouse, B., Sifakis, J. (2005). Real-Time Scheduling. In Embedded Systems Design (pp. 242-257). Springer, Berlin, Heidelberg.

[2] Lennvall, T. (2003). Handling Aperiodic Tasks and Overload in Distributed Off-line Scheduled Real-Time Systems. Mälardalen University.

[3] Minaeva, A., Hanzálek, Z. (2021). Survey on periodic scheduling for time-triggered hard real-time systems. ACM Computing Surveys (CSUR), 54(1), 1-32.

[4] Kocian, A., Chessa, S. (2018, June). Static Dataflow Analysis for Soft Real-Time System Design. In International Symposium on Distributed Computing and Artificial Intelligence (pp. 175-182). Springer, Cham.

[5] Carletto, P., Carletto, T. (2018). Ravenscar-EDF: Further Results from Improved Comparative Benchmarking. ACM SIGAda Ada Letters, 38(1), 40-40.

[6] Kumar, S., Devaraj, R., Sarkar, A. (2018). A hybrid offline-online approach to adaptive downlink resource allocation over LTE. IEEE/CAA Journal of Automatica Sinica, 6(3), 766-777.

[7] Phavorin, G., Richard, P., Goossens, J., Maiza, C., George, L., Chapeaux, T. (2018). Online and offline scheduling with cache-related preemption delays. Real-Time Systems, 54(3), 662-699.

[8] Koufakis, A. M., Rigas, E. S., Bassiliades, N., Ramchurn, S. D. (2019). Offline and online electric vehicle charging scheduling with V2V energy transfer. IEEE Transactions on Intelligent Transportation Systems, 21(5), 2128-2138.

[9] Yoo, S., Jo, Y., Bahn, H. (2021). Integrated scheduling of real-time and interactive tasks for configurable industrial systems. IEEE Transactions on Industrial Informatics, 18(1), 631-641.

[10] Malik, S., Ahmad, S., Ullah, I., Park, D. H., Kim, D. (2019). An adaptive emergency first intelligent scheduling algorithm for efficient task management and scheduling in hybrid of hard real-time and soft real-time embedded IoT systems. Sustainability, 11(8), 2192.

[11] Pietrykowski, M., Smidts, C. (2022). Predictive Execution of Parallel Simulations in Hard Real-Time Systems. IEEE Transactions on Computers.

[12] Yu, Q., Wan, H., Zhao, X., Gao, Y., Gu, M. (2019). Online scheduling for dynamic VM migration in multicast time-sensitive networks. IEEE Transactions on Industrial Informatics, 16(6), 3778-3788.

[13] Lin, T. H., Tarng, W. (1991). Scheduling periodic and aperiodic tasks in hard real-time computing systems. ACM SIGMETRICS performance evaluation review, 19(1), 31-38.

[14] Ahmad, S., Malik, S., Ullah, I., Fayaz, M., Park, D. H., Kim, K., Kim, D. (2018). An Adaptive approach based on resource-awareness towards power-efficient real-time periodic task modeling on embedded IoT devices. Processes, 6(7), 90.

[15] Khan, A. A., Ali, A., Zakarya, M., Khan, R., Khan, M., Rahman, I. U., Abd Rahman, M. A. (2019). A migration aware scheduling technique for real-time aperiodic tasks over multi-processor systems. IEEE Access, 7, 27859-27873.

[16] Goubaa, A., Khalgui, M., Li, Z., Frey, G., Zhou, M. (2020). Scheduling periodic and aperiodic tasks with time, energy harvesting and precedence constraints on multi-core systems. Information Sciences, 520, 86-104.

[17] Carlson, J., Lennvall, T., Fohler, G. (2001, June). Value based overload handling of aperiodic tasks in offline scheduled real-time systems. In Work-in-progress Session, 13th Euromicro Conference on Real-Time Systems.

[18] Jeffay, K., Becker, D., Bennett, D., Bharrat, S., Gramling, T., Housel, M. (1994). The design, implementation, and use of a sporadic tasking model. university of north corolina at chapel Hill Department of computer sciences, Chapel Hill, NC, 27599-3175.

[19] Isovic, D., Fohler, G. (2000, November). Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In Proceedings 21st IEEE Real-Time Systems Symposium (pp. 207-216). IEEE.

[20] Isovic, D., Fohler, G. (1999, June). Online handling of hard aperiodic tasks in time triggered systems. In Proceedings of the 11th Euromicro Conference on Real-Time Systems.

[21] Isovic, D., Fohler, G. (2009). Handling mixed sets of tasks in combined offline and online scheduled real-time systems. Real-time systems, 43(3), 296-325.

[22] Fohler, G., Lennvall, T., Buttazzo, G. (2001, October). Improved handling of soft aperiodic tasks in offline scheduled real-time systems using total bandwidth server. In ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 01TH8597) (pp. 151-157). IEEE.

[23] Pazzaglia, P., Hamann, A., Ziegenbein, D., Maggio, M. (2021, February). Adaptive design of real-time control systems subject to sporadic overruns. In 2021 Design, Automation Test in Europe Conference Exhibition (DATE) (pp. 1887-1892). IEEE.

[24] Wang, X., Li, Z., Wonham, W. M. (2018). Priority-free conditionally-preemptive scheduling of modular sporadic real-time systems. Automatica, 89, 392-397.

[25] Quinton, S., Hanke, M., Ernst, R. (2012, March). Formal analysis of sporadic overload in real-time systems. In 2012 Design, Automation Test in Europe Conference Exhibition (DATE) (pp. 515-520). IEEE.

[26] Fohler, G. (1995, December). Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In Proceedings 16th IEEE Real-Time Systems Symposium (pp. 152-161). IEEE.

[27] Isovic, D. (2001). Handling sporadic tasks in real-time systems: Combined offline and online approach. Mälardalen University.

[28] Awadalla, M. (2015). Processor speed control for power reduction of real-time systems heuristically. International Journal of Electrical and Computer Engineering, 5(4), 701.

[29] Jarray, F., Chniter, H., Khalgui, M. (2015, June). New adaptive middleware for real-time embedded operating systems. In 2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS) (pp. 610-618). IEEE.