sustainability

MDPI

Article

Towards the Design of a Formal Verification and Evaluation Tool of Real-Time Tasks Scheduling of IoT Applications

Shabir Ahmad [1], Sehrish Malik [1], Israr Ullah [1], Dong-Hwan Park [2], Kwangsoo Kim [2] and DoHyeun Kim [1,*]

[1] Department of Computer Engineering, Jeju National University, Jeju 63243, Korea; shabir@uetmardan.edu.pk (S.A.); serrym29@gmail.com (S.M.); israr.ullah@jejunu.ac.kr (I.U.)
[2] Electronics and Telecommunications Research Institute, Daejeon-si 34129, Korea; dhpark@etri.re.kr (D.-H.P.); enoch@etri.re.kr (K.K.)
* Correspondence: kimdh@jejunu.ac.kr

Received: 29 November 2018; Accepted: 27 December 2018; Published: 3 January 2019

Abstract: Real-Time Internet of Things (RT-IoT) is a newer technology paradigm envisioned as a global inter-networking of devices and physical things enabling real-time communication over the Internet. The research in Edge Computing and 5G technology is making way for the realisation of future IoT applications. In RT-IoT tasks will be performed in real-time for the remotely controlling and automating of various jobs and therefore, missing their deadline may lead to hazardous situations in many cases. For instance, in the case of safety-critical and mission-critical IoT systems, a missed task could lead to a human loss. Consequently, these systems must be simulated, as a result, and tasks should only be deployed in a real scenario if the deadline is guaranteed to be met. Numerous simulation tools are proposed for traditional real-time systems using desktop technologies, but these relatively older tools do not adapt to the new constraints imposed by the IoT paradigm. In this paper, we design and implement a cloud-based novel architecture for the formal verification of IoT jobs and provide a simulation environment for a typical RT-IoT application where the feasibility of real-time remote tasks is perceived. The proposed tool, to the best of our knowledge, is the first of its kind effort to support not only the feasibility analysis of real-time tasks but also to provide a real environment in which it formally monitors and evaluates different IoT tasks from anywhere. Furthermore, it will also act as a centralised server for evaluating and tracking the real-time scheduled jobs in a smart space. The novelty of the platform is purported by a comparative analysis with the state-of-art solutions against attributes which is vital for any open-source tools in general and IoT in specifics.

Keywords: Internet of Things; wireless sensor networks; real-time systems; scheduling; embedded systems

1. Introduction

In this era of innovation, we are experiencing a shift from physical space to a cyberspace [1], where every physical thing could be interconnected to form a mashup of things. This mashup is called Internet of Things (IoT) [2–5]. Technologies like Quick Response (QR) codes and Radio Frequency Identification (RFID) are being exploited to identify physical objects. Similarly, the power of wireless sensor networks, wireless mesh networks, mobile networks and wireless Local Area Network (LAN) is employed to interconnect these smart objects. These connections, forming an IoT space, greatly enhance the business processes concerning efficiency and costs in enterprise systems [6–10]. In such smart space, every physical object connecting to IoT can be regarded as an edge node of it. These edge nodes

can sense ambient scenarios and bridge the gap between the real world and networks [11]. The huge number of links is enabling the interaction between edge nodes which generate an enormous amount of data which can't be stored and processed on these edge nodes due to their limited capabilities. To address the challenge, cloud technologies [12] have been utilised which deals with huge sets of data.

The use of cloud technologies has been dramatically increased due to the pervasive nature of IoT as it acts as a central server to process and hosts a massive volume of contextual data emitting from edge-nodes of IoT [12]. The IoT and cloud are so often integrated that it is recently known as Cloud of Things (CoT) [12–14]. In a recent study, cloud technologies have been overwhelmingly employed in smart homes and smart cities. In smart cities domain, some notable efforts have been made to highlight the state-of-the-art and overview the relevant technologies [15]. Visvizi et al. [16] debate about the difference of smart cities and megacities and delineate the focus and scope of smart cities and megacities respectively. Moreover, as a fact of interdisciplinary research [17], the demand of cloud technologies are spotted for the big data processing. It is also envisioned that with the realisation of Industry 4.0, it would not be possible to process such an enormous chunk of manufacturing data without cloud technologies.

In IoT applications, sometimes the interaction among IoT edge nodes has real-time requirements, and thus their behaviour needs to be determined, and the deadline must be formally proved beforehand. In the research by Chen [18], it has been said that the tasks which edge nodes need to be scheduled and deployed on smart objects in the same fashion as in conventional real-time systems. In conventional real-time systems, task scheduling is regarded as a trivial issue, and thus many studies conducted for conventional real-time systems can be appropriated in real-time IoT applications.

The development of real-time IoT based systems is an emerging application area which has attracted significant research attention in recent past. Currently, soft real-time systems such as streaming applications are supported by many existing IoT platforms [19,20]. Recently, the term RT-IoT was introduced by Chen et al. to address the associated challenges of both soft and hard real-time systems [18]. Specialised operating systems are developed to support real-time applications requirements in IoT based embedded systems. RT-IoT poses a significant challenge because of the additional Internet layer. Much like in traditional real-time applications, a task can have a valid deadline which must be ensured. One scenario in which tasks miss the deadline is the unrealistic system requirement and selection of inappropriate scheduling policies in accordance with them. In RT-IoT in addition to the careful selection of scheduling policies, communication delay also plays a vital role [21]. Therefore, extensive testing and simulation need to be carried out to understand the behaviour of both the scheduling policy and the communication medium before deploying them in a real situation. For instance, in safety-critical applications, if an edge-node associated with an IoT server hosted on an embedded device failed to meet the deadline, it could lead to a catastrophic consequence or even a human loss. Since the communication delay is not deterministic, so the hard real-time IoT systems are not realistic in the current state-of-the-art.

There are some notable efforts which model deterministic communication delay. EtherCAT [22], RTnet [23] and time sensitive networking (TSN) which is currently standardised by IEEE 802.1 Working Group [24] are some of the examples. However, it is still very early to say that with the maturity of these technologies hard real-time systems can be realised. Moreover, with the realisation of 5G technologies, future IoT applications may also target hard real-time systems if the upper bound on the delays is 100% deterministic. Nevertheless, regardless of the type of real-time systems, extensive testing and schedulability analysis must be formally carried out to determine the behaviour of the application program and the scheduling policies. In literature, this is called feasibility analysis of real-time applications. The objective of the study presented in this paper is to design and implement an application that can support feasibility analysis and verification of existing and newly proposed scheduling algorithms for task scheduling for any category of real-time IoT systems.

As it is mentioned earlier, in the realisation of RT-IoT, the communication medium is a massive barrier in a sense that even if the scheduling policies are well-designed, and the feasibility analysis

ensures that no deadline will be missed, but it still can't guarantee that the task will reach on time because the delay of IoT networks are always unpredictable. To minimise the latency and increase the response time, one solution is the use of edge computing technologies, in which the edge nodes are moved closer to IoT devices [25,26]; however, they still can't model the delay of the communication medium in a deterministic way. Moreover, 5G technologies have also found an excellent way for the accurate prediction of the delay because of the ultra-low latency the upper bound on delay can be determined. Recent research studies by Aliyu et al., Nunna et al. and Condoluci et al. propose that 5G can pave the way toward modelling latency and end-to-end communication delays which were a significant barrier in older days [24,27–29].

Many tools exist for conducting a feasibility analysis and schedulability testing of conventional real-time systems, but they mostly use outdated desktop technologies and thus, cannot adapt well with IoT and CoT. The significance of this work is that it is a novel attempt to design and implement a cloud-centric tool which serves as a platform in which the feasibility analysis of different sizes of task sets are conducted, and the schedulability testing is performed in a simulated environment with the help of modern front-end visualisation libraries. It will help in understanding the behaviour of a given RT-IoT application by supposing a maximum delay of the communication medium. One significance of this architecture is that it is a web-based approach, and allows remote access of the system which is regarded as one of the preliminary requirements of IoT applications. Moreover, the proposed architecture provides a central point to conduct the feasibility analysis of input IoT tasks under certain scheduling policies. Thus, tasks are only deployed in a real situation if it guarantees not to miss the deadline or miss the deadline in the allowed range. Additionally, the cloud-centric nature of the tool allows to track and evaluate the real-time IoT tasks using any conventional scheduling policies like Rate Monotonic (RM) and Earliest Deadline First (EDF). The scheduling footprints of the system will allow users to track and monitor the systems in real-time. The effectiveness of the proposed tool is evaluated with virtual simulated tasks using the aforementioned algorithms and then with actual physical IoT tasks deployed on IoT edge nodes. The proposed tool presents HTML5-based modern user interfaces to help users with the status of currently executing tasks and scheduling footprints of the previously executed processes. This tracking can prove vital in IoT smart spaces, i.e., smart cities, in a sense that by looking the scheduling footprints, end users can get the idea of the status of currently executing tasks and all completed tasks. The main contributions of this paper are as follows:

- Design and Implementation of a cloud-centric tool to overcome the shortcomings in traditional real-time systems by enabling remote access to the tool.
- Provide an environment in which feasibility analysis of input IoT tasks under different scheduling policies by considering a maximum bound on communication delays.
- Provide a space where a scheduled job in a smart space will be monitored and tracked without the need to go on premises.

In this paper, the authors aim at presenting a novel approach based on web technologies to overcome the shortcoming found in the earlier tools and proposes a generic tool to work effectively both in traditional real-time computing as well as modern IoT applications. The proposed tool leverages the latest tools and frameworks to more effectively visualise tasks, understand them and undergo feasibility analysis which is considered necessary for real-time tasks scheduling. Moreover, the proposed tool is hosted in the cloud to receive contextual data from the IoT edge nodes and track IoT tasks in the real-time.

Web-based visualisation of many research projects is getting more attention lately. In the Bio-medical field, specifically, more than 50% of the projects exploit web application to inspect and visualise patient care units. For instance, in the research by Benítez et al., Lu et al., Côté et al. and Liu et al. [30–33], a web-based application is designed to inspect and visualize crucial information pertaining patients. Surprisingly, in real-time task scheduling analysis, the use of web-based tools is not commonly utilised despite their many facets and advancement. Therefore, this paper is the first

attempt to encourage and use these technologies and implement a visualisation tool where real-time tasks can be simulated and visualised and additionally, these tasks are deployed, remotely monitored and tracked in a typical IoT application.

The rest of the paper is organised as follows; Section 2 discusses related work and relevant research in this area. Section 3 presents the Input Tasks system model and provides a detailed overview of the tasks types and attributes. Section 4 illustrates the conceptual architecture of the proposed system and describes the main component of the architecture. Section 5 provides the sequence of operations of the system and provides a detailed description regarding the sequence diagram. Section 6 illustrates the implementation details and provides guidelines for running the proposed architecture on general purpose system and embedded systems. Section 7 describes the results and snapshots of the system for the RM and EDF algorithm. Additionally, this Section also describes the virtual tasks simulation and physical tasks simulation in a typical IoT context. Section 8 gives an overview of communication delay modelling and performance evaluation using different scenarios. Section 9 covers the comparison and significance of the proposed work. Section 10, finally, concludes the paper and identifies future directions within the scope of this work.

2. Related Work

RT-IoT system at their core, intersect with modern cyber-physical systems and traditional real-time system [18,34]. Ahmad et al. [34] provided a brief comparison of the difference between conventional real-time and contemporary real-time systems. Notable similarities are the scheduling policies, tasks' types and worst-case analysis. In contrast, the major difference is communication flow with mixed priority and the dependence of delay due to the remote communication over the Internet. Till recent past, the realisation of RT-IoT was not possible due to the communication delays but since the introduction of mobile edge computing and 5G technologies the end-to-end delays and latencies can be ultra-reliable which is necessary for RT-IoT systems [28,29,35]. For a certain upper bound of delay, the challenge is fall back on the implementation of effective tools which can study the scheduling policies and ensure that no tasks will be missed by considering the maximum delay specified by the upper bound [27]. Such simulation tools present an appropriate solution for the development and testing of strategies that aim to identify and fix problems before their application in a real environment [36].

2.1. Overview of Real-Time Systems Scheduling and Visualisation Tools

The main research problem addressed in this paper is to investigate the existing tools for the simulation of real-time systems and formally prove the deadline of these systems in IoT applications. To cope with this, the efforts in real-time systems to propose such tools are discussed. Real-time systems are one of the oldest research areas, and in its early development stage, numerous simulators were designed for general purpose real-time systems. In the early 1990s, some notable advancement has been made. For instance, a tool named STRESS [37] was proposed by Audsley et al. which considered the first notable tool of it's time for real-time task simulations. It was written in a new programming language called STRESS and had the flexibility of adding new algorithms for simulation without modifying the core. Another effort came in the form GHOST by Sensini et al. [38], which supported diverse types of tasks in contrast to the previous efforts which only supported a single category of tasks, e.g., periodic tasks. In [39] an attempt was made to design a simulation tool for hybrid task sets on distributed systems with the support of shared resources and also had a native visualisation in its core. However, the maintenance state of this project is unknown, and the source code is not open-source.

In the post 2000 era, simulation tools like MAST [40], VizzSchedular [41], ARTISST [42], STORM [43], Realtss [44], Cheddar [45] and Simso [46] have been proposed. Their end goal was to support simulation and feasibility analysis of more recent scheduling policies. They used relatively newer technologies offering better user experience interfaces. However, these tools also face the same problem of keeping up with the latest trends, and the maintenance state of many of them

are unknown. Additionally, the visualisation techniques implemented in these tools are based on desktop technologies and do not consider a more dynamic IoT context in which the tasks are remotely configured and scheduled without the need to go on premises. Finally, these tools are purely aimed at simulation and visualisation of tasks and do not focus on monitoring of input tasks in real-time IoT scenario [43,45]. The majority of these tools are no longer maintained, and their source code is not open.

In the recent past, there have been several efforts focused towards building simulation tools which can effectively simulate and visualise the behaviour of real-time systems in the context of IoT. A recent study has been conducted by Ahmad et al. on these real-time task scheduling tools to highlight the shortcoming of the traditional tools when utilised in the context of IoT, and the need to have a tool which focuses on latest web-based technologies rather than conventional, outdated technologies are encouraged [47]. The difference between the traditional simulation tools and the modern IoT-based real-time simulation tools is not subtle. Most of the constraints like deadline, energy and power hold same for IoT application as well. However, in the latter case, the real-time operations and jobs should be performed on a much larger scale, and thus the applications need to degrade gracefully by employing proper load-balancing techniques [48].

Nevertheless, with the application of real-time systems in IoT and industry 4.0, there is still a dire need of a testbed to simulate the footprint of the system to understand the behaviour to a full extent. One of such efforts is put by Tao et al. which is named as SDMSim [49] to realise and understand the intelligent supply-demand modelling in smart factories. The tool supports the formulation of supply-demand, statistical analysis and visualisation; however, it is not generic in purpose and very specific to the scenario for which it is proposed. ScSF [50] by Rodrigo et al. is another similar effort based on high performance computing (HPC) systems. This tool presents an environment to simulate workloads and tasks for large-scale tests. A more recent trend is to add an insight on why the scheduling takes a certain chunk in a scenario and a TaskInsight tool [51] is proposed to characterise the memory behaviour of different task schedulers through analysing of the data used between different tasks. This way it can diagnose a scheduling decision culprit of high memory utilisation. Other modern tools like MCRTsim [52], Score [36], iMOPSE [53] and ABEMAT [54] are also worth considering. However, as it is said earlier, one thing which is common in all these tools is they are designed with a specific goal in mind, and thus the domain of all of them is very narrow. A more general-purpose tool similar to STRESS but for large-scale data in IoT and smart industries application is the demand of time, and this paper is the first attempt towards it.

2.2. Overview of IoT Application to Support Activity Tracking and Monitoring

The second research challenge which is addressed in this paper is the dire need to have a tool which can track and monitor real-time jobs without the need to go on premises. Ahmed et al. [55] first proposed the advantage of remote monitoring and tracking. Monitoring and tracking activities in a typical IoT smart space is getting more attention these days. For instance, intelligent tools are designed to track the routine of the older adults in smart homes. Sensors installed in home alliances and other equipment to monitor the activities of the older adults and the sensor contextual data is analysed to identify an irregular pattern. The intelligence is applied to contextual data which has been hosted on cloud platforms [56,57]. Other events are also tracked like falling of older adults because it is very critical due to their weak health conditions [58]. In the health domain, many pieces of research focus on daily life routine tasks inside a smart home and are being tracked [59]. However, these applications also address very narrow and dedicated requirements and cannot be reused in other domains. Similarly, there are broad categories of IoT platforms whose sole aim is to ease up and speed up the development of IoT applications. These applications are exposing their APIs and consumers IoT application leverages those APIs to make real-time smart IoT applications with a little effort. FIWARE [60] is one of such open-source platforms designed for the above-mentioned purpose. However, these applications do not allow any analysis of real-time algorithms and feasibility analysis

and using them needs technical skills. Nonetheless, all of the aforementioned tools are detecting a certain pattern and not focus on real-time scheduling algorithms' based tracking and monitoring which this paper, to the best of authors' knowledge, is the first attempt to approach.

3. System Model

IoT applications comprise of edge nodes, IoT gateways and IoT servers [34,55]. IoT servers listen to requests from application users as depicted in Figure 1. The request is communicated over the Internet using IP protocol. IoT servers are deployed on embedded hardware like Raspberry PI, Arduino or Intel Edison Board. IoT gateways are physical devices or software programs that connect cloud and IoT servers. One role of an IoT gateway is to translate user request in one protocol to another protocol [4]. For instance, an IoT front-end application interacting with end users listens to HTTP requests, but IoT servers are communicating with physical objects listens to Constraint Application Protocol (CoAP) requests, so one role of the gateway is to translate the HTTP request to a CoAP request. Similarly, edge nodes are physical objects connected locally with IoT gateways and perform specific tasks. For instance, in a smart home, a temperature sensor keeps sensing room temperature after a certain period. Similarly some edge nodes responsible for actuating an alarm, ring alarm only on the occurrence of a particular event. That being said, IoT tasks are also of two broad categories; periodic tasks which repeat themselves after a predefined interval called period and event-driven tasks which are executed on the occurrence of a specific event.

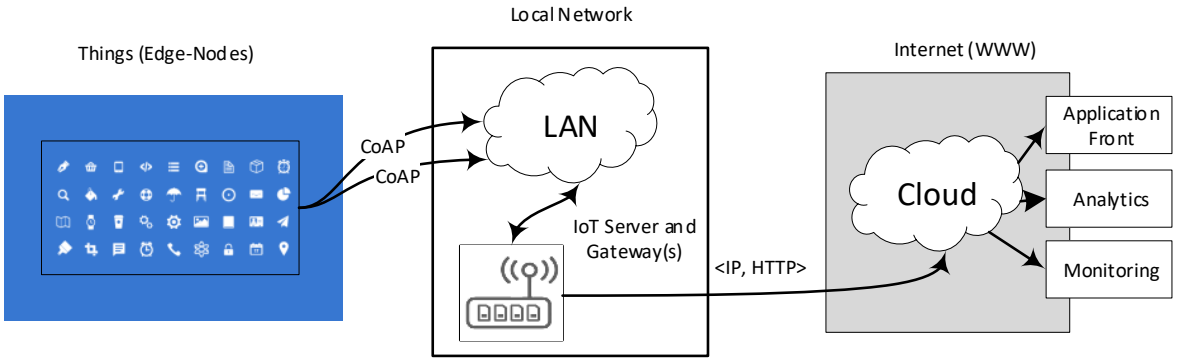


Figure 1. General System Model of IoT Applications.

As discussed in earlier sections that the core scheduling theory can be applied in IoT domain [18] so the system modeling can be formulated by leveraging the conventional tasks models. Consider in a typical IoT smart space, edge nodes are performing certain tasks in real-time. We consider a set of n independent tasks denoted by $[\tau_1; \tau_2; ...; \tau_n]$ which are executed on IoT resources. For periodic tasks, an infinite identical tasks are activated at constant rate determined by their period. Let τ_i has a period $P_i \in R$ after which the task is reactivated then every release of task, also called instance of that particular task, $k \in Z$ takes place at time

$$r_{ik} = \phi_i + kP_i$$

where ϕ_i is the release time of ith task's zeroth instance.

The list of all the symbols and acronyms are summarised along with their definitions in Table 1.

Table 1. Summary of the symbols and acronyms and their description.

Symbol/Acronym	Description
τ_i	ith Task.
P_i	Period of ith Task.
ϕ_i	The release time of ith Task's zeroth instance.
r_{ik}	The release time of ith Task's kth instance.
c_{ik}	The active execution time of ith Task's kth instance.
end_{ik}	The ending execution time of ith Task's kth instance.
WR_i	Worst case response time of ith task.
BR_i	Best case response time of ith task.
H	Hyperperiod of set of n tasks.
n	Total number of tasks.

It should be noted that instance number k can be negative, i.e., we have endless repetitions towards both ends of the time axis. $\phi_i \in R$ is the time of release also called the phase of the zeroth instance of τ_i, which serves as a reference release. The first release of all tasks called the phasing ϕ of the tasks. It has been assumed that we do not have control on the phasing of the tasks, for example, since the tasks are released with the occurrence of some external events, so we assume that any arbitrary phasing may occur. This assumption is common in real-time scheduling literature [61–63]. At any point, the resource is used to execute the first priority task that has some work in the pending. So in case of preemptive scheduling policies, when a task τ_i is being executed, and a release occurs for a higher priority task τ_j, then the execution of τ_i gets preempted, and will resume when the execution of τ_j has finished, as well as all other releases of tasks with a higher priority than τ_i that have taken place in the meantime. For simplicity, we assume the overhead from switching to be negligible. Figure 2 shows an example of the above. In the example, the priorities of the tasks are rate monotonic [63], i.e., a task with a smaller period has a higher priority. However, the analysis in this paper holds for any fixed priority assignment.

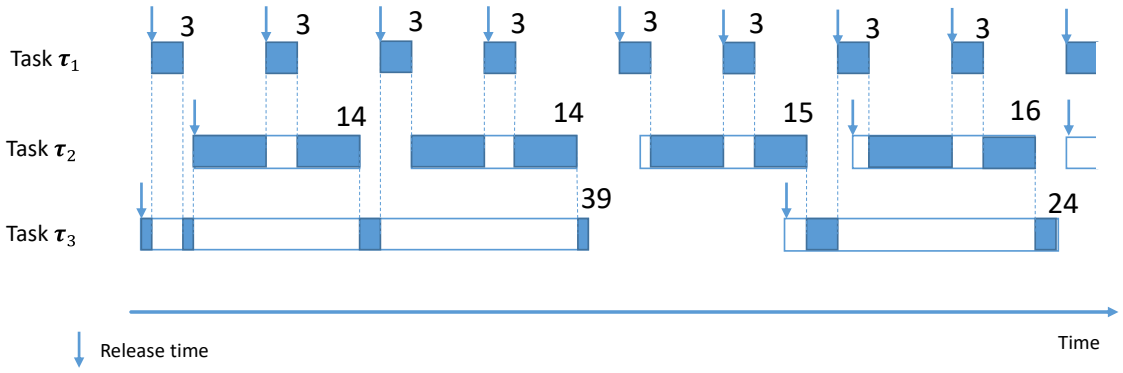


Figure 2. An example of the effect of fixed-priority preemptive scheduling on the execution of three periodic tasks τ_1, τ_2, τ_3, with release periods 10, 19, and 56, and computation times 3, 11, and 5, respectively. We assume that task τ_1 has the highest priority, and task τ_3 has the lowest priority. The numbers to the top right corner of the boxes denote the response times of the respective releases. Note that the response time is counted from the moment of release (indicated by small arrows) up to the corresponding completion.

Definition 1. *The maximum response time of any release of τ_i under any arbitrary phasing is called worst-case response time and is measured using Equation (1).*

$$WR_i = max_{\phi,k}(end_{ik} - r_{ik}) \quad (1)$$

where WR_i is the worst case response time of task τ_i, end_{ik} is execution end time of task instance k and r_{ik} is the release time. It is worth to note that WR_i must fall within the given deadline of the task for all the execution to meet their deadline.

Definition 2. *The best case response time of any task τ_i which is also called the minimum response time is denoted by Equation (2)*

$$BR_i = min_{\phi,k}(c_{ik} - r_{ik}) \tag{2}$$

where BR_i is the best response time, end_{ik} is execution end time of task instance k and r_{ik} is the release time. Best response time is not necessary in most cases for the analysis of the system, but it is sometime good to keep track of it.

Definition 3. *The hyperperiod of periodic input tasks are defined as the time interval after which all the periodic tasks executions are repeated. Hyperperiod **H** of set of n tasks τ with the period P can be found by the Equation (3)*

$$H = lcm\{P_i | 1 \leq i \leq N\} \tag{3}$$

where lcm is the least common multiple of tasks' period.

The hyperperiod is a very crucial factor in a sense that it is a function of CPU computation cycles. The higher the hyperperiod will be the more CPU clocks the scheduling footprints will consume and vice versa. As embedded devices have scarce resources, therefore, it is essential to cautiously model the period of tasks in order to reduce the hyperperiod [34]. For instance, the result of the hyper period of the periods $\{12, 11, 19, 23, 17\}$ is 980,628 which is considered very high, and the CPU will at least iterate this amount of time to cover just one hyperperiod.

4. Conceptual Architecture

Data visualisation and reporting are considered vital for useful results analysis [64,65]. In real-time computing, feasibility analysis and formal proof of the input tasks require more attention since real-time systems, if properly not analysed, could lead to a deadline failure [18,34]. Effective visualisation techniques help greatly analysing such system and can reduce the effort of the analysis with visual effects. Many tools are proposed in different researches to realise the visualisation and simulation of tasks as discussed in introductory sections. Some of the shortcomings of those tools are platform dependency and not keeping up with newer technology trends [47].

