Time and Cost Efficient Cloud Resource Allocation for Real-Time Data-Intensive Smart Systems

Muhammad Shuaib Qureshi [1,2] , **Muhammad Bilal Qureshi** [3,] *, **Muhammad Fayaz** [2] , **Muhammad Zakarya** [4] , **Sheraz Aslam** [5] **and Asadullah Shah** [1]

[1] KICT, International Islamic University, Kuala Lumpur 50728, Malaysia; muhammad.qureshi@ucentralasia.org (M.S.Q.); asadullah@iium.edu.my (A.S.)

[2] Department of Computer Science, School of Arts and Sciences, University of Central Asia, 310 Lenin Street, Naryn 722918, Kyrgyzstan; muhammad.fayaz@ucentralasia.org

[3] Department of Computer Science, Shaheed Zulfikar Ali Bhutto Institute of Science and Technology, Islamabad 44000, Pakistan

[4] Department of Computer Science, Abdul Wali Khan University, Mardan 23200, Pakistan; mohd.zakarya@awkum.edu.pk

[5] Department of Electrical Engineering, Computer Engineering and Informatics, Cyprus University of Technology, Limassol 3036, Cyprus; sheraz.aslam@cut.ac.cy

* Correspondence: muhdbilal.qureshi@gmail.com

Abstract: Cloud computing is the de facto platform for deploying resource- and data-intensive real-time applications due to the collaboration of large scale resources operating in cross-administrative domains. For example, real-time systems are generated by smart devices (e.g., sensors in smart homes that monitor surroundings in real-time, security cameras that produce video streams in real-time, cloud gaming, social media streams, etc.). Such low-end devices form a microgrid which has low computational and storage capacity and hence offload data unto the cloud for processing. Cloud computing still lacks mature time-oriented scheduling and resource allocation strategies which thoroughly deliberate stringent *QoS*. Traditional approaches are sufficient only when applications have real-time and data constraints, and cloud storage resources are located with computational resources where the data are locally available for task execution. Such approaches mainly focus on resource provision and latency, and are prone to missing deadlines during tasks execution due to the urgency of the tasks and limited user budget constraints. The timing and data requirements exacerbate the efficient task scheduling and resource allocation problems. To cope with the aforementioned gaps, we propose a time- and cost-efficient resource allocation strategy for smart systems that periodically offload computational and data-intensive load to the cloud. The proposed strategy minimizes the data files transfer overhead to computing resources by selecting appropriate pairs of computing and storage resources. The celebrated results show the effectiveness of the proposed technique in terms of resource selection and tasks processing within time and budget constraints when compared with the other counterparts.

Keywords: data-intensive smart application; cloud computing; resource allocation; real-time systems; smart grid

---

1. Introduction

With fast-growing advancements in smart systems, the real-time applications are handy candidates for utilizing the computing power in a cloud computing environment in order to maintain deadline constraints. Cloud computing is an economic-based paradigm consisting of distributed resources and providing services by collaborating in executing user applications. The cloud services are categorized into Infrastructure-as-a-Service (IaaS) that deals with providing computing such as VMs and storage resources as services, Platform-as-a-Service (PaaS) that offer deployment and development platforms as services, and Software-as-a-Service (SaaS) that facilitate users with web-based applications. The most common examples are IBM's Blue Cloud (for IaaS), Google AppEngine and Microsoft Azure (for PaaS), and EC2 (for SaaS). These services are employed by using cloud deployment models, namely public, private, and hybrid established on the basis of organization preferences. The cloud compute and storage resources are selected and allocated on the basis of nature of user application. In addition, the cloud storage resources provide facilities such as accommodating data replication to satisfy data requirements of the data-intensive real-time systems that need to access, process and transfer data files stored in distributed data repositories [1,2]. Examples of such applications are self-driving vehicles, which depend on the data and computations under a complex network of interconnected devices such as GPS, surveillance cameras, radar, laser light, odometry, etc., to perceive the surroundings [3,4]. The cloud providers such as Amazon EC2 [5] provide computing facilities (virtual machines) on a pay-as-you-go basis at the rate of 10 cents per hour. The lease prices vary depending on the virtual machines (VMs) specifications. The normal VM offers an approximate processing power of 1.2 GHz Opteron processor with a storage capacity of 160 GB disk space and 1.7 GB of memory [5–7]. Such facilities pave the way for executing time-critical and IoT applications which demand high processing and storage capabilities [8]. Stergioua et al. [9] merged cloud computing with IoT to improve the functionality of the IoT. The IoT devices offload many tasks to the cloud environment from the smart systems because these devices have very limited processing and storage capabilities. Leveraging the capabilities of virtualization technology, VMs can be scaled up and down depending on the current system workloads [10]. For executing VM code on smart platforms, smart virtual machines are proposed by Lee et al. [11]. However, there is a lack of efficient resource scheduling and allocation strategies for deploying real-time applications with stringent QoS and data requirements in cloud computing environments. The scheduling policy for data-intensive real-time systems proposed in [12] allocates HPC resources by considering that single copies of data-files are available on storage resources without taking into account the computation and transfer cost. The concept of real time scheduling is used in the deployment of smart environments [13–17].

The Global Institute [18] report on analyzing the economic impact of the IoT devices show that it will increase upto $11 trillion by the year 2025. This increase is because the IoT devices and smart home appliances ranging from small sensors to large scale biometrics offload data and computation to the cloud computing environment on a regular basis [19,20]. For this purpose, the IT companies provide solutions such as Apple's HomeKit [21], Samsung's SmartThings [22], Amazon's Alexa [5], and Google's Home [23], etc. The smart systems are considered as data-intensive systems that are different from compute-intensive or eScience applications because of the storage, access, execution and management requirements of distributed datasets and hence, require different scheduling and allocation policies. These systems basically deal with the data and transport layers for replication and access of datasets. The data-intensive smart systems can be considered as a combination of data producers and consumers geographically distributed across multiple organizations. The producers are the entities that produce data and manage its distribution over multiple locations. The consumers can be the users or their applications which need this data produced by the producers for multiple purposes. The consumers investigate efficient ways out of many to access the data for executing applications on remote compute nodes.

The driving force of cloud computing is the virtual machine manager (VMM) that creates the virtual resources of the physical machines. The basic functionality of the VMM is to separate the virtual computing environment from the underlying physical infrastructure. In this research work, we implement the rate monotonic (RM) scheduling policy to allocate cloud computing resources for the real-time data-intensive periodic tasks. The scheduling problem is divided into three parts: the processing environment (cloud virtual machines), the nature of the real-time task (fixed priority system), and the optimization criteria (time- and cost-efficient allocation). The real-time task set is a collection of multiple tasks, each of which requires data for processing. The required data files are requested from the remotely located storage resources. The intelligent selection and assignment of cloud computing resources are investigated while the data files are replicated on decoupled storage resources and accessed by utilizing networks of varying capabilities. The proposed strategy evaluates all the storage locations for the replicas of the same data file and selects the one which has minimum data access and transfer cost. It submits the application to the computing resource that is closest to the selected storage location and can complete execution within minimum possible execution time. These files are fetched to the computing resources where tasks are executed, which add transfer time to the red tasks' total execution time. In our proposed model, a user-specified budget is associated with each smart system request, and hence resources are selected not only on the basis of their high computational power, but also the cost associated with each and the storage resource are computed, as well as the ability to process jobs within tasks deadlines and scheduling preferences.

The major research contributions of our work are:

- Creating a model for selecting appropriate cloud computing and storage resources to execute real-time data-intensive systems generated by smart devices where data is replicated on multiple storage resources,
- Partitioning the task sets into groups based on common data-file demands such that the timing constraints of the original tasks set are not disturbed,
- Analysing the economic perspective of data storage and processing by scheduling real-time smart systems on distributed nodes with different storage, execution, and data transfer costs and allocating heterogeneous cloud resources,
- Allocating cloud computing resources to periodic real-time tasks such that the overall timing constraints of the smart devices remain intact,
- Analysing cloud computing and IoT usability in the context of data-intensive smart systems.

Cloud computing is considered a promising platform for executing large-scale computing-intensive IoT and smart grid applications in a cost-efficient way due to the large pool of computing resources. These resources need intelligent scheduling and allocation of the tasks such that all tasks in a batch can be processed within the stipulated time span. The research community focuses on searching mechanisms for scheduling and allocating distributed cloud resources in a systematic way which can satisfy formulated objective function such as load balancing, makespan minimization, and cost-efficiency with respect to the user-defined QoS criteria [3,24,25]. But, instead of the vast study in the cloud resource allocation domain, the existing literature is not mature in providing suitable scheduling mechanisms for real-time systems generated by smart devices due to the high deadline-miss ratio by a number of tasks in a batch [26]. The problem becomes more challenging when such systems need data from external sources. A real-time system is characterized by the deadlines of the tasks. In such systems, deadline meeting is primarily important for utilizing maximum capabilities of the cloud resources, since most of the real-time systems such as sensors in smart systems or actuators in automated systems generate periodic tasks, which are sent to the processing units after regular intervals. Such systems need proper priority-based and non priority-based strategies for scheduling. In priority-based scheduling policy, the scheduling

criteria is saved for the entire duration of task execution, while in non priority-based systems, the tasks have no precedence constraints and the scheduling criteria may change with the arrival of the next job of a task. The main advantage of priority-based policies is time-saving and its simple implementation. The priority-based scheduling is also known as static priority scheduling. The well-known static priority assignment algorithm is Rate Monotonic developed by Liu and Layland in 1973. The main features of this algorithm are its simple implementation at OS kernel level, predictability in real-time behavior like in smart systems, and its easy modification for implementing task priority inheritance protocol for the purpose of synchronization [27].

In order to utilize the full potential of the cloud and IoT platforms, Suciu et al. [28] proposed a conceptual framework for the deployment of IoT based smart microgrid applications. The developed framework has the capability of integrating real-time data generated by the ubiquitous sensing devices constituting the smart home with the cloud computing environment, but their system is prone to missing deadlines because they have not evaluated their system on each time instant according to the workload of the task. The dynamic algorithm for scheduling soft real-time systems on grid resources was proposed in [29]. The proposed algorithm provides room for tasks with missed deadlines. Ye et al. [30] proposed architecture of smart home-oriented cloud. They have suggested a layered cloud design that provides efficient services for digital appliances. The authors have considered the sensors sending data continuously without specifying any real-time constraints. Caron et al. [1] consider task priorities for scheduling real-time tasks. The tasks are checked one by one. Shang et al. [1] formulated a model which elaborates grid service reliability assessment for dependable and cost-efficient applications. They have derived a cost function based on genetic and particle swarm optimization techniques that calculate service expense of each utilized resource. Isard et al. [4] considered scheduling problem of data-intensive tasks using the Hadoop structure. They have supposed coupled computing and data resources located at the same place and that the data is available for each task without transferring from remotely located resources. Therefore, they have not included the data transfer time in the task feasibility analysis. The proposed tasks are preemptable but no specific criterion is discussed for which task is to be preempted when an interruption occurs or when high priority tasks arrive. In ref. [5], the authors have focused on submitting preemptable tasks to the federated grid. They have developed a schedule which maximizes the acceptance of incoming tasks, and minimizes user-defined QoS criteria violation. The authors have not emphasized the heterogeneity of resources. Poola et al. [6] presented a mechanism for robust and fault-tolerant scheduling of scientific workflows on heterogeneous resources which concurrently optimizes makespan and execution cost. Ma et al. [7] used a hybrid approach by combining the best features of genetic and greedy approaches for QoS-aware web service composition. They have focused on minimizing cost, but they have considered non-real-time tasks. A cost optimization technique for executing data-intensive tasks on distributed resources was developed by Mansouri et al. [8]. Leveraging data storage and migration cost is addressed by using an optimal online algorithm, but their system is not suitable for executing real-time smart systems tasks. The proposed algorithm provides tasks scheduling and cloud resource allocation criteria for real-time tasks generated by smart systems considering resources heterogeneity, availability, data-intensive and timing constraints of the tasks.

The rest of the paper is structured as follows. In Section 2, we throw light on discussing task, resource, and cost model. The proposed time- and cost-efficient scheduling algorithm is explained in Section 3, while the performance of the proposed resource allocation strategy and details of the experimental setup is evaluated in Section 4. The produced results are presented in Section 5 and conclusions and future directions are provided in Section 6.

2. Task, Resource and Cost Models

In this research, we consider scheduling feasibility of real-time periodic tasks in a cloud environment. Our model as shown in Figure 1 is composed of smart devices which generate periodic tasks $(\tau_1, \tau_2, \ldots, \tau_n)$. The tasks represent data-intensive applications that generate data on a regular basis and need computing resources for processing. The represented smart devices have limited memory, storage and processing capabilities. Each task needs data stored on some remote storage locations. The tasks constitute a task set $T_i(i = 1, 2, \ldots, n)$ where the collection of task sets form a smart microgrid that sends data to a main smart hub known as Smart Grid Management System. The smart hub manages the received data and uploads the collected tasks to the cloud environment. The Cloud Resource Management System (Cloud RMS) handles task requests and manages cloud resources. The Cloud RMS has the responsibility to search suitable resources and required data files information from the Resource Files Information Directory (RFID) according to the task requirements and schedule tasks on cloud resources. The Cloud RMS also implements the scheduling policy. Our concerned cloud environment is comprised of both computational $(CR_1, CR_2, \ldots, CR_n)$ and data storage resources $(DR_1, DR_2, \ldots, DR_n)$ located remotely and connected by network links of different bandwidths. The resources are heterogeneous and characterized by power and cost constraints.

In this paper, we concentrate on two basic constraints; (a) the real-time tasks' deadlines, and (b) user-specified budget. The presented model extends the RDTA model [12] by introducing cost parameters, data files replication scenarios, and tasks' grouping criteria.

2.1. Task and Resource Model

We consider batch processing of real-time periodic tasks, each of which can generate an infinite number of jobs. In periodic tasks set $T = \{task_1, task_2, \ldots, task_n\}$, each $task_k$ is defined by the quadruple:

$$task_k = (r_k, e_k, d_k, period_k) \tag{1}$$

where r_k shows the release time of the first job, e_k the required computation time, d_k the relative deadline of $task_k$ which is the time difference between the absolute deