IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 14, 2019, accepted February 1, 2019, date of publication February 20, 2019, date of current version March 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2900288

Investigating the Schedulability of Periodic Real-Time Tasks in Virtualized Cloud Environment

HITHAM ALHUSSIAN[1], (Member, IEEE), NORDIN ZAKARIA[1], AHMED PATEL[2], AYMAN JARADAT[3], SAID JADID ABDULKADIR[4], (Member, IEEE), ABDELAZIZ Y. AHMED[5], (Member, IEEE), HUSSEIN T. BAHBOUH[6], SALLAM OSMAN FAGEERI[7], ASIM ABDALLAH ELSHEIKH[8], AND JUNZO WATADA[4], (Member, IEEE)

[1]High Performance Cloud Computing Center (HPC[3]), Institute of Autonomous Systems, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Malaysia
[2]Computer Networks and Security Laboratory, State University of Ceara, Fortaleza 60020-181, Brazil
[3]Department of Computer Science, College of Science and Human Studies, Hutat Sudair, Majmaah University, Majmaah 11932, Saudi Arabia
[4]Centre for Research in Data Science, Institute of Autonomous Systems, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Malaysia
[5]Department of Electrical Engineering, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Malaysia
[6]College of Mechanical and Electrical Engineering, Damascus University, Damascus 5442, Syria
[7]College of Computer Science and Information Technology, Alzaeim Alazhari University, Khartoum 13311, Sudan
[8]College of Computer Science and Information Technology, Tabuk University, Tabuk 71491, Saudi Arabia

Corresponding author: Ayman Jaradat (ay.jaradat@mu.edu.sa)

This work was supported in part by the University Teknologi Petronas (UTP)-Malaysia, under Project 0153AA-F54, and in part by the Deanship of Scientific Research, Majmaah University, under Project 1440-46.

ABSTRACT In this paper, we developed a computing architecture and algorithms for supporting soft real-time task scheduling in a cloud computing environment through the dynamic provisioning of virtual machines. The architecture integrated three modified soft real-time task scheduling algorithms, namely, Earliest Deadline First, Earliest Deadline until Zero-Laxity, and Unfair Semi-Greedy. A deadline look-ahead module was incorporated into each of the algorithms to fire deadline exceptions and avoid the missing deadlines, and to maintain the system criticality. The results of the implementation of the proposed algorithms are presented in this paper in terms of the average deadline exceptions, the extra resources consumed by each algorithm in handling deadline exceptions, and the average response time. The results not only suggest the feasibility of the soft real-time scheduling of periodic real-time tasks in cloud computing but that the process can also be scaled up to handle the near-hard real-time task scheduling.

INDEX TERMS Real-time, cloud computing, virtual machine, deadline, laxity.

I. INTRODUCTION

With the exponential growth of big data induced by the proliferation of Internet technologies, there has been an increase in the demand for cloud computing over the last few years. Cloud computing is a type of distributed parallel computing system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements between the service provider and consumers [1], [2]. Cloud computing facilitates flexible and dynamic outsourcing of applications while improving cost-effectiveness.

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

Cloud Computing offers three different types of service models: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). In SaaS model, the consumer is offered to use provider's applications hosted in a cloud infrastructure. In PaaS, the consumer is provided with required software and hardware tools to develop and deploy cloud applications that are hosted in the provider's cloud infrastructure. In IaaS, the consumer is provisioned the use of virtualized computing, storage and network resources that are delivered on demand basis. In IaaS model, the consumer will not have control on the cloud infrastructure, however, he can control the operation system, the storage and deployed applications beside the possibility of controlling limited network components such as firewalls [1], [2].

Cloud computing is enabling the development of the next generation of computing services, which would be heavily geared toward massively distributed on-line computing. It also affords a new model of globally accessible on-demand high-performance computing (HPC) services. However, such benefits are presently not available to real-time safety-critical applications [3]. This could be due to the inability of current cloud infrastructures to support timing constraints. Compared with the case of existing real-time scheduling platforms such as multiprocessors and multicores [4], the handling of real-time constraints on cloud platforms is more complex due to the difficulty of predicting system performance in virtualized and heterogeneous environments [5].

However, real-time applications are gradually progressing unto cloud computing platforms, driven by the tremendous possibilities afforded by such platforms. Examples of hard and soft real-time system applications of cloud computing are military distributed control systems for remote surveillance, early warning and response systems, sensor-driven unmanned vehicles with augmented intelligence, and cloud gaming [5]–[7].

Cloud computing technology is not actually geared toward hard real-time applications in closed environments, but soft real-time applications that do not require direct exposure to the hardware bypassing system software [6], [7]. Incidentally, soft real-time scheduling policies can be integrated in virtualization platforms, enabling the system to deliver hierarchical real-time performance [6].

In this paper, we propose and analyze the possibility of applying real-time task scheduling, or deadline-constrained task scheduling, on IaaS cloud computing service model, for the support of soft and near-hard real-time applications. Examples of such applications are cloud-based gaming, online video streaming, and telecommunication management [5]–[7]. Such applications could benefit significantly from cloud computing despite the limitations associated with the start-up time and communication of virtual machines. This is because of the ability of cloud computing to support dynamic workloads, thereby enabling the elastic allocation of resources [6], [7].

The rest of the paper is organized as follows. Section 2 introduces the task model and the proposed cloud architecture, and also defines some terms used in this paper. Section 3 briefly reviews related works. Section 4 further describes the proposed cloud architecture and its underlying scheduling algorithms. Section 5 presents and discusses the results of the demonstrative implementation of the architecture. Finally, the conclusions drawn from the study are presented in Section 6.

II. MODEL AND TERMS DEFINITION

This paper considers the problem of scheduling independent periodic real-time tasks with implicit deadlines (i.e. task deadlines are equal to task periods, i.e. $d_i = p_i$) on a virtualized cloud environment that offers IaaS services to cloud subscribers.

In real-time systems, a periodic real-time task is one that is 'released' periodically at constant intervals. Such periodic real-time tasks can be found in a broad range of real-time system applications. In fact, many of the tasks carried out by these systems are typically periodic in nature. For-example, monitoring certain conditions in a plant or in a flight control system with different set of sensors sending their data periodically at constant rates, i.e. at every specific period. Another example is changing the track in a radar system over specified period according to the target movement. Yet another example is polling information from sensors periodically and respond accordingly (e.g. driving some actuators).

A periodic real-time task T_i, $i = 1, 2, \ldots, n$, is usually described by three parameters, namely, its worst-case execution time e_i, its deadline d_i, and its period p_i.[1] An instance *i.e.* release of a periodic task is referred to as a job and is denoted by $T_{ij} = (e_{ij}, p_{ij}), j = 1, 2, 3, \ldots,$. e_{ij} refers to the worst-case execution requirement of job T_{ij}, and p_{ij} refers to its period. The deadline of a job is the arrival time of its successor. For example, the deadline of job T_{ij} is the arrival time of job $T_{i(j+1)}$, namely, at time $(j+1)p_i$. The laxity of a job T_{ij} at time t, denoted by $l_{ij,t}$, is the duration over which T_{ij} can remain idle before its execution is commenced, as denoted by equation 1.

$$l_{ij,t} = p_{ij} - e_{ij,t} - t \tag{1}$$

where $e_{ij,t}$ denotes the remaining time in the execution of job T_{ij} at time t.

Another important parameter that is used to describe a task T_i is its utilization, denoted by equation 2, and refers to the proportion of the time required for its execution relative to the entire duration between its release and deadline.

$$u_i = e_i/p_i \tag{2}$$

We use the term, U_{sum} to denote the total utilization of a given taskset T, and U_{max} to denote the maximum utilization. A periodic real-time taskset $T = \{T_1, T_2, \ldots, T_n\}$ is said to be schedulable on N nodes each with m identical multiprocessors if and only if $U_{sum}(T) <= N \times m$ and $U_{max}(T) \leq 1$.

Table 1 summarizes the notations and terms used in this paper.

In IaaS cloud service model, Virtual Machines (VMs) are used to deliver computing, memory, and other resources to cloud subscribers. We assume that the Master Node (MN) in the cloud platform has access to a pool of virtual machine nodes. Initially, the pool contains N_{VM} virtual machine nodes, each of which contains m symmetric shared-memory multiprocessors (SMPs). The total number of available processors in the cloud is denoted by $M = N_{VM} \times m$.

Considering the target of a virtualized environment, α is used to denote the VM instantiation time, and β the delay introduced by the communication between the MN and local

[1] In the case of real-time tasks with implicit deadlines, i.e. $d_i = p_i$, the term d_i is usually omitted, with the term p_i used to refer to both the task deadline and period.

TABLE 1. NotationS and terms.

Notation	Definition
N, N_{VM}	Number of nodes
M	The total number of processors
n	Number of tasks
T	Set of tasks $\{T_1, T_2, \ldots, T_n\}$
T_i	ith task
T_{ij}	jth job of task T_i
p_i	Period (minimum inter-arrival time) of task T_i
p_{ij}	Period (minimum inter-arrival time) of jth job of task T_i
e_i	Worst-case execution requirement of task T_i
e_{ij}	Worst-case execution requirement of jth job of task T_i
$e_{ij,t}$	Remaining worst-case execution requirement of jth job of task T_i at time t
l_i	Laxity of task T_i
l_{ij}	Laxity of jth job of task T_i
$l_{ij,t}$	Remaining laxity of jth job of task T_i at time t
u_i	Utilisation of task T_i
U_{sum}	Total utilisation of T
U_{max}	Maximum utilisation in T
α	VM instantiation time
β	Communication Delay

VM nodes.[2] Hence, if an instance of a task T_i, i.e. T_{ij}, is migrated and affected by the VM instantiation time, its new worst-case execution time is given by equation 3

$$e'_{ij} = e_{ij} + \alpha + \beta \tag{3}$$

Accordingly, the new laxity of job T_{ij} at time t is given by equation 4.

$$l'_{ij}(t) = p_{ij} - e'_{ij} - t \tag{4}$$

III. LITERATURE REVIEW

A real-time system is defined by Burns and Wellings [1] as an 'information processing system which has to respond to externally generated input stimuli within a finite and specified period: the correctness depends not only on the logical result but also on the time it was delivered; the failure to respond is as bad as the wrong response.' This means that a real-time system has a dual notion of correctness, namely, logical and temporal.

Another important feature of a real-time system is that it must be predictable. Predictability implies the possibility to show, demonstrate, or prove that the real-time constraints are always met based on any assumptions such as the workloads [2]–[7].

Real-time systems can be classified into two main classes, hard and soft, based on the cost of the failure associated with missing or violating the timing constraints [2], [7]–[9], [12]. There have been many proposals of optimal real-time algorithms for multiprocessors [8]–[13]. Such algorithms are used in hard real-time systems to guarantee that no deadline is missed in a schedulable taskset. However, they are inflexible and their schedulability performance may be degraded by overutilization of the system [14], [15]. Conversely, non-optimal algorithms, which are known to miss some task deadlines and thus applied to soft real-time systems, afford greater flexibility, and some of them may even perform better than optimal schedulers under system overutilization.

Although many scheduling algorithms have been proposed for virtualized cloud environments, most of them are targeted at resource management [16]–[19], performance, cost-effectiveness [20], [21], and energy conservation [22], [23]. A few studies have been conducted on deadline-based scheduling in cloud computing [24], [25]; however, they prioritized other scheduling metrics such as energy consumption, resource management, and cost-effectiveness, with task deadlines considered as a secondary scheduling factor. This could be due to the inability to predict the missing of a deadline to assess the system performance, owing to the delay introduced by the virtualization layer in a virtualized cloud-based system.

Utilizing current cloud computing resources, the constrained deadline of a real-time task was considered as the only scheduling metric in the present study. Modifications of three real-time scheduling algorithms, namely, EDF [26], EDZL [27], [28] and USG [29], were developed and integrated with a deadline look-a-head module to improve the performance with respect to the response time and the ability to intercept and handle deadline misses.

IV. METHODOLOGY

The following subsections discuss in detail the proposed cloud computing architecture and its underlying algorithms for real-time job scheduling on a cloud platform.

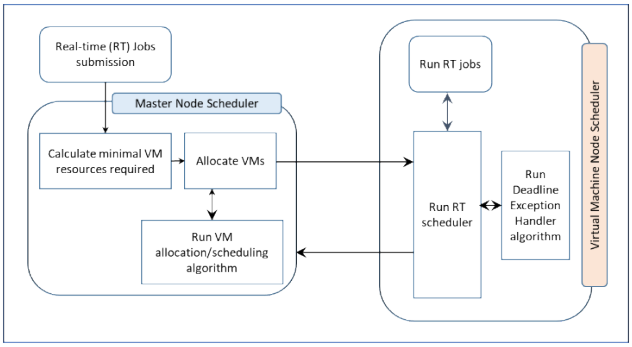

FIGURE 1. Proposed real-time cloud computing architecture.

A. PROPOSED ARCHITECTURE

The proposed cloud computing architecture, shown in Fig. 1, is composed of two main parts: the MN and the VM nodes. As mentioned previously, it was assumed that the MN had access to the VM pool. Hence, when submitted real-time jobs arrive at the MN, the master scheduler selects a VM resource from the pool and assigns tasks to it until equation 5 is violated, i.e. until the VM is fully utilized.

$$\sum \frac{e_i}{p_i} \leq m_{VM} \tag{5}$$

[2]In this paper, the delay caused by the communication between the VM nodes and the master node is considered zero, given the relatively low communication level, which is also supported by current high-speed networks.

```
1.  Receive real-time tasks
2.  Calculate minimum computing resources (VMs)
3.  Allocate VMs from VM pool
4.  For every VM node in VMs
5.      Assign tasks to the VM node.
6.      Run the VM node.
7.  Wait for deadline exceptions raised by VMs.
8.  If a deadline exception is received
9.      look for a node with an idle processor
10.     If such a node (VM) exists,
11          assign the task to that node
12.     else
13.         start a new VM node
14.         assign the task to that VM node
```

FIGURE 2. Pseudocode of the MN scheduler.

Once the minimum required number of VM resources is calculated, the MN scheduler starts by assigning real-time tasks to the VM node and start it immediately. Figure 2 shows the pseudo-code of the master node MN scheduler.

Conversely, once a VM node scheduler is started, it runs its assigned real-time tasks using one of the real-time scheduling algorithms EDF [26], EDZL [27], [28], and USG [29]. In addition, each VM node scheduler is equipped with a deadline look-a-head module that fires a deadline exception whenever a deadline miss is expected. The real-time task that is expected to cause the deadline miss event is then removed from the scheduling queue and inserted into an urgent queue shared by the VMs and the MN. The MN scheduler then seeks for a VM node with an empty processor and assigns the urgent tasks to any such node that is found. Otherwise, the MN scheduler would create a new VM node and assign the urgent task to it. Figure 3 shows the pseudocode of the VM node scheduler.

```
1.  Initialize scheduling queues

2.  Run the tasks according to proposed algorithm (USG,
    EDZL, EDF)
3.  Check for any deadline exceptions.
4.  If a deadline exception is suspected.
5.      remove the suspected task from the waiting queue
6.      raise a deadline exception event
7.      and send the suspected task to the master
```

FIGURE 3. Pseudo code of VM node scheduler.

B. IMPLEMENTATION OF VM NODE SCHEDULERS

The VM node schedulers are implemented using USG [29], EDZL [27], [28], and EDF [26]. Each of these algorithms was modified to incorporate a deadline look-a-head module to intercept any possible deadline miss. The following lemma shows exactly when a running task *i.e.* job is expected to miss its deadline.

Lemma 1: A job T_{ij} scheduled for execution on a virtual node VM_{N_i} is expected to fire a deadline exception at time $t = t_x$ when its laxity reaches $\alpha + \beta$ (i.e. $l_{ij}(t) = \alpha + \beta$) and the nearest running job to completion still has remaining execution units greater than $\alpha + \beta$ (i.e $e_{ij}(t) > \alpha + \beta$).

Proof: Suppose that job T_{ij} is being selected for execution at time $t = t_x$. Suppose also that, while task T_i is running it misses its deadline at time $t = t_z$, where $t_z > t_x$. This means that at time $t = t_z$, T_{ij} reached its deadline while still having some remaining units to execute. However, we know that when a task is being executed, its remaining units decrease while its laxity remains constant. Nevertheless, job T_{ij} misses its deadline during its execution. This could only happen if, when job T_{ij} is being selected for execution at time $t = t_x$, its remaining time to deadline (i.e. laxity) is less than its remaining execution units, i.e. $p_i(t = x) < e'_i(t = x) \Rightarrow p_i(t = x) - e'_i(t = x) < 0$. This means that $l'_i(t = x) < 0$. Hence, the last means of avoiding a deadline miss for job T_{ij} is when $l'_i(t) = 0$.

However, since we are assuming a virtualized cloud environment, job T_{ij} should fire a deadline exception before its laxity reached 0 to avoid missing its deadline. This is because we are considering the delay of VM instantiation time which is assumed to be α.

Hence job T_{ij} should fire the deadline exception at time $t = t_{cur} - (\alpha + \beta)$ considering into account that the nearest running job to completion still has remaining execution units greater than $\alpha + \beta$ (*i.e.* $e_{ij}(t) > \alpha + \beta$).

C. DEADLINE LOOK-A-HEAD SCHEDULERS

With the above lemma in mind, we modified the above-mentioned algorithms, namely, USG, EDZL, and EDF, by adding a deadline look-a-head module to each. This was achieved by maintaining a queue of waiting jobs in order of increasing laxity. The deadline look-a-head scheduling module constituted an additional queue in EDF because the jobs in its original waiting jobs queue are in order of increasing deadline, and not laxity.[3] Conversely, both USG and EDZL already maintain a queue of waiting jobs in order of increasing laxity. All the algorithms were thus appropriately updated with a deadline look-a-head handler to handle any deadline exception raised by the algorithm.

```
1.  while ((deadline_Look_A_HeadQ.size()>0) &&
    (deadline_Look_A_HeadQ.peek().key == tCur))
2.      Task tU = deadline_Look_A_HeadQ.poll();
3.      waitingQ.remove(tU);//remove the correspondent
4.      tU.key = tU.p - tU.key%tU.p+ tCur;//change to key
5.      MasterScheduler.urgentQ.add(tU);
```

FIGURE 4. Deadline exception handler.

Figure 4 shows the proposed deadline exception handler, which is called upon whenever a deadline exception is fired.

[3]More information on EDF, EDZL and USG is available on references [26]–[29] consecutively.

V. RESULTS AND DISCUSSION

To test the performance of the modified algorithms in the proposed cloud computing architecture, we used a standard procedure to generate the real-time tasksets [8], [9], [30]–[35]. The task periods p_i were generated using the Uniform Integer Distribution, which produces random integers within the range $[a, b]$, with all the possible values having the same likelihood of being produced. To generate the task worst-case execution times e_i, Uniform Real Distribution was first used to uniformly generate a random real number x within the range (0, 1]. The worst-case execution time of the tasks was then calculated using equation 6:

$$e_i = \lfloor x \times p_i \rfloor \tag{6}$$

The task periods and the worst-case execution requirements were subsequently uniformly chosen within the period [1, 1000].

With regard to the *VM* instantiation time, it was assumed that the worst-case delay caused by the *VM* startup time was 100 s. This was based on the performance evaluation of the VM startup time for cloud computing in [39]. In this previous work, it was shown that the average *VM* startup times for Linux and Rackspace machines were 96.9 and 44.2 s, respectively. The delay was divided by the number of real-time tasks to be scheduled, n, and the result was added to the worst-case execution time of each task. To ease the simulation, the delay caused by the communication between the *VM* nodes and the master node was neglected, given the relatively low communication level, which is also supported by current high-speed networks.

It was further assumed that the number of real-time tasks to be executed was twice the total number of processors on the targeted cloud computing platform. Table 2 enumerates the details of the simulation test bed with regard to the generated tasksets and their corresponding cloud computing architecture. For example, 10000 samples of real-time tasks of size 16, i.e. 16 tasks per sample, were generated for execution on a cloud platform containing four nodes, each of which was equipped with two processors, and another two nodes, each equipped with four processors. This configuration enabled the tracing and investigation of the relevant factors, namely, the numbers of nodes and processors that more significantly impacted the response time and deadline exceptions reported by each algorithm.

The results were presented in terms of the average number of deadline exceptions, the average number of additional resources required to handle the fired deadline exceptions, and the average response time. The following subsections discuss the results.

The simulation has been developed using Java SE 8 development kit installed on an Oracle Solaris 10 operating system running on a Core I7 machine equipped with 8 GB of RAM.

TABLE 2. Details of the simulation test bed.

Tasksets	Targeted Cloud Architecture	
	No. of Nodes	No. of Processors Per Nodes
10000 tasksets, , each taskset contains 16 tasks	4	2
	2	4
10000 tasksets, each taskset contains 32 tasks	8	2
	4	4
	2	8
10000 tasksets of size 64	16	2
	8	4
	4	8
	2	16
10000 tasksets of size 128	32	2
	16	4
	8	8
	4	16
	2	32
10000 tasksets of size 256	64	2
	32	4
	16	8
	8	16
	4	32
	2	64

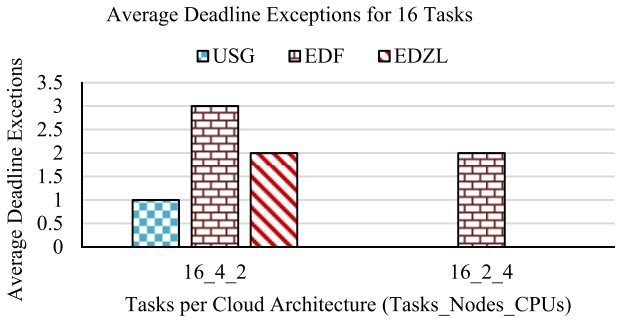

FIGURE 5. Average deadline exceptions of 16 task.

A. DEADLINE EXCEPTIONS

Figures 5-9 show the average deadline exceptions fired by each algorithm. It can be clearly seen that the USG fired the least number of deadline exceptions, followed by EDZL, and then EDF. This agrees with the findings of previous studies [14], [29], which showed that the performance of Least Laxity First (LLF)-based algorithms surpassed that of EDF-based algorithms. For example, as indicated in Fig. 5, when 16 tasks are scheduled on four nodes with two processors each, USG fires one deadline exception, EDZL fires two deadline exceptions, and EDF fires three deadline exceptions. However, when 16 tasks are scheduled on two nodes with four processors each, both USG and EDZL fire no deadline exception, while EDF fires two deadline exceptions. The superior performance of USG and EDZL could be attributed to the integrated Zero Laxity (ZL) policy [14], [28], [29].

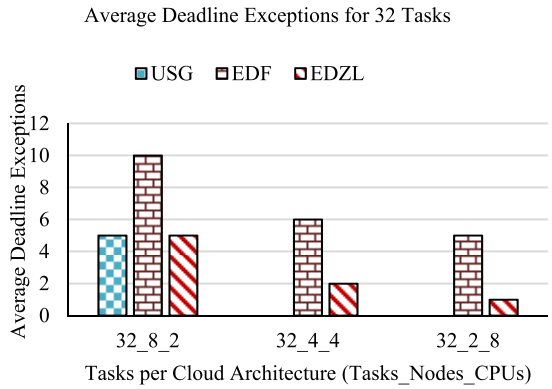

FIGURE 6. Average deadline exceptions of 32 task.

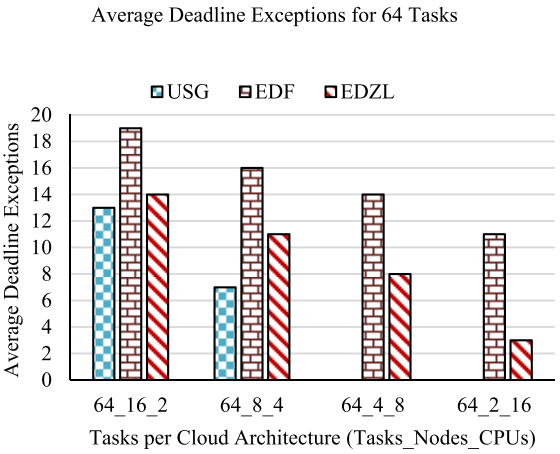

FIGURE 7. Average deadline exceptions of 64 task.

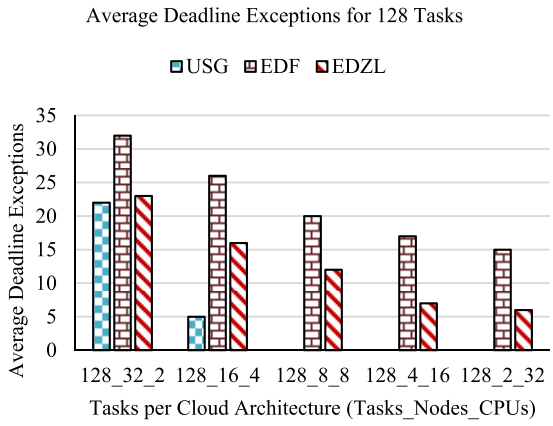

FIGURE 8. Average deadline exceptions of 128 tasks.

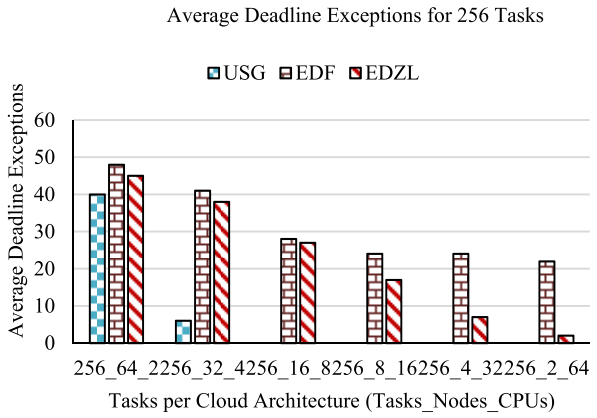

FIGURE 9. Average deadline exceptions of 256 tasks.

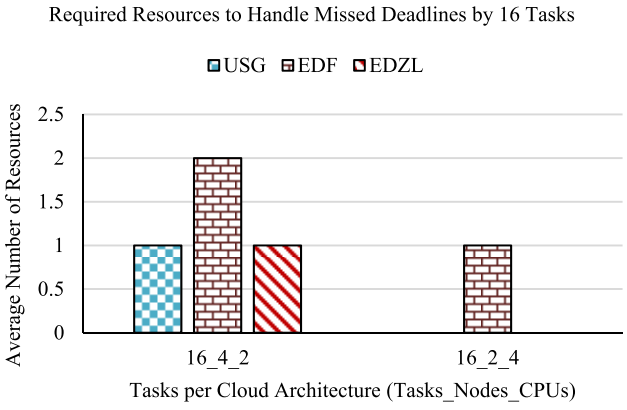

FIGURE 10. Extra resources of 16 task.

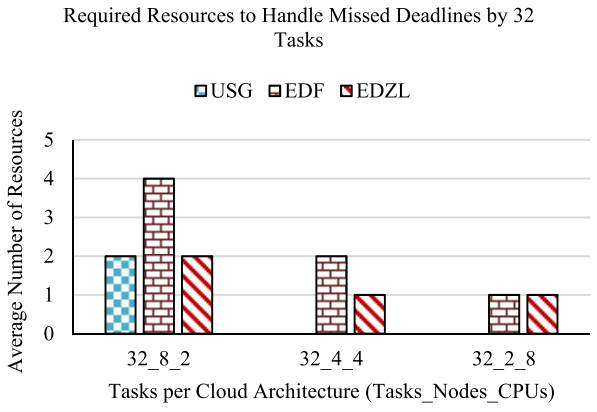

FIGURE 11. Extra resources of 32 task.

B. EXTRA REQUIRED RESOURCES

Based on the fired deadline exceptions, the MN scheduler allocates new VMs and assigns the urgent tasks to them in the absence of a VM node(s) with idle processors. Hence, the number of additional required resources is directly proportional to the number of fired exceptions. Figures 10-14 show the additional resources that the different algorithms require to handle the deadline exceptions. It can be clearly seen from the figures that USG requires the least amount of additional resources, which is because it fires the least number of deadline exceptions. It is followed by EDZL, and then EDF. For example, in Fig. 10, for 16 tasks scheduled on four nodes with two processors each, USG is provisioned with one extra resource, and each of EDZL and EDF with two extra resources. Conversely, for the same number of tasks scheduled on two nodes with four processors each, both

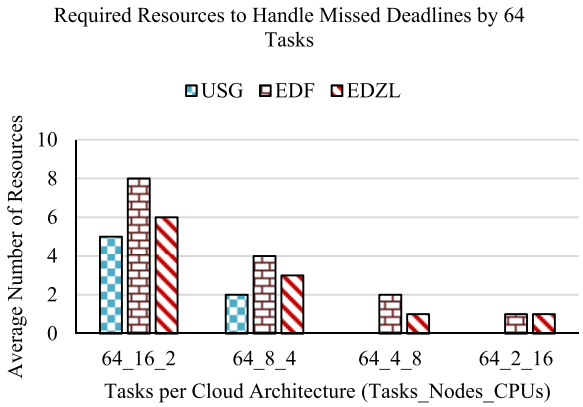

FIGURE 12. Extra resources of 64 task.

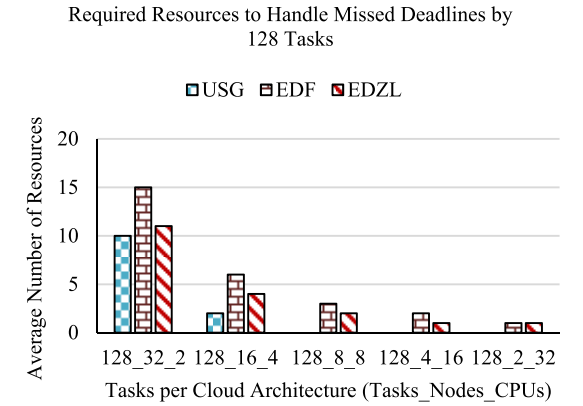

FIGURE 13. Extra resources of 128 tasks.

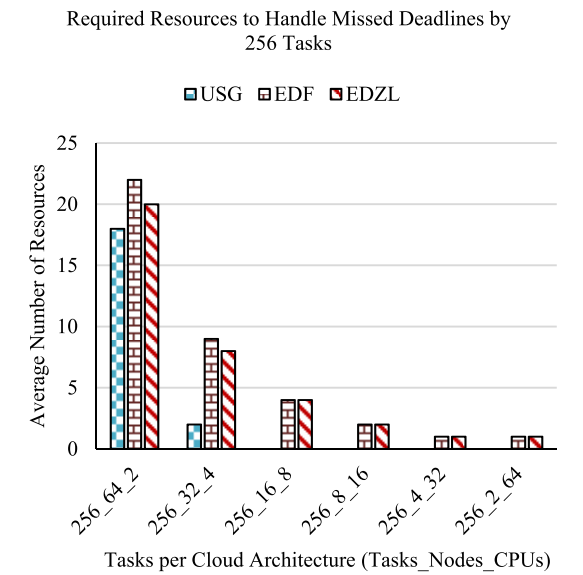

FIGURE 14. Extra resources of 256 tasks.

USG and EDZL require no additional resource, while EDF is provisioned with an extra resource.

C. RESPONSE TIME ANALYSIS

With regard to the response time, EDF produced the shortest among the three algorithms in the proposed

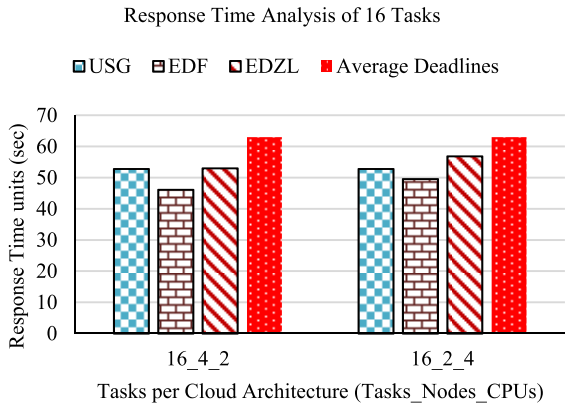

FIGURE 15. Average Response Time Analysis of 16 Tasks.

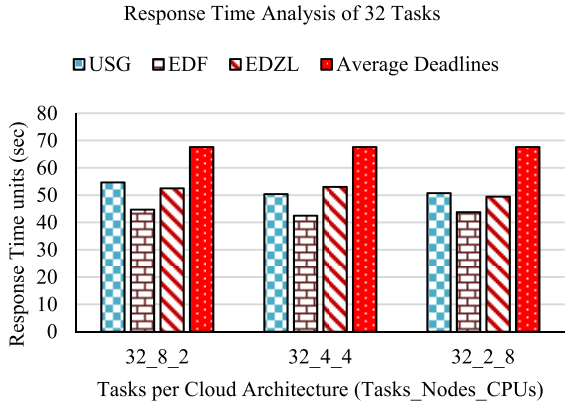

FIGURE 16. Average Response Time Analysis of 16 Tasks.

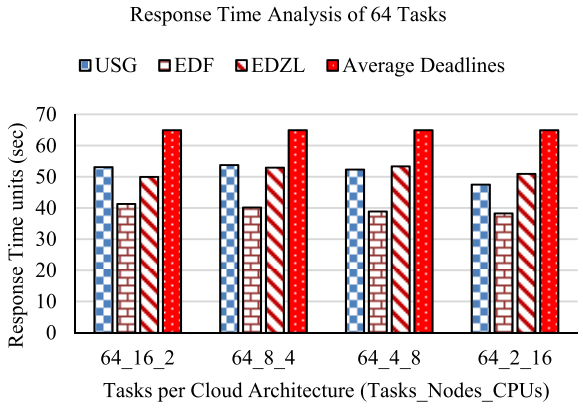

FIGURE 17. Average Response Time Analysis of 16 Tasks.

cloud architecture. This was, however, mainly because of the extra resources that the algorithm was provisioned with to handle the deadline exceptions. For example, as indicated in Figs. 15, when 16 tasks are scheduled on four nodes with two processors each, EDF produces an average response time of 46.1 s compared with the 52.8 s of USG and 53 s of EDZL. This is mainly because, for example, for the given architecture and number of tasks in Fig. 10, EDF is provisioned with two extra resources, which is twice that of each of

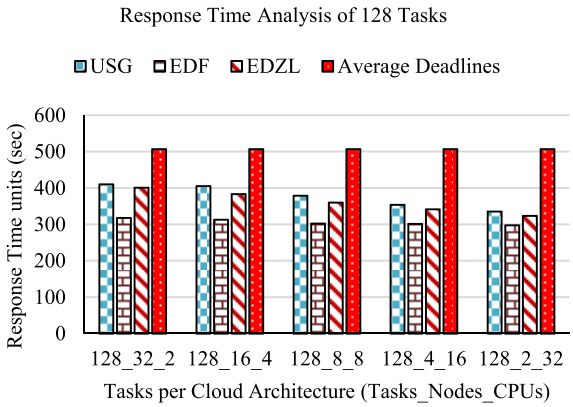

FIGURE 18. Average Response Time Analysis of 16 Tasks.

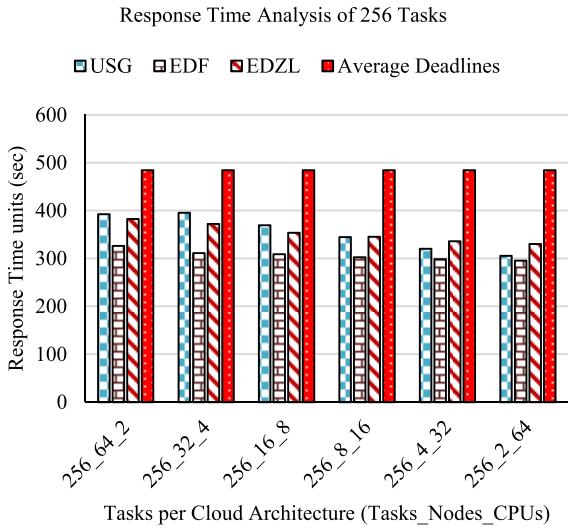

FIGURE 19. Average Response Time Analysis of 16 Tasks.

USG and EDZL. However, the important observation is that all the algorithms produced average response times within the corresponding average deadlines in all the simulations. This implies that all the deadline exceptions were successfully handled by each algorithm and all the tasks could be completed before their deadlines. Further, for the same number of provisioned extra resources, USG outperformed EDZL with respect to the response time. Nevertheless, EDZL often requested for more resources owing to its earliest deadline policy.

VI. CONCLUSION

In this study, we developed a cloud computing architecture and algorithms for soft and near-hard real-time scheduling. The idea was to utilize the features of cloud computing in the dynamic allocation and release of computing resources in response to workload needs. For this purpose, deadline look-a-head schedulers were developed for the firing of deadline exceptions before their occurrence. When such exceptions are detected, the VM node sends the tasks causing them to the MN. The MN then attempts to assign the tasks to an empty processor. In the absence of such a processor, a new VM node is dynamically created and assigned the task. The results of the demonstrative implementation of the proposed architecture and algorithms in the present study were expressed in terms of the number of deadline exceptions fired by each algorithm, the number of extra resources provisioned to each algorithm to handle the deadline exceptions, and the average response time of the tasks. The USG algorithm was found to fire the lowest number of deadline exceptions, and require the lowest number of extra resources, followed by the EDZL and EDF algorithms, respectively. In terms of the response time, EDF produced the shortest response time, although it also recorded the highest number of deadline exceptions. USG outperformed EDZL in terms of the response time for the same provisioned number of resources. These results suggest the applicability of the scheduling of periodic real-time tasks to cloud computing. This assumes that the start-up time and communication limitations of virtual machines would be overcome in the near future through advancements in virtual machine technologies [41].

The non-consideration of some other important scheduling parameters such as cost and energy consumption constitutes a limitation of the present study. Further experimental study is planned to consider such parameters as secondary priorities in addition to meeting the constrained deadlines of real-time tasks.

REFERENCES

[1] A. Burns and A. J. Wellings, *Real-Time Systems and Programming Languages*, 4th ed. Toronto, ON, Canada: Pearson Education, 2009.
[2] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, vol. 24, 3rd ed. New York, NY, USA: Springer, 2013.
[3] J. W. S. Liu, *Real-Time Systems*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.
[4] R. Mall, *Real-Time Systems: Theory and Practice*. London, U.K.: Pearson, 2009.
[5] H. Kopetz, *Real-Time Systems*, 2nd ed. New York, NY, USA: Springer, 2013.
[6] J. A. Stankovic and K. Ramamritham, "What is predictability for real-time systems?" *Real-Time Syst.*, vol. 2, no. 4, pp. 247–254, 1990.
[7] I. Lee, J. Y. Leung, and S. H. Son, *Handbook of Real-time and Embedded Systems*. Boca Raton, FL, USA: CRC Press, 2007.
[8] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Multiprocessor scheduling by reduction to uniprocessor: An original optimal approach," *Real-Time Syst.*, vol. 49, no. 4, pp. 436–474, 2013.
[9] G. Nelissen, V. Berten, V. Nélis, J. Goossens, and D. Milojevic, "U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks," in *Proc. 24th Euromicro Conf. Real-Time Syst. (ECRTS)*, Pisa, Italy, Jul. 2012, pp. 13–23.
[10] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Proc. 22nd Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2010, pp. 3–13.
[11] S. Funk and V. Nanadur, "LRE-TL: An optimal multiprocessor scheduling algorithm for sporadic task sets," in *Proc. 17th Int. Conf. Real-Time Netw. Syst.*, 2009, pp. 159–168.
[12] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *Proc. 12th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2006, pp. 322–334.

[13] E. Massa, G. Lima, P. Regnier, G. Levin, and S. Brandt, "OUTSTANDING PAPER: Optimal and adaptive multiprocessor real-time scheduling: The quasi-partitioning approach," in *Proc. 26th IEEE Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 291–300.

[14] J. Lee, A. Easwaran, and I. Shin, "Laxity dynamics and LLF schedulability analysis on multiprocessor platforms," *Real-Time Syst.*, vol. 48, no. 6, pp. 716–749, 2012.

[15] U. C. Devi, "Soft real-time scheduling on multiprocessors," Ph.D. thesis, Univ. North Carolina, Chapel Hill, NC, USA, 2006.

[16] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surveys*, vol. 47, pp. 63:1–63:33, Jul. 2015.

[17] J.-T. Tsai, J.-C. Fang, and J.-H. Chou, "Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm," *Comput. Oper. Res.*, vol. 40, pp. 3045–3055, Dec. 2013.

[18] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, and J. Kołodziej, "Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing," *Future Gener. Comput. Syst.*, vol. 51, pp. 61–71, Oct. 2015.

[19] Y.-J. Chiang, Y.-C. Ouyang, and C.-H. Hsu, "Performance and cost-effectiveness analyses for cloud services based on rejected and impatient users," *IEEE Trans. Services Comput.*, vol. 9, no. 3, pp. 446–455, May/Jun. 2016.

[20] E. N. Alkhanak, S. P. Lee, and S. U. R. Khan, "Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities," *Future Gener. Comput. Syst.*, vol. 50, pp. 3–21, Sep. 2015.

[21] L. Mao, W. W. Lin, B. Liu, and Y. Da Li, "An energy-efficient resource scheduling algorithm for cloud computing based on resource equivalence optimization," *Int. J. Grid High Perform. Comput.*, vol. 8, no. 2, pp. 43–57, 2016.

[22] G. K. Karthikeyan, P. Jayachandran, and N. Venkataraman, "Energy aware network scheduling for a data centre," *Int. J. Big Data Intell.*, vol. 2, no. 1, pp. 37–44, 2015.

[23] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *J. Syst. Softw.*, vol. 99, pp. 20–35, Jan. 2015.

[24] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Apr./Jun. 2014.

[25] M. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proc. Inf. Process.*, 1974, pp. 807–813.

[26] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *Proc. IEEE Region 10's 9th Annu. Int. Conf.*, Aug. 1994, pp. 607–611.

[27] J. Lee, A. Easwaran, I. Shin, and I. Lee, "Zero-laxity based real-time multiprocessor scheduling," *J. Syst. Softw.*, vol. 84, pp. 2324–2333, Dec. 2011.

[28] H. Alhussian, N. Zakaria, and A. Patel, "An unfair semi-greedy real-time multiprocessor scheduling algorithm," *Comput. Electr. Eng.*, vol. 50, pp. 143–165, Feb. 2016.

[29] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *Proc. IEEE 32nd Real-Time Syst. Symp. (RTSS)*, Nov./Dec. 2011, pp. 104–115.

[30] G. Nelissen, S. Funk, and J. Goossens, "Reducing preemptions and migrations in EKG," in *Proc. 18th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2012, pp. 134–143.

[31] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Toyama, Japan, Aug. 2011, pp. 15–24.

[32] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "DP-fair: A unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Syst.*, vol. 47, pp. 389–429, Sep. 2011.

[33] S. Funk, "LRE-TL: An optimal multiprocessor algorithm for sporadic task sets with unconstrained deadlines," *Real-Time Syst.*, vol. 46, pp. 332–359, Dec. 2010.

[34] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Proc. 27th IEEE Real-Time Syst. Symp.*, Rio de Janeiro, Brazil, Dec. 2006, pp. 101–110.

HITHAM ALHUSSIAN received the B.Sc. and M.Sc. degrees in computer science from the School of Mathematical Sciences, Khartoum University, Sudan, and the Ph.D. degree from Universiti Teknologi Petronas, Malaysia, where he is currently a Lecturer with the Computer and Information Sciences Department. His main research interests are in real-time systems, parallel and distributed systems, and big data mining.

NORDIN ZAKARIA received the Ph.D. degree from Universiti Sains Malaysia, in 2007, working in the area of computer graphics. Since then, his areas of research has been diverse, encompassing high-performance computing, quantum computing, motion capture, and visualization. He is currently heading the High-Performance Computing Center, Universiti Teknologi Petronas.

AHMED PATEL received the M.Sc. and Ph.D. degrees from Trinity College Dublin, specializing in packet switched networks. He is currently a Full Professor with Jazan University, Saudi Arabia. His research interests include networking, security, forensic computing, and distributed systems. He has authored 260 publications and co-authored several books. He is a member of the Editorial Advisory Board of International Journals and has participated in Irish, Malaysian, and European funded research projects.

AYMAN JARADAT received the B.Sc. degree from Yarmouk University, Jordan, in 1989, the M.Sc. degree from University Sains Malaysia, in 2007, and the Ph.D. degree from Universiti Teknologi Petronas, in 2013. He was the Dean of the Faculty of Computer and Information Technology, Al-Madinah International University. He is currently an Assistant Professor with Al Majmaah University. He is specialized in computer science and his research interests include high-performance computing, grid computing, cloud computing, genetic algorithm, and distributed algorithms and applications.

SAID JADID ABDULKADIR received the Ph.D. degree from Universiti Teknologi Petronas, Malaysia, in 2015, where he is currently a Lecturer with the Computer and Information Sciences Department. His area of research interest include, but not limited to, machine learning, big data analytics, and high-performance computing.

ABDELAZIZ Y. AHMED received the B.Sc. degree (Hons.) in electronic engineering from the University of Gezira, Wad Medani, Sudan, in 2006, and the M.Sc. and Ph.D. degrees in electrical and electronics engineering from Universiti Teknologi Petronas, Malaysia, in 2009 and 2015, respectively. In 2016, he joined Universiti Teknologi Petronas, where he is currently a Lecturer with the Electrical and Electronic Engineering Department. The area of focus is microelectromechanical systems (MEMS) actuator/sensor design and microfabrication/sensor technology based on CMOSMEMS technologies.

HUSSEIN T. BAHBOUH is currently an Assistant Professor of electrical and computer engineering with the College of Mechanical and Electrical Engineering, Damascus University. His main research interests include microprocessor design, FPGA, and hardware testability.

SALLAM OSMAN FAGEERI is currently an Assistant Professor and the Dean of the Faculty of Computer Science and Information Technology, Alzaeim Alazhari University, Khartoum, Sudan. His main research interests are on data mining, machine learning, and big data analytics.

ASIM ABDALLAH ELSHEIKH received the Ph.D. degree from Universiti Teknologi Petronas. He is currently an Assistant Professor with the Department of Information Technology, University of Tabuk, Saudi Arabia. His main research interests are on statistical database security as well as database concurrency.

JUNZO WATADA received the B.Sc. and M.Sc. degrees in electrical engineering from Osaka City University, Osaka, Japan, and the Ph.D. degree from Osaka Prefecture University, Sakai, Japan. He is currently a Professor with Universiti Teknologi Petronas, Malaysia, a Research Professor with the Research Institute of Quantitative Economics, Zhejiang Gongshang University, China, and a Professor Emeritus with Waseda University, Japan. His research interests include big data analytics, soft computing, tracking systems, knowledge engineering, and management engineering and finance engineering. He is a Life Fellow of the Japan Society for Fuzzy Theory and Intelligent Informatics and the Biomedical Fuzzy System Association. He is the President of the International Society of Management Engineers and the Vice President of the Forum of Interdeciprinary [A24] Mathematics based in India. He received the Henri Coanda Medal Award from Inventico in Romania, in 2002, and the GH Asachi Medal from Universitatea Tehnica GH Asachi, IASI, Romania, in 2006.