Moreover, the majority of them are desktop-based and is not compliant with cloud technologies. The need to have a generic tool which utilises start-of-the-art latest technologies and platform independence to some extent arises lately. The use of cloud technologies is widespread in the IoT context. This paper proposes a cloud-centric web-based tool to enable simulation, visualisation and tracking IoT tasks more adequately. The use of a cloud-centric web-based tool offers several benefits over traditional desktop technologies. First and foremost, the web-based tool can be accessed remotely, so the application user does not necessarily go on premises to use the tool and analyses the tasks. Second, due to the modern visualisation technologies in the web like HTML5 canvas, web containers, javascript, HTML5 and CSS3 the job is getting more comfortable but at the same time more effective. Finally, the open-source maintainers of these technologies are very active and continuously provide patches to reflect on the latest trends. The idea of this paper is to have a visualisation and simulation toolkit based on web technologies which can act as a valuable tool for understanding and undertaking the input tasks and feasibility of them. The initial sitemap of the tool is exhibited in Figure 3. There are three main modules of the tool; Input, Output and Process. Input module deals in tasks generation. A task can be added, edited, uploaded in batches and can be deleted. As discussed in previous sections, tasks can be of periodic nature or event-driven nature. Nonetheless, all tasks have some common attributes like arrival time, deadline, periods, urgency, and slack. Process deals in applying scheduling

algorithms to input process. As a proof of concept, two existing algorithms are considered: EDF and RM. EDF is used for event-based tasks while RM is used for periodic tasks. Lastly, output deals in presenting algorithms' result in effective user experience (UX) interface and leverage the power of HTML5/CSS3 for visualisation of the CPU analysis. It has different interfaces for presenting different scheduling algorithms, but the overall idea of representing scheduling algorithms' outcome in a better UX way is the central focus of all the interfaces.

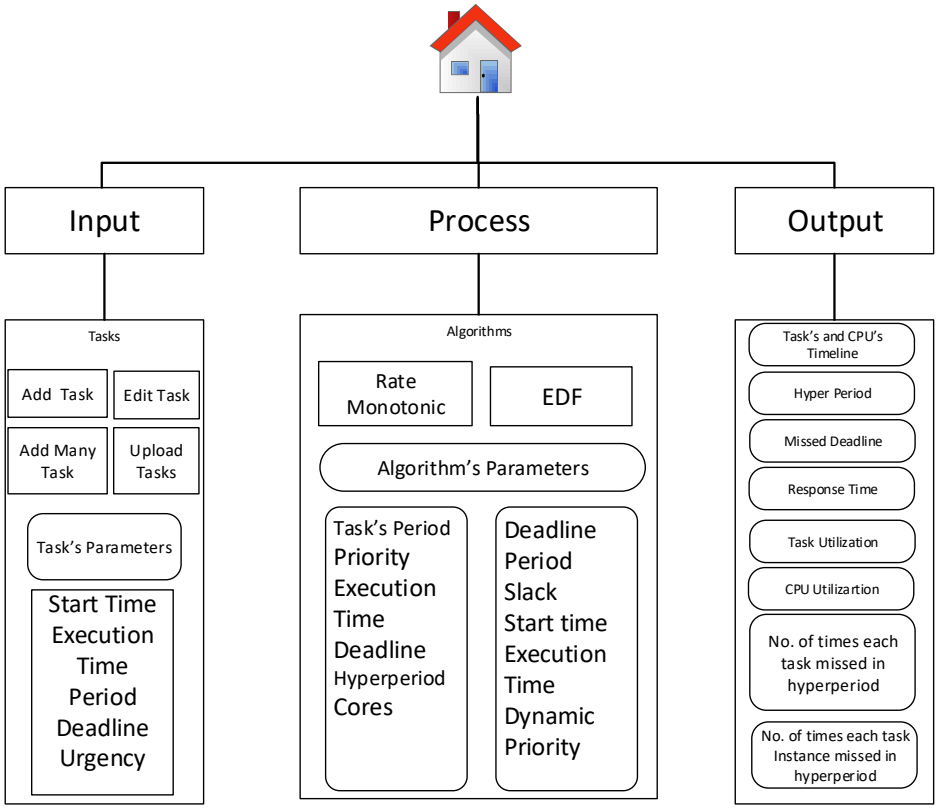


Figure 3. Sitemap of the Proposed Web-based Visualization Tool.

The parameters of input tasks are summarised in Table 2. Table 3, in the same way, identifies and describes the parameters utilised in the process. Finally, Table 4 shows the output performance measures of a specific algorithm.

Table 2. Summary of the Input Tasks' Parameters and their definitions.

Parameters	Definition
Start time or arrival time	The arrival of tasks also called phase of the tasks.
Execution time	The time it requires to execute during its period, in other words, The execution time within its period.
Period	The time after which the tasks repeat itself. This parameter is specific to periodic tasks only, and for sporadic tasks it is irrelevant.
Urgency	Urgency is the quantitative measure of the priority of the tasks. 0 means less urgent, one means highly urgent. It is only for event-driven tasks and irrelevant for periodic tasks.
Deadline	The maximum time limit in which the task must have to be executed otherwise will be considered invalid. For event tasks, the deadline is considered 0.

Table 3. Summary of the Algorithms' Parameters and their definitions.

Parameters	Definition
Start time or arrival time	The arrival of tasks also called phase of the tasks.
Hyper period	The least common multiple of periods of input periodic tasks.
Task's Period	The time after which the tasks repeat itself. This parameter is specific to periodic tasks only, and for event-driven tasks it is irrelevant.
No. of Cores	The number of homogeneous cores on which the algorithms schedule tasks.
Slack and Laxity	The maximum time a task can be delayed on its activation to complete within the deadline.

Table 4. Summary of the Input Tasks' Output Parameters and their definitions.

Parameters	Definition
Tasks' Timeline	The timeline of tasks instances which runs on core within its period.
Total Missed Deadline	The number of instances a task misses the deadline.
Missed Tasks	The number of tasks which don't get any CPU share at all
Response Time	The time after which the tasks starts executed for the first time.
Tasks Utilization	Tasks Utilization is the percentage of tasks currently on CPU.
CPU utilization	CPU utilization is the percentage of CPU utilized by tasks.

Figure 4 shows the architecture diagram of the proposed work where we generate tasks firstly. The input to the tasks is the number of tasks, period range, execution time range, arrival time range as covered in the Table 2. The tasks are generated and passed to the Model View Controller (MVC) controller for processing. MVC controller gets scheduling algorithm from the MVC Model and applies the algorithm. The results are passed to MVC View which in turns gives it to the visualisation component and Web-based Graphical User Interface (GUI). The GUI's interfaces of the input module are event-driven and periodic tasks. Output module interfaces are performance measures like CPU utilisation, missed tasks, and completed tasks. Lastly, the scheduling algorithm which is either RM or EDF constitutes the Process module.

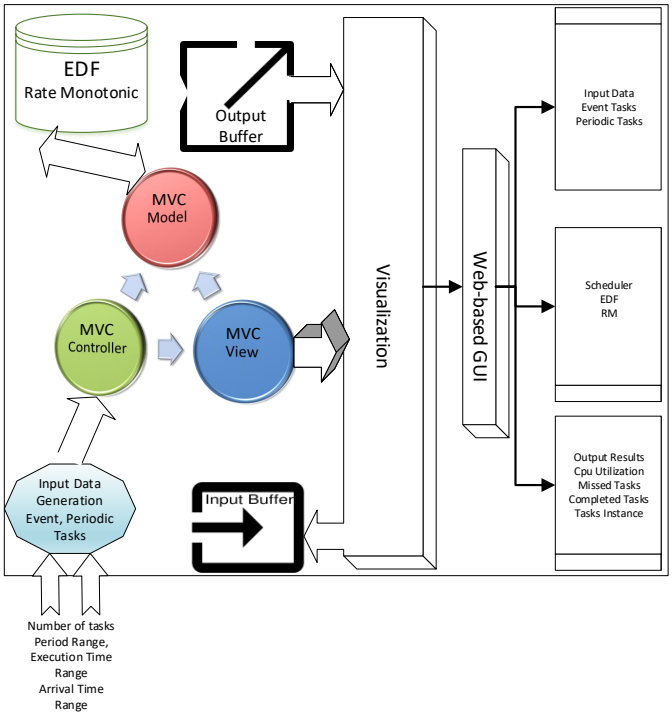


Figure 4. Conceptual Architecture of Cross Platform Communication of General Purpose PC and Raspberry PI.

5. Interaction Model

Figure 5 illustrates the sequence of connectivity among the Task Input module, Input Buffer, Algorithm Controller and Visualization using MVC views. It also provides a general overview of the operation of the configuration. The Tasks Unit is a controller module which first initiates random tasks generations. This module creates tasks based on user input and stores them in a CSV file. The CSV file is then loaded to the Input buffer, and virtual instances of the tasks are created which are ready to be scheduled using an algorithm which is the role of the algorithm unit. The algorithm unit loads the tasks instances from the input buffer and applies any algorithms specified in the request. Application users can interact with the visualisation module of the system. The user login to the system and make a GET request to understand the response of input tasks using algorithms described earlier. The result is presented in HTML format and is being rendered to the App User by MVC view. It can be seen from Figure 5 that the tasks have been passed via terminal or some sensors and the scheduler schedules the task and the result has been communicated back using Restful web services.

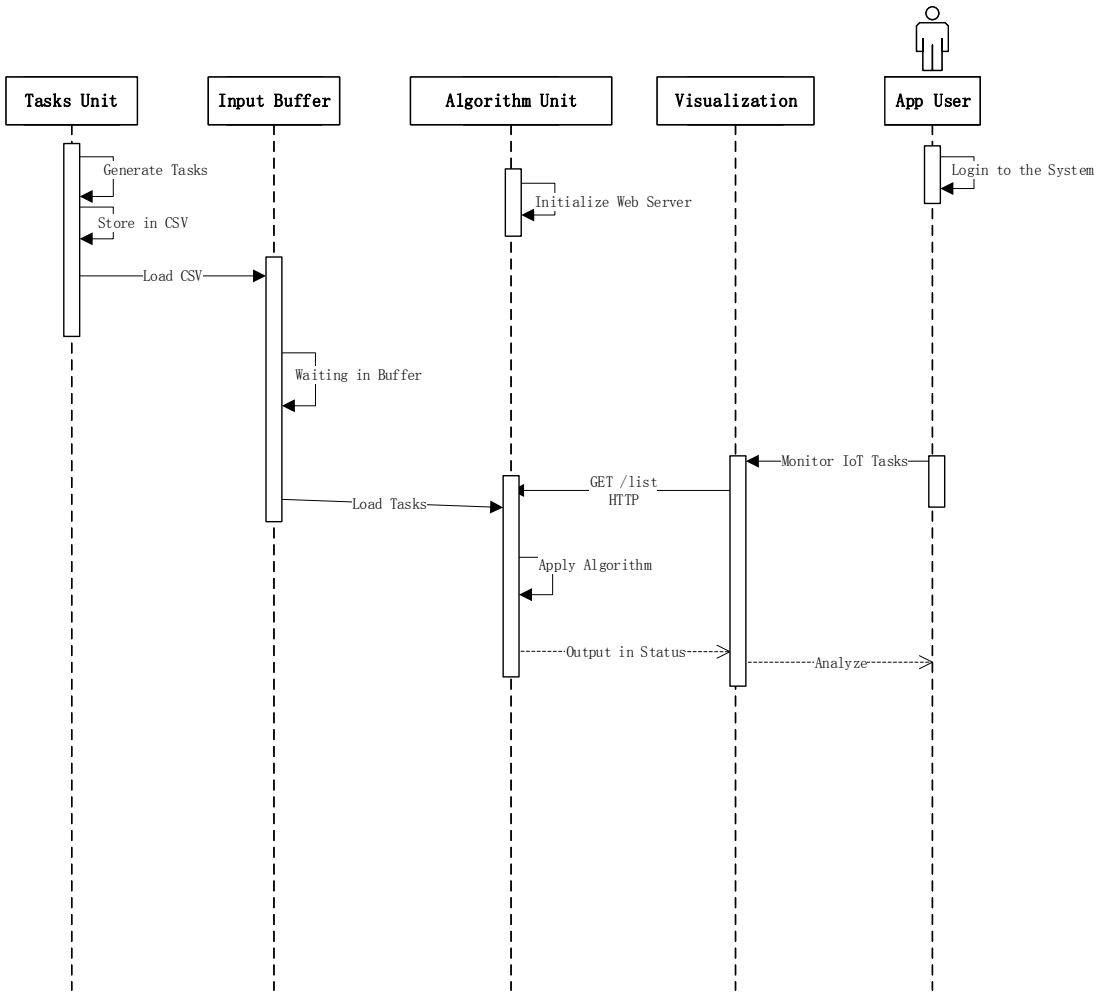


Figure 5. Sequence of interaction of different modules of Proposed Tool.

The focus of this paper is two-fold; first, to design and implement a web-based graphical interface tool aimed to simulate various algorithms and to present the result of algorithms more cleanly and robustly. Second, the use of the proposed system in an IoT environment where multiple physical tasks are scheduled, and their performance is tracked and monitored using the proposed system.

Figure 6 demonstrates that input tasks are admitted for scheduling using any scheduling algorithm. In the first case, virtual tasks are added to assess and analyse the behaviour of input task sets for any chances of missing the deadline. The virtual tasks are not real tasks but rather simulated tasks to carry out the feasibility analysis of the scheduling algorithm under a certain task load. In the second case, once it is confirmed that tasks are guaranteed to meet the deadline, physical tasks are added, and the results based on the input algorithm is visualised. The proposed Web-based application is used to aid in the visualisation of results. The results of both iterations are then evaluated and analysed in better UX interfaces.

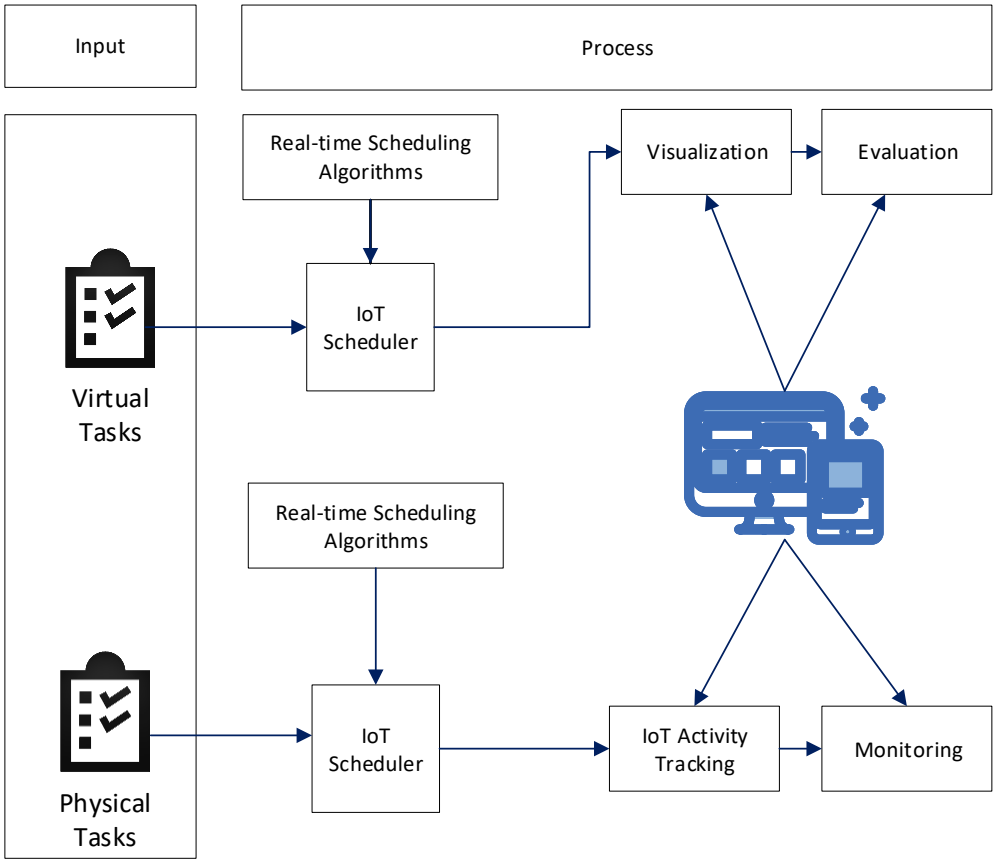


Figure 6. Input and Process Modules Conceptual flow in Experimental Setup.

6. Implementation Details

This section provides an overview of the implementation tools and techniques used to develop the system. The proposed tool is re-using some of the interfaces and technology stack outlined in [55]. The tool is intended to be run on general purpose machine hosted on the cloud. We will use the cloud to host the web-based tool. For this, we are using EC2 Compute node of Amazon Web Services [12]. Specialized services like Azure IoT and AWS IoT exist, but they offer a more dedicated services like data insight and analytics on sensory contextual data. Moreover, they are relatively over-expensive for the proposed work. Since the work presented in this paper is not a full-fledged IoT application, therefore it is best to opt for a compute node. Table 5 describes the technologies used on general purpose machine.

Table 5. Specification of Implementation Environment of Cloud-Centric Web Application hosted on AWS.

System Parameter	Value
Operating System	Linux AWS EC2 Compute Node
CPU	Intel (R) Core(TM) i5-4570 CPU @ 3.20 GHz
Primary Memory	12 GB
IDE	PyCharm, Sublime Text 3
Framework	Flask MVC
Core Programming Language	Python 3
Libraries	Jinja 2, CSV Parser, Bootstrap 3, HTML 5/CSS3
Persistance	CSV

The proposed tool communicates with Raspberry PI based IoT server to which numerous physical resources are connected. The physical resources which are discussed in later sections are attached to it. The cloud-based tool and the Raspberry PI based IoT server communicate using HTTP protocol. The IoT server listens to the request which is the scheduled instance of the task. It parses the request and performs the task on actual resource. The specifications of the Raspberry PI based implementation environment are demonstrated in Table 6.

Table 6. Specification of Implementation Environment of IoT Server.

System Parameter	Value
Rasberry PI based IoT Server	
Hardware	Raspberry PI 3 Model B
Operating System	Raspbian
Memory	1 GB
Server	Flask Webserver
Resources	LED, Fan, Temperature Sensor, Humidity Sensor, Breadboard, Expansion Board, Connecting wires, e-Health Sensor Platform
Libraries	GPIO, CSVReader, Jinja Template, Bootstrap 3, HTML 5/CSS3
IDE	Vim, PyCharm (Remote Access)
Programming Language	Python 3

We use Python as a core programming language. Python is a general purpose and common programming language. It is widely used for developing web-based applications, desktop-based applications, data analysis and simulations. It has vast community and strong developers bench. We use Python for three reasons. First, it is equally popular among web programmers and scientist, and second, it is really easy to learn and adapt and last, it has strong API support. In this research, we used Python 3.5. Moreover, following a design pattern for implementing the tool is time-saving and leads to a very organised source code which can be easily maintainable if required [66]. Consequently, we used Flask which is an MVC based framework [67]. Flask is a mini framework based on model view controller architecture. It is a mini-framework in a sense that it does not utilise the full-fledged features of an MVC framework. It uses only those modules which are required on demand so brings efficiency in its core. The modules of the framework are loosely coupled and can be used as per need. For front-end technologies we use HTML5, CSS3 and JavaScript. We also use Bootstrap 3, a powerful front-end framework for reusing the code and making the tool responsive to run on mobile screens [68]. The major part of the presentation has been developed with the help of the Jinja templating engine. The controller is responsible for all the processing tasks. It takes requests, parses them, and processes them. For processing the requests, it needs data which is taken from the model. After processing the requests, the processed data is given to the view. The view generates a response in the form of HTML or JSON. Figure 7 shows typical flow of an MVC Pattern.

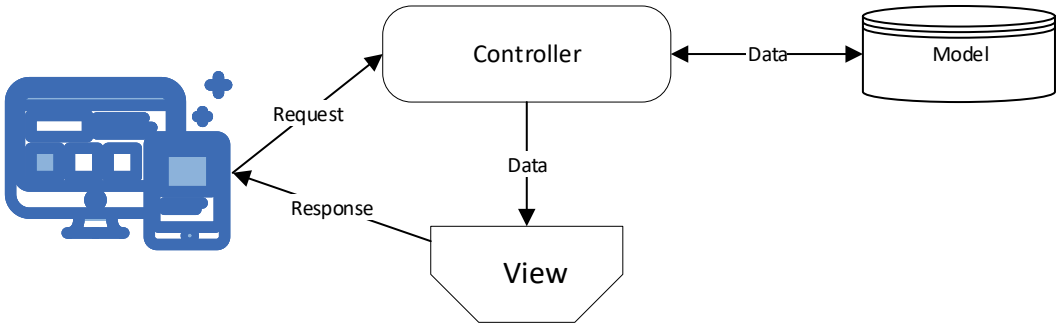


Figure 7. Model View Controller Design Pattern Information Flow.

7. Execution Results

This section illustrates the execution of the proposed system and provides some snapshots illustrating the process of execution. Figure 8 shows the implementation environment for the proposed tool. The Raspberry PI 3 which is used as an embedded device has some physical resources connected to it. These resources can do specific tasks. The resources are a fan actuator, led lights and sensors. The fan actuator and temperature sensors are considered as periodic non-critical tasks and periodically scheduled after a predefined time interval. The led light is considered as event-driven tasks and will be generated only in certain circumstances. Similarly, there are also some resources which perform critical tasks such as ECG and pulse oximeter. These resources are performing critical tasks and connected to e-Health Libilium platform [69] as shown in Figure 8. In the subsequent subsections, a detailed mechanism of tasks generation and schedulability analysis using the proposed tool is investigated and discussed. As discussed in earlier sections, that the tool is evaluated with simulated virtual tasks and then with actual physical tasks based on the resources connected to Raspberry PI, so it mainly analyses the system using the two approaches.

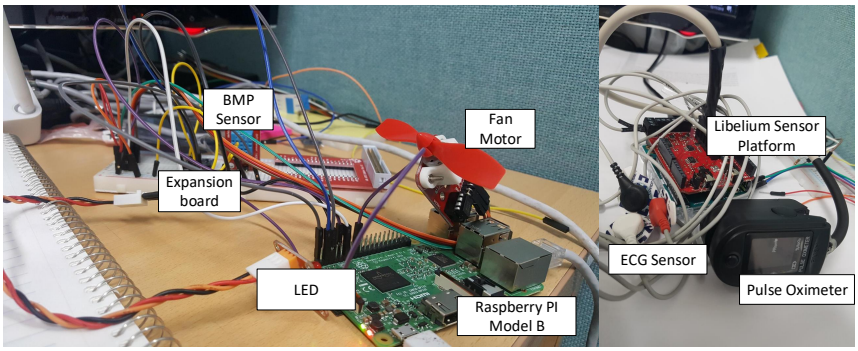


Figure 8. Implementation Environment based on Raspberry PI.

7.1. Analysis Using Virtual Tasks

First off, the scheduling policies are assessed using arbitrary task sets. Task sets are generated based on the input attributes of the task. As described earlier, a task can have attributes like period, execution time, priority, arrival time and type of task, i.e., periodic or event-driven as common in various studies [34]. Figure 9 shows the interface showing a web form to allow the users to add input tasks. For each attribute, a form field is created to let the task designers enter the data. The data have been validated, and if the form field is not filled out or contains an invalid value, an exception is thrown, and as a result, the form does not proceed further. The data validation is crucial in any system design since if the parameters are given wrong values this will have a drastic effect on the overall results of the system and there will be endless anomalies in the overall flow of the system.

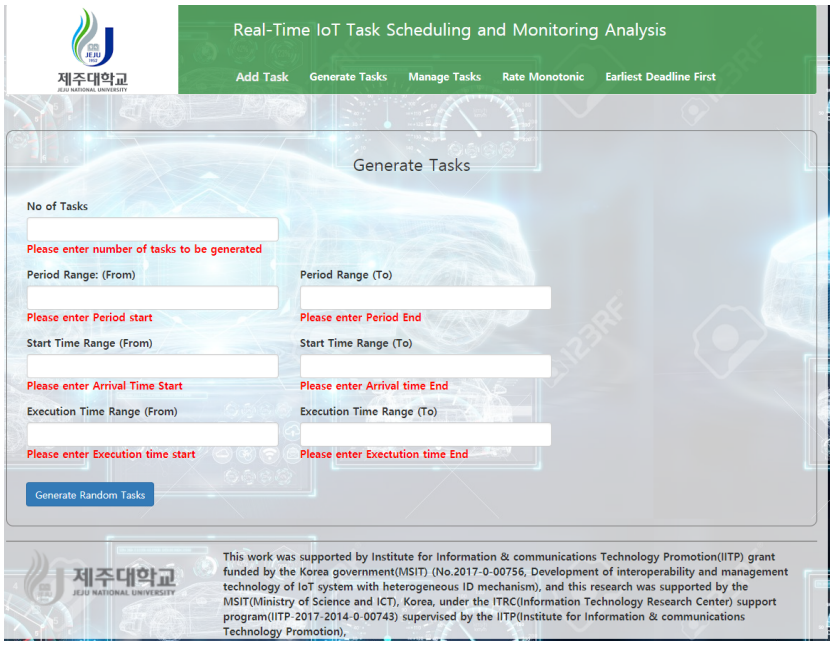
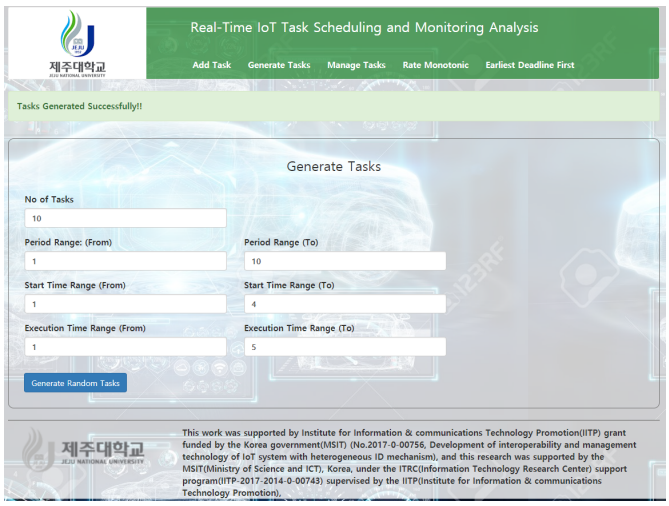


Figure 9. Snapshot of Task Data Validation.

If the validation is passed and the form is submitted as shown in Figure 10a, a success message is displayed indicating the successful generation of tasks file. The resultant CSV file is shown in Figure 10b. The CSV contains tasks, and each column represents the attributes of that particular task. For event-driven and sporadic tasks, the period attribute is zero while for periodic tasks, this has some non-zero value. It is worth noting that this form allows users to generate random tasks; however, a user can add tasks of their desired input parameters too. Once the input task load is generated and persisted in CSV, then the same task set can be consumed by any scheduling policies and perform feasibility analysis. If the feasibility analysis indicates that the CPU never misses any task's instance and at the same time the CPU is also not under-utilised, then the claim that the tasks are guaranteed to meet the deadline given the communication delay is known to be under certain upper bound.



(**a**) Web-Form Interface for Adding Tasks

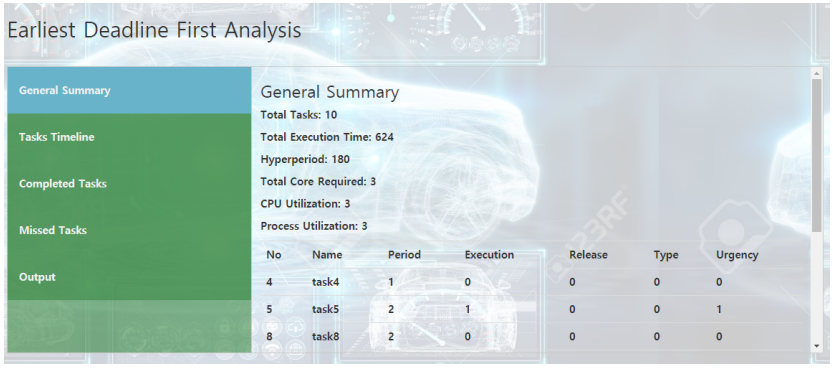
	A	B	C	D	E	F
1	Tasks No	Period	Execution	Start Time	Deadline	Urgency
2	1	5	1	0	5	0
3	2	2	0	0	2	0
4	3	5	2	0	5	0
5	4	5	2	0	5	1
6	5	2	1	0	2	1
7	6	2	1	0	2	0
8	7	3	2	0	3	0
9	8	3	1	0	3	0
10	9	5	1	0	5	1
11	10	4	2	0	4	1

(**b**) Tasks Persistance result in the form of CSV

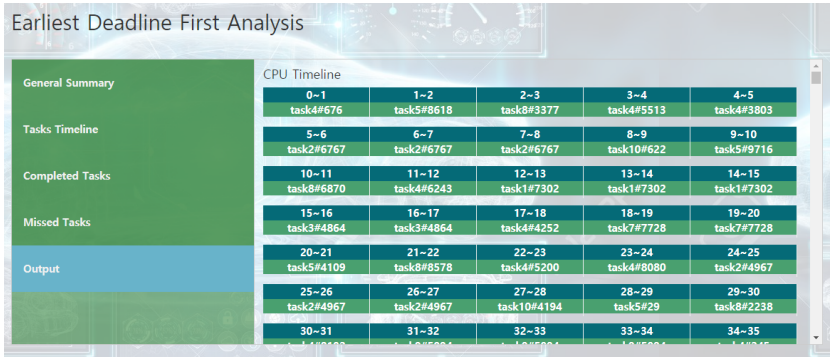
Figure 10. Snapshots of Tasks Addition and CSV Generation using the Web Interface.

The tasks are scheduled in real-time using any of the fixed-priority or dynamic-priority algorithms. For simplicity, we use RM and EDF approach, but this tool can support the simulation of any real-time tasks scheduling algorithms. Whatever the algorithm may be, the proposed tool has certain tabs on

the visualisation page of the algorithm. These tabs are named General Summary, Tasks Timeline, Completed tasks, Missed tasks and CPU timeline. Every tab contents are next to the tab on the same page. The feasibility analysis under EDF scheduling policy is shown in Figure 11. In Figure 11a, the general summary section is exhibited which accounts for summarising the overall process. It includes items like what are the input tasks and their parameters, hyper period of the tasks, total core needed for efficient simulation, CPU utilisation and process utilisation. For instance, it shows total tasks as 10, total execution time as 624, total core required as 3, CPU utilisation and process utilisation as 3 followed by input tasks representation in a tabular format.



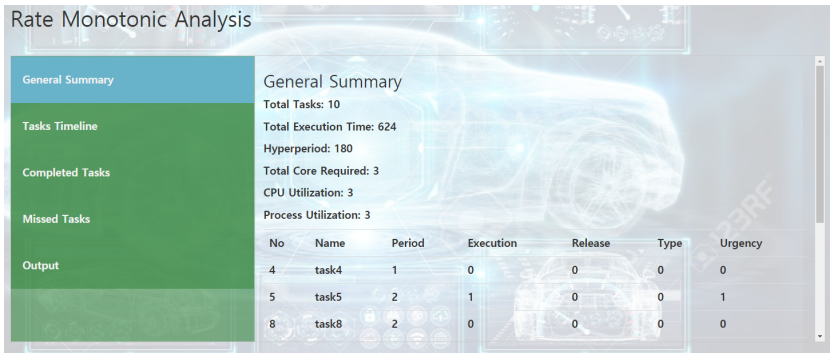
(**a**) General Summary of Hyperparameters of the Scheduling policy



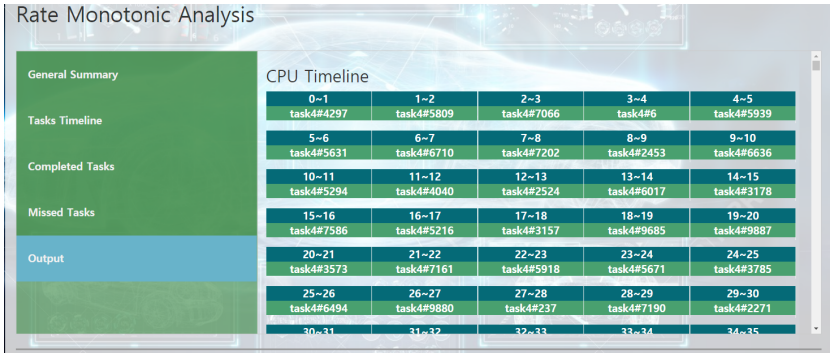
(**b**) Stacked CPU Timeline Showing the Clock Cycles and the Tasks Instances Schedules against it

Figure 11. Snapshot of EDF Algorithm's simulation using Virtual Tasks.

Figure 11b portrays the CPU timeline of the scheduling footprints of EDF. There are two strips stacked on one another. The upper strip shows the clock cycle while the bottom strip shows the tasks instance which gets CPU on that particular clock cycle. For instance, on the 0 1 clock cycle, task 5's instance 676 is scheduled and for next clock cycle task 5's instance 8618 is scheduled, and this goes on till the clock cycles equal the hyper period. Like Figure 11, Figure 12 illustrates the snapshots of some of the screens of the RM algorithms using the same task set. Figure 12a shows the general summary interface as in the case of EDF. The general summary interfaces list the overview of the input load and hyperparameters which are necessary for RM scheduling policy. These hyper-parameters include hyper period, total execution time, number of core required, CPU utilisation and process utilisation. These parameters define the fate of the feasibility analysis at first glance. Figure 12b shows the CPU timeline which is also two strips stacked on one another as in the case of EDF. However, the tasks are scheduled differently under RM policy. For instance, The first two clock cycles of the RM is both allocated to task 4, while in case of EDF only the first of it is allocated to task 4. The reason why the scheduling footprints of these scheduling policies are beyond the scope of this work.



(**a**) General Summary of Hyper parameters of the Scheduling policy



(**b**) Stacked CPU Timeline Showing the Clock Cycles and the Tasks Instaces Schedules against it

Figure 12. Snapshot of Rate Monotonic Algorithm's Simulation using Virtual Tasks.

7.2. Analysis Using Physical Tasks

The second major part of this research work is to allow users to schedule jobs and monitor them remotely without the need to go on premises. For instance, a user can schedule a real-time job in an IoT space like smart home and made them run using any of the real-time tasks scheduling algorithms. The system will provide the interface for the user to schedule these jobs and track them even if the user is not on his premises. Moreover, the CPU timeline will let the user know which task is executing at which instance and which task is missed at a particular instance. This way the IoT smart space will behave autonomously without the interaction of users and more importantly without the need to be on the location. That being said, to analyse and verify the correct working of the algorithm using actual physical resources, we use Raspberry PI 3 as a central server to which the physical resources are connected. Figure 13 shows the conceptual architecture of the proposed system. Application users logged into the proposed tool which is a web-based application hosted on AWS EC2 as described in Table 5. The user can generate tasks and apply an algorithm to evaluate and visualise the performance and output results of the algorithm. In the same way, if the tasks are physical tasks in a smart world, the tasks can be generated and associated with the respective IoT resources as presented in the smart space studies [55]. The communication between the cloud-based web application and Raspberry PI based IoT node is performed using the HTTP protocol. The Raspberry PI based IoT server parses the request and perform the task on associated physical devices. This way the end goal, which is the monitoring and the evaluation of any algorithm on actual IoT jobs, are fulfilled and can be remotely accessed from anywhere.

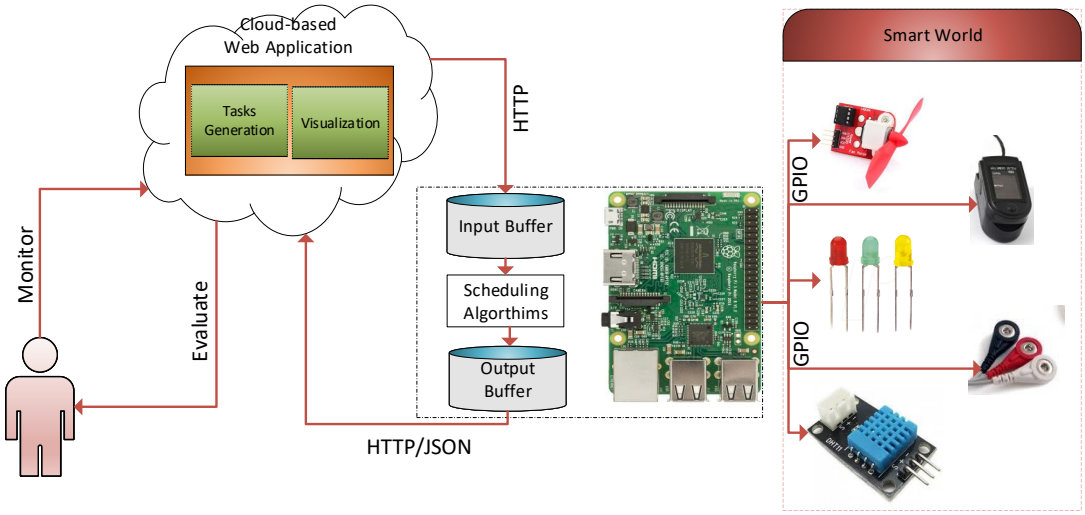


Figure 13. Experimental Setup Design using Raspberry PI and AWS Cloud.

Table 7 describes the tasks which are associated with physical IoT resources. There are mainly two types of tasks; critical tasks which have priority as 1 and non-critical tasks which have priority as 0. For non-critical tasks, temperature sensor, Led Actuator and Fan actuator have been taken whereas for critical tasks e-Health sensor platform's two sensors are taken for getting the ECG and pulse of any elderly under observation.

Table 7. Physical Tasks Set for Analysis.

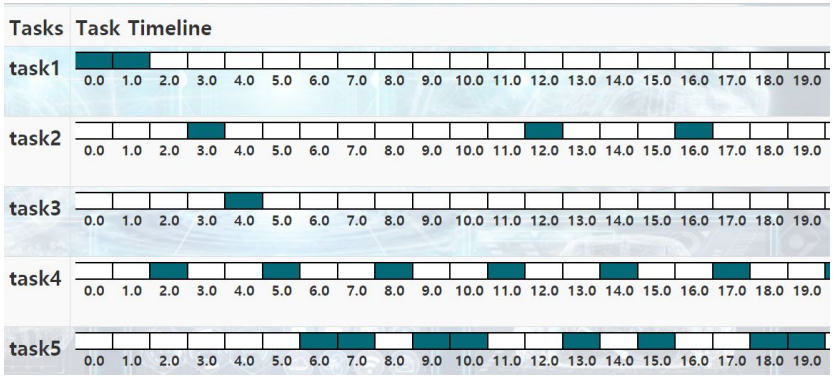
Task No	Task Name	Description	Period	Execution Time	Phase	Deadline	Priority
Task1	GetTemp	This will read current temperature from Temperature sensor for execution time of 2 s and this will be repeated every 10 s.	10	2	0	10	0
Task2	BlinkLed	This will repeatedly turn on and off the light LED for 1 s and this will be repeated every 5 s.	5	1	2	5	0
Task3	TurnOnFan	It will turn on the fan actuator and off it after 2 s and this will be repeated every 15 min.	15	2	4	15	0
Task4	getECG	This is a periodic critical task which will take the ECG values from e-Health sensor platform.	3	1	0	3	1
Task5	getPulse	This is a periodic critical task which will take the Pulse of the user from e-Health sensor platform.	4	2	1	4	1

It is to note that we periodically simulate the tasks for repeatedly assessing the response time and the execution within the deadline. The tasks are created using the web interface against every resource, and the input parameters are provided in the form field in accordance with the Table 2.

In order to see the various tasks in action, the tasks associated with physical resources are made to communicate with the Raspberry PI on a predefined time specified by the main algorithm. The physical resources communicate using general purpose I/O (GPIO), and the status of the algorithms are displayed in runtime on bash terminal as well as on the connected device in case of the actuating devices like a fan.

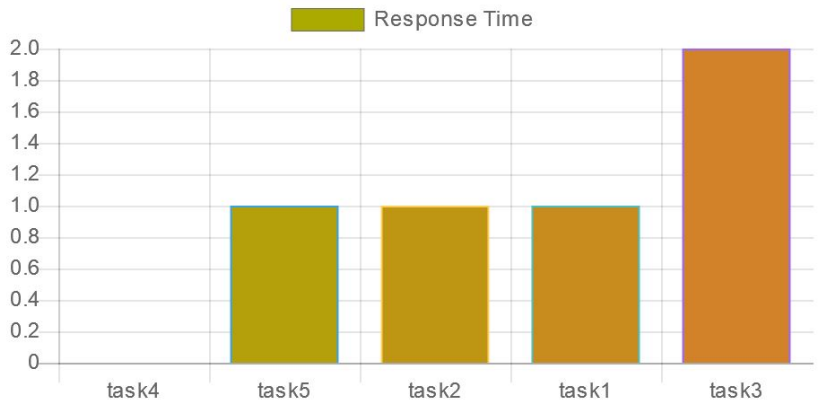
Figure 14 shows the interfaces when the tasks associated with physical resources are scheduled using RM algorithm. As described earlier, we consider two categories of tasks; critical and non-critical tasks which are identified by unique colour codes in the system. If at a specific time, two tasks of critical and not-critical nature are waiting in the system, it will give priority to the critical system.

In Figure 14a, the task timeline is shown for every task. The rows show the CPU share of the individual task while the columns list all the tasks. In Figure 14b, the CPU timeline is illustrated in clock cycles of 1 epoch. For instance, in the clock interval 0–1 task 1 instance, which is getting temperature value from the sensor, gets executed and the reading is displayed on bash terminal of Raspberry PI. At this point, non-critical tasks task1 and task2 are in the waiting state the CPU is given to the critical task task4 which is reading ECG. The majority of the pink colour codes confirms that the system always gives more priority to critical tasks. Figure 14c illustrates the performance visualisation in terms of response time of every task. The response time of task3 is maximum, as it has to wait for critical tasks task4 and task5 to finish execution. Overall, the response time of critical tasks is less than non-critical tasks.



(**a**) Task Timeline

(**b**) Stacked CPU Timeline Showing the Clock Cycles and the Tasks Instaces Schedules against it



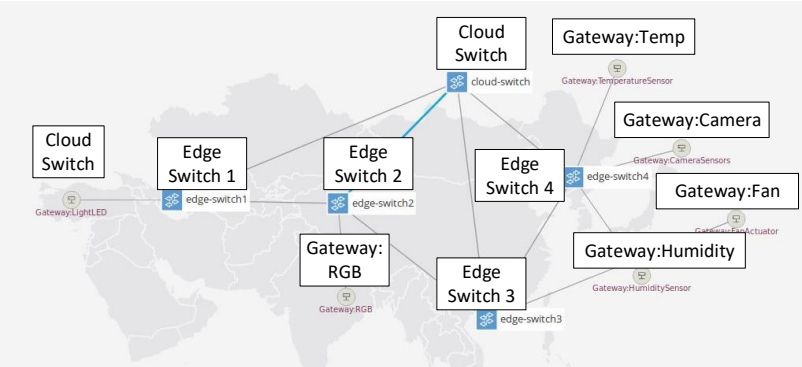
(**c**) Performance Visualisation Interface: Response Time

Figure 14. Snapshot of Rate Monotonic Algorithm's Simulation using Physical IoT Tasks.

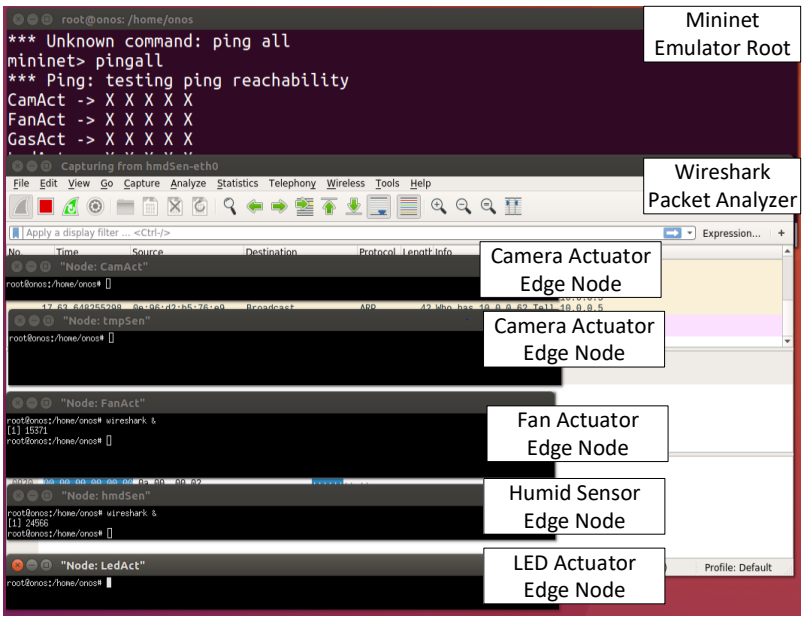
8. Performance Evaluation

In this section of the paper, the performance of the proposed work is evaluated and discussed. The network testbed is based on Mininet emulator [70] which is utilising ONOS SDN controller for

managing the entire network flow [71]. We have built a custom topology which has been exhibited in Figure 15a. The network is based on Asia region, and it has a variety of switches and hosts. The switch named cloud-switch is the gateway for the traffic coming to the edge nodes. Different hosts are connected to the edge switches receiving and sending the traffic to different hosts. There are six hosts in this setup acting as IoT gateways where the sensors and actuators are connected. The Raspberry PI-based hosts are named by their respective resources names. For simplicity, it is assumed that only one IoT resource is connected to the Raspberry PI gateway. The actuators are an LED and a fan whereas the sensors are a temperature sensor, a humidity sensor, and a camera and a gas sensor. Simple flask web servers have been deployed on every host to listen on port 8080, and different hosts are sending data of different sizes and their respective response times are recorded. For every terminal, Wireshark has been run as a daemon process to log the network traffic and different packets as shown in Figure 15b.



(**a**) Network Topology based on Mininet and ONOS SDN controller

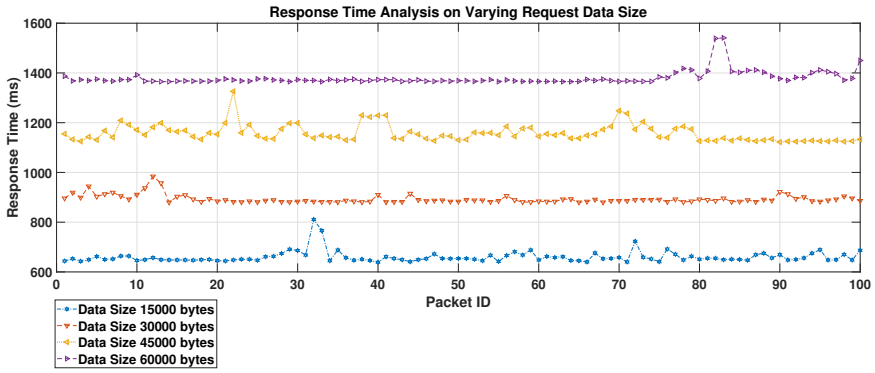


(**b**) Mininet Network Emulator Environment

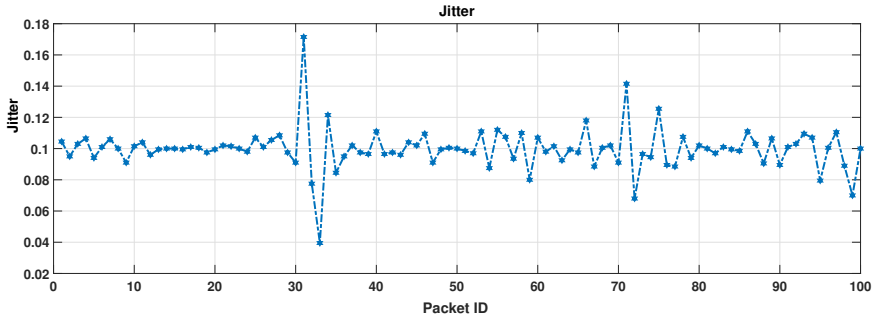
Figure 15. Snapshots of the Network Setup for Performance Analysis.

The edge-to-edge communication is emulated with a low communication delay of 10 ms while the cloud to edge communication is emulated with a high delay of 100 ms. Figure 16 shows the response time and jitter of the interpacket delay of the communication between Camera Sensor and Fan Actuator. For this setup, we have taken the link bandwidth between 760 kbps to 1 Mbps and the communication delay of 105 ms. Moreover, the communication between different hosts have

been performed with varying sizes of data and the response time is recorded. Figure 16a exhibits the response time in milliseconds. For a low bandwidth of around 1 Mbps, the response time is very low even for a massive data of size 15,000. As the data size is increased, the response time also keeps increased but overall the response graph is steady, and if no external disruptions occur, the upper bound on the communication delay can be determined. This can also be verified by the jitter graph which is shown in Figure 16b. Apart from the randomness on the packet number 33, the graph is steady which proves that the communication medium is stable and the inter-packet delay is uniform in many cases.



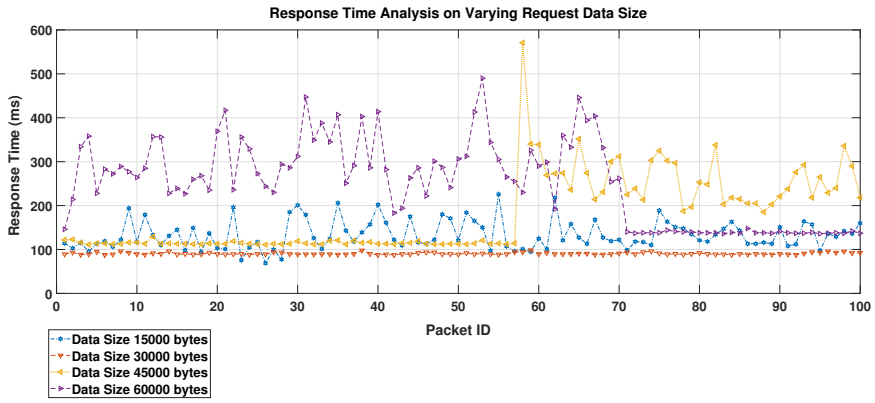
(**a**) Response Time



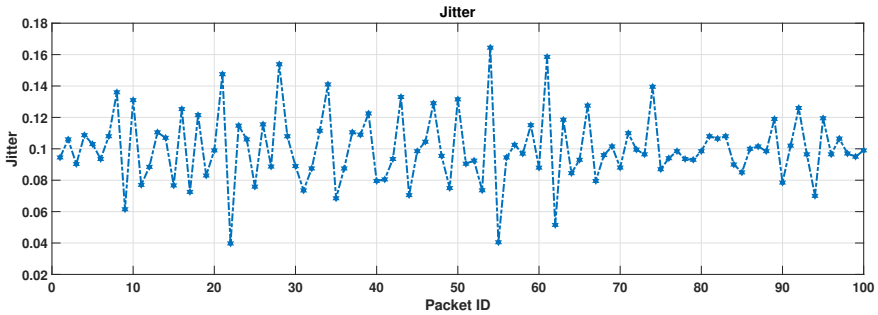
(**b**) Jitter

Figure 16. Performance Analysis: Case 1: Cloud-to-edge Communication, Delay = 100 ms, Bandwidth = 1 Mbps.

For the latter case, the delay is decreased 10 ms, and the bandwidth is increased to 9 Mbps, and it has been found the effect of the response time for the same data sizes are very abrupt. Figure 17 shows the response time and jitter of the interpacket delay of the communication between Gas Sensor and Led Actuator. In this case, the link bandwidth between 9 Mbps and 12 Mbps and the communication delay of 10 ms. Similarly, the communication between different hosts have been performed with the same data sizes as the previous case and the response time is recorded. Figure 17a displays the response time in milliseconds. It is evident that when the bandwidth is increased, and the communication delay is decreased which is usually in the case of edge-to-edge communication, the response time is very little as compared to the former scenario. As the data size is increased, the response time also keeps increased but overall the response graph is steady, and if no external disruptions occur, the upper bound on the communication delay can be determined. The jitter in this can also prove the stability of the medium which is shown in Figure 17b. Apart from the randomness on the packet number 33, the graph is steady which demonstrates that the communication medium is stable and the inter-packet delay is uniform in many cases.



(**a**) Response Time



(**b**) Jitter

Figure 17. Performance Analysis: Case 2: Edge-to-edge Communication, Delay = 10 ms, Bandwidth = 10 Mbps.

9. Comparison and Significance

In this section of the paper, we emphasise the significance of this work and provide a comparison with the existing tools that have already been used. As described in the earlier sections of the paper, this work is the first ever attempt to provide a general-purpose simulation and visualisation of the real-time behaviour of IoT applications. The proposed tool is also novel in a sense that it leverages web-based technologies hosted on AWS cloud unlike previous traditional desktop-based tools like STRESS and SimSo. Another unique feature of the proposed tool is that it acts as a central point to manage IoT nodes and schedule various tasks in a smart space. There are a few solutions that are believed to be similar. STORM and VizzSchedular, for instance, which are described in the introduction of the paper are considered more close to this research. However, the proposed approach is different in many aspects; first, the proposed tool is very generic and can be utilised for traditional real-time systems, IoT application and even smart industries. Second, it is using the latest technology stack to get benefited from the cutting-edge technologies. Lastly, it is hosted in AWS cloud which provides the added benefits of intelligent and monitoring of different IoT tasks. Table 8 summarises various tools and their comparison with the proposed work.

Table 8. Comparative Analysis of Proposed Tool with existing state-of-the-art tools and testbeds.

S.No	Name	Cloud Support	Open-Source	Major Domain	Programming Language	Maintanance Status	Real Simulation in IoT Environment	General-Purpose	Tool Type	Shared Resources	Data Insight
1	STRESS [37]	No	No	Real-time task scheduling algorithms analysis	C Language	Not maintained	No	Yes	Desktop	No	No
2	GHOST [38]	No	No	Real-time task scheduling algorithms analysis	C Language	Not maintained	No	Yes	Desktop	No	No
3	MAST [40,72]	No	Yes	Scheduling Visualization	Ada	Not maintained	No	Yes	Desktop	Yes	No
4	STORM [43]	No	Yes	Visualization and Simulation	Java	Not maintained	No	Yes	Desktop	Yes	No
5	SimSo [46]	No	Yes	Simulation and Visualization	Python	Maintained for bugs only	No	Yes	Desktop	No	No
6	SDMSim [49]	No	No	Supply-demand matching simulation and analysis	Multiple Languages Support	Unknown	No	No	Web-based	No	No
7	ScSF [50]	No	Yes	Data intensive analysis tool on HPC	Python	Actively Maintained	No	No	Desktop	Yes	Partial
8	TaskInsight [51]	Yes	No	Scheduling Algorithms Memory insight	C/C++	Unknown	Yes	Yes	Desktop	Yes	Yes
9	MCRTsim [52]	No	Yes	Visualization and Simulation	Java	Maintained for bugs only	No	Yes	Desktop	No	No
10	Score [36]	No	No	Energy Consumption Analysis tool	Multiple Languages	Unknown	Yes	No	Web-based	Yes	Partial
11	iMOPSE [53]	No		Resource-Constrained Scheduling	Java	Actively Maintained	No				
12	ABEMAT [54]	No	No	Energy Management tool	Python, C++	Unknown	No	No	Desktop	No	No
13	Location Tracker	Yes	Yes	IoT Context Tracker	Android	Maintained for bugs only	N/A	No	Mobile Application	No	Yes
14	Fiware and DIY testbeds [34,60]	Yes	Yes	IoT API Provider	Javascript	Actively Maintained	Yes	No	Headless Backend	N/A	Yes
15	Proposed Tool	Yes	Yes	IoT Scheduling Simulator	Python	Actively Maintained	Yes	Yes	Web-based	Yes	Yes

The significance of this research lies in the fact that this is the first tool which is built for generic IoT tasks scheduling. Additionally, we believe that this tool provides live simulation using real-time tasks when IoT nodes are connected to embedded IoT gateways. The tool is also platform independent which is very crucial in real-time computing because, as discussed in the introduction, almost in all scenarios real-time tasks are associated with embedded devices, and thus an analysis and visualisation tool which can be ported to an embedded system without any significant revamp to the architecture is highly encouraged.

Another major significance of this work is that the proposed system is web-based so it can be accessed remotely from everywhere. Therefore, the user can track their scheduled tasks anytime from anywhere. One example scenario could be a smart home, where user schedule different real-time tasks. For instance, Turning on AC, Getting Temperature and Humidity and performing some action based on it, Turning on Washing Machine and scheduling it to run stop it after one hour and many other are possible use cases. These tasks can be added to the system and scheduled. The user can track all the tasks and analyse it even if not in the home and thus can smartly make decisions based on the outcome of the algorithm.

10. Conclusions and Future Directions

RT-IoT systems are an emerging field to support real-time systems in the IoT context, where the jobs are not only performed in real-time but also can be communicated over the Internet [18,34]. Investigation on formal verification of schedulability of tasks and communication mediums on which these tasks are sent would pave the way for the realisation of RT-IoT and therefore is the focus of attention in many recent studies [36,50,52]. In this paper, we presented a novel architecture to address the aforementioned challenges in RT-IoT systems and utilise the latest web-based technologies to simulate and visualise the real-time tasks scheduling behaviour in a typical IoT context. We also discussed the results and snapshots of the system and compared them with the existing state-of-the-art tools. To test and monitor the live simulation using physical IoT edge nodes, we made the tool to communicate with Raspberry PI to which physical resources are connected, and the analysis of these physical tasks are carried out. We have also investigated the communication delay among different IoT nodes by leveraging the popular SDN-based tools such as ONOS and Mininet. As demonstrated by our results in different scenarios, communication delay can be determined with high accuracy. The run-time behaviour is monitored and evaluated. In the end, a detailed comparison with its counterparts is provided, and the significance of this work is highlighted in example scenarios like smart home.

10.1. Implication for the Industry

This tool can be utilised in a very effective way in the industries which deals with IoT and real-time operations. One particular example in which this tool can contribute much is in "Industry 4". The tool can allow users and stakeholders to track and monitor the life cycle of critical manufacturing activities and their scheduling footprints will enable them to understand the processes better. If some products manufacturing is not as critical as other, it can also be prioritised. The development of these activities can be monitored from anywhere, and the users need not be on the premises. Additionally, this tool can also help in other IoT fields which requires continuous tracking of certain processes. For instance, in smart farm such as a greenhouse, it can track and monitor the growth of plants and based on the progress the environment can be tuned. Similarly, in smart health industries, it will also have a massive impact due to the remote support of tracking and monitoring.

10.2. Implication for Academia

The proposed tool is based on the latest open-source tools and technologies and utilises better design patterns such as MVC, and moreover, it is intended to be open-source. Therefore, it can pave a great way for students and researcher to use it and in research projects. Additionally, the formal

verification and understanding of real-time systems is always a very hard problem for students in terms of conceptualisation. This tool will help as a great teaching tool to implement simple case studies under a certain scheduling algorithm and understand how the scheduling algorithm behaves in relation to another scheduling algorithm. Consequently, it can help in better understanding in one of the hardest but yet important an research area.

As a future work, the proposed tool can be extended and employed in a real smart home scenario using actual sensors and actuators to track more real parameters like energy consumption and power consumption and determine the actual communication delay profile based on real communication.

Author Contributions: S.A. conceived the idea for this paper, designed the experiments and wrote the paper; S.M. assisted in algorithms implementation; I.U. assisted in model design and simulation; D.-H.P., K.K. and D.K. conceived the overall idea of the paper, proof-read the paper and supervised the work.

Funding: This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00756, Development of interoperability and management technology of IoT system with heterogeneous ID mechanism), and this work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00501, Development of Self-learnable common IoT SW engine).

Conflicts of Interest: The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. Xu, L.D. Enterprise systems: State-of-the-art and future trends. *IEEE Trans. Ind. Inform.* **2011**, *7*, 630–640. [CrossRef]
2. Vermesan, O.; Friess, P.; Guillemin, P.; Gusmeroli, S.; Sundmaeker, H.; Bassi, A.; Jubert, I.S.; Mazura, M.; Harrison, M.; Eisenhauer, M.; et al. Internet of things strategic research roadmap. *Internet Things-Glob. Technol. Soc. Trends* **2011**, *1*, 9–52.
3. Friess, P. *Internet of Things-Global Technological and Societal Trends From Smart Environments and Spaces to Green ICT*; River Publishers: Gistrup, Denmark, 2011.
4. Xia, F.; Yang, L.T.; Wang, L.; Vinel, A. Internet of things. *Int. J. Commun. Syst.* **2012**, *25*, 1101–1102. [CrossRef]
5. Ahmad, S.; Hussain, I.; Fayaz, M.; Kim, D.H. A Distributed Approach towards Improved Dissemination Protocol for Smooth Handover in MediaSense IoT Platform. *Processes* **2018**, *6*, 46. [CrossRef]
6. Guo, B.; Zhang, D.; Wang, Z.; Yu, Z.; Zhou, X. Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things. *J. Netw. Comput. Appl.* **2013**, *36*, 1531–1539. [CrossRef]
7. Leu, J.S.; Chen, C.F.; Hsu, K.C. Improving heterogeneous SOA-based IoT message stability by shortest processing time scheduling. *IEEE Trans. Serv. Comput.* **2014**, *7*, 575–585. [CrossRef]
8. Ding, Y.; Jin, Y.; Ren, L.; Hao, K. An intelligent self-organization scheme for the internet of things. *IEEE Comput. Intell. Mag.* **2013**, *8*, 41–53. [CrossRef]
9. Vlacheas, P.; Giaffreda, R.; Stavroulaki, V.; Kelaidonis, D.; Foteinos, V.; Poulios, G.; Demestichas, P.; Somov, A.; Biswas, A.R.; Moessner, K. Enabling smart cities through a cognitive management framework for the internet of things. *IEEE Commun. Mag.* **2013**, *51*, 102–111. [CrossRef]
10. Lazarescu, M.T. Design of a WSN platform for long-term environmental monitoring for IoT applications. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2013**, *3*, 45–54. [CrossRef]
11. Lima, D.F.; Amazonas, J.R. TCNet: Trellis Coded Network-Implementation of QoS-aware Routing Protocols in WSNs. *IEEE Latin Am. Trans.* **2013**, *11*, 969–974.
12. Aazam, M.; Khan, I.; Alsaffar, A.A.; Huh, E.N. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In Proceedings of the 2014 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 14–18 January 2014; pp. 414–419.
13. Skouby, K.E.; Lynggaard, P. Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services. In Proceedings of 2014 International Conference on the Contemporary Computing and Informatics (IC3I), Mysore, India, 27–29 November 2014; pp. 874–878.
14. Petrolo, R.; Loscri, V.; Mitton, N. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Trans. Emerg. Telecommun. Technol.* **2017**, *28*, e2931. [CrossRef]

15. Visvizi, A.; Lytras, M.D.; Damiani, E.; Mathkour, H. Policy making for smart cities: Innovation and social inclusive economic growth for sustainability. *J. Sci. Technol. Policy Manag.* **2018**, *9*, 126–133. [CrossRef]
16. Visvizi, A.; Lytras, M.D. Rescaling and refocusing smart cities research: From mega cities to smart villages. *J. Sci. Technol. Policy Manag.* **2018**, *9*, 134–145. [CrossRef]
17. Lytras, M.; Visvizi, A. Who uses smart city services and what to make of it: Toward interdisciplinary smart cities research. *Sustainability* **2018**, *10*, 1998. [CrossRef]
18. Chen, C.Y.; Hasan, M.; Mohan, S. Securing Real-Time Internet-of-Things. *arXiv* **2017**, arXiv:1705.08489.
19. Tönjes, R.; Barnaghi, P.; Ali, M.; Mileo, A.; Hauswirth, M.; Ganz, F.; Ganea, S.; Kjærgaard, B.; Kuemper, D.; Nechifor, S.; et al. Real time iot stream processing and large-scale data analytics for smart city applications. In *Poster Session, European Conference on Networks and Communications. sn*; City Pulse: Surrey, UK, 2014.
20. Kolozali, S.; Bermudez-Edo, M.; Puschmann, D.; Ganz, F.; Barnaghi, P. A knowledge-based approach for real-time iot data stream annotation and processing. In Proceedings of the 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), Taipei, Taiwan, 1–3 September 2014 ; pp. 215–222.
21. Davis, R.I.; Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.* **2011**, *43*, 35. [CrossRef]
22. Rostan, M.; Stubbs, J.E.; Dzilno, D. EtherCAT enabled advanced control architecture. In Proceedings of the IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC), San Francisco, CA, USA, 11–13 July 2010; pp. 39–44.
23. Jansen, D.; Buttner, H. Real-time Ethernet: The EtherCAT solution. *Comput. Control Eng.* **2004**, *15*, 16–21. [CrossRef]
24. Schneider, B.; Voss, S.; Wenger, M.; Zoitl, A. Using IEC 61499 Models for Automatic Network Configuration of Distributed Automation Systems. In Proceedings of the MATHMOD 2018 Extended Abstract Volume, 9th Vienna Conference on Mathematical Modelling, Vienna, Austria, 21–23 February 2018.
25. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
26. Shi, W.; Dustdar, S. The promise of edge computing. *Computer* **2016**, *49*, 78–81. [CrossRef]
27. Aliyu, S.O.; Chen, F.; He, Y.; Yang, H. A game-theoretic based qos-aware capacity management for real-time edgeiot applications. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 25–29 July 2017; pp. 386–397.
28. Nunna, S.; Kousaridas, A.; Ibrahim, M.; Dillinger, M.; Thuemmler, C.; Feussner, H.; Schneider, A. Enabling real-time context-aware collaboration through 5G and mobile edge computing. In Proceedings of the 2015 12th International Conference on Information Technology-New Generations (ITNG), Las Vegas, NV, USA, 13–15 April 2015, pp. 601–605.
29. Condoluci, M.; Araniti, G.; Mahmoodi, T.; Dohler, M. Enabling the IoT machine age with 5G: Machine-type multicast services for innovative real-time applications. *IEEE Access* **2016**, *4*, 5555–5569. [CrossRef]
30. Benítez, J.A.; Labra, J.E.; Quiroga, E.; Martín, V.; García, I.; Marqués-Sánchez, P.; Benavides, C. A Web-Based Tool for Automatic Data Collection, Curation, and Visualization of Complex Healthcare Survey Studies including Social Network Analysis. *Comput. Math. Methods Med.* **2017**, *2017*, 2579848. [CrossRef] [PubMed]
31. Lu, Y.H.; Lee, L.Y.; Chen, Y.L.; Cheng, H.I.; Tsai, W.T.; Kuo, C.C.; Chen, C.Y.; Huang, Y.B. Developing an App by Exploiting Web-Based Mobile Technology to Inspect Controlled Substances in Patient Care Units. *BioMed Res. Int.* **2017**, *2017*, 3195369. [CrossRef] [PubMed]
32. Côté, J.; Cossette, S.; Ramirez-Garcia, P.; Rouleau, G.; Auger, P.; Boudreau, F.; Gagnon, M.P. Improving Health and Reducing Comorbidity Associated with HIV: The Development of TAVIE en santé, a Web-Based Tailored Intervention to Support the Adoption of Health Promoting Behaviors among People Living with HIV. *BioMed Res. Int.* **2017**, *2017*, 4092304. [CrossRef] [PubMed]
33. Liu, Y.L.; Shih, C.T.; Chang, Y.J.; Chang, S.J.; Wu, J. Performance enhancement of a Web-based picture archiving and communication system using commercial off-the-shelf server clusters. *BioMed Res. Int.* **2014**, *2014*, 657417. [CrossRef] [PubMed]
34. Ahmad, S.; Malik, S.; Ullah, I.; Fayaz, M.; Park, D.H.; Kim, K.; Kim, D. An Adaptive Approach Based on Resource-Awareness Towards Power-Efficient Real-Time Periodic Task Modeling on Embedded IoT Devices. *Processes* **2018**, *6*, 90. [CrossRef]

35. Han, Y.; Elayoubi, S.E.; Galindo-Serrano, A.; Varma, V.S.; Messai, M. Periodic Radio Resource Allocation to Meet Latency and Reliability Requirements in 5G Networks. In Proceedings of the 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), Porto, Portugal, 3–6 June 2018; pp. 1–6.
36. Fernández-Cerero, D.; Fernández-Montes, A.; Jakóbik, A.; Kołodziej, J.; Toro, M. SCORE: Simulator for cloud optimization of resources and energy consumption. *Simul. Model. Pract. Theory* **2018**, *82*, 160–173. [CrossRef]
37. Audsley, N.C.; Burns, A.; Richardson, M.F.; Wellings, A.J. STRESS: A simulator for hard real-time systems. *Soft. Pract. Exp.* **1994**, *24*, 543–564. [CrossRef]
38. Sensini, F.; Buttazzo, G.; Ancilotti, P. Ghost: A tool for simulation and analysis of real-time scheduling algorithms. In Proceedings of the IEEE Real-Time Educational Workshop, Montreal, QC, Canada, June 1997; pp. 42–49.
39. Casile, A.; Buttazzo, G.; Lamastra, G.; Lipari, G. Simulation and tracing of hybrid task sets on distributed systems. In Proceedings of Fifth International Conference on the Real-Time Computing Systems and Applications, Hiroshima, Japan, 27–29 October 1998; pp. 249–256.
40. Harbour, M.G.; García, J.G.; Gutiérrez, J.P.; Moyano, J.D. Mast: Modeling and analysis suite for real time applications. In Proceedings of 13th Euromicro Conference on the Real-Time Systems, Delft, The Netherlands, 13–15 June 2001; pp. 125–134.
41. Löwe, W.; Liebrich, A. Vizzscheduler-a framework for the visualization of scheduling algorithms. In *Euro-Par 2001 Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 62–66.
42. Decotigny, D.; Puaut, I. ARTISST: An extensible and modular simulation tool for real-time systems. In Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Washington, DC, USA, 29 April–1 May 2002; pp. 365–372.
43. Urunuela, R.; Déplanche, A.M.; Trinquet, Y. Storm a simulation tool for real-time multiprocessor scheduling evaluation. In Proceedings of the 2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Bilbao, Spain, 13–16 September 2010; pp. 1–8.
44. Diaz, A.; Batista, R.; Castro, O. Realtss: A real-time scheduling simulator. In Proceedings of ICEEE 2007 4th International Conference on the Electrical and Electronics Engineering, Mexico City, Mexico, 5–7 September 2007; pp. 165–168.
45. Singhoff, F.; Legrand, J.; Nana, L.; Marcé, L. Cheddar: A flexible real time scheduling framework. In Proceedings of the ACM SIGAda Ada Letters, Atlanta, GA, USA, 14–18 November 2004; Volume 24, pp. 1–8.
46. Chéramy, M.; Hladik, P.E.; Déplanche, A.M. SimSo: A simulation tool to evaluate real-time multiprocessor scheduling algorithms. In Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Madrid, Spain, 8–11 July 2014; 6p.
47. Ahmad, S.; Malik, S.; Kim, D.H. Comparative Analysis of Simulation Tools with Visualization based on Real-time Task Scheduling Algorithms for IoT Embedded Applications. *Int. J. Grid Distrib. Comput.* **2018**, *11*, 1–10. [CrossRef]
48. Wan, J.; Chen, B.; Wang, S.; Xia, M.; Li, D.; Liu, C. Fog Computing for Energy-aware Load Balancing and Scheduling in Smart Factory. *IEEE Trans. Ind. Inform.* **2018**. [CrossRef]
49. Tao, F.; Cheng, J.; Cheng, Y.; Gu, S.; Zheng, T.; Yang, H. SDMSim: A manufacturing service supply–demand matching simulator under cloud environment. *Robot. Comput.-Integr. Manuf.* **2017**, *45*, 34–46. [CrossRef]
50. Rodrigo, G.P.; Elmroth, E.; Östberg, P.O.; Ramakrishnan, L. ScSF: A scheduling simulation framework. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Cham, Switzerland, 2017; pp. 152–173.
51. Ceballos, G.; Grass, T.; Hugo, A.; Black-Schaffer, D. Taskinsight: Understanding task schedules effects on memory and performance. In Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, Austin, TX, USA, 4–8 January 2017; pp. 11–20.
52. Wu, J.; Huang, Y.C. MCRTsim: A simulation tool for multi-core real-time systems. In Proceedings of the 2017 International Conference on Applied System Innovation (ICASI), Sapporo, Japan, 13–17 May 2017; pp. 461–464.
53. Myszkowski, P.B.; Laszczyk, M.; Nikulin, I.; Skowroński, M. iMOPSE: A library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem. *Soft Comput.* **2018**, 1–14. [CrossRef]
54. Kamel, E.; Memari, A.M. Automated Building Energy Modeling and Assessment Tool (ABEMAT). *Energy* **2018**, *147*, 15–24. [CrossRef]

55. Ahmad, S.; Hang, L.; Kim, D.H. Design and Implementation of Cloud-Centric Configuration Repository for DIY IoT Applications. *Sensors* **2018**, *18*, 474. [CrossRef] [PubMed]
56. Kim, S.C.; Jeong, Y.S.; Park, S.O. RFID-based indoor location tracking to ensure the safety of the elderly in smart home environments. *Pers. Ubiquit. Comput.* **2013**, *17*, 1699–1707. [CrossRef]
57. Fahim, M.; Fatima, I.; Lee, S.; Lee, Y.K. Daily life activity tracking application for smart homes using android smartphone. In Proceedings of the 2012 14th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea, 19–22 February 2012; pp. 241–245.
58. Sixsmith, A.; Johnson, N. A smart sensor to detect the falls of the elderly. *IEEE Pervas. Comput.* **2004**, *3*, 42–47. [CrossRef]
59. Dawadi, P.N.; Cook, D.J.; Schmitter-Edgecombe, M. Automated cognitive health assessment using smart home monitoring of complex tasks. *IEEE Trans. Syst. Man Cybern. Syst.* **2013**, *43*, 1302–1313. [CrossRef] [PubMed]
60. Zahariadis, T.; Papadakis, A.; Alvarez, F.; Gonzalez, J.; Lopez, F.; Facca, F.; Al-Hazmi, Y. FIWARE lab: Managing resources and services in a cloud federation supporting future internet applications. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC), London, UK, 8–11 December 2014; pp. 792–799.
61. Liu, C.L.; Layland, J.W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]
62. Joseph, M.; Pandya, P. Finding response times in a real-time system. *Comput. J.* **1986**, *29*, 390–395. [CrossRef]
63. Klein, M.; Ralya, T.; Pollak, B.; Obenza, R.; Harbour, M.G. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
64. van Berloo, R. GGT 2.0: Versatile software for visualization and analysis of genetic data. *J. Hered.* **2008**, *99*, 232–236. [CrossRef]
65. Huber, W.; Carey, V.J.; Gentleman, R.; Anders, S.; Carlson, M.; Carvalho, B.S.; Bravo, H.C.; Davis, S.; Gatto, L.; Girke, T.; et al. Orchestrating high-throughput genomic analysis with Bioconductor. *Nat. Methods* **2015**, *12*, 115–121. [CrossRef]
66. Romanyk, C.; McCallum, R.; Salehi, P. A Model Based Approach to Web Application Design for Older Adults Using MVC Design Pattern. In *International Conference on HCI in Business, Government and Organizations*; Springer: Cham, Switzerland, 2016; pp. 348–357.
67. Flask. Web Development One Drop at a Time. 2010. Available online: http://flask.pocoo.org/ (accessed on 5 January 2018).
68. Bootstrap. 2010. Available online: https://getbootstrap.com/ (accessed on 5 January 2018).
69. Cooking Hacks. e-Health Sensor Platform. 2015. Available online: https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical (accessed on 15 December 2018).
70. Mininet Walkthrough. 2014. Available online: http://mininet.org/walkthrough/ (accessed on 11 November 2018).
71. ONOS Tutorial with Mininet. 2014. Available online: http://www.stackguy.com/archives/248 (accessed on 11 November 2018).
72. Lenz, J.E. MAST: A simulation tool for designing computerized metalworking factories. *Simulation* **1983**, *40*, 51–58. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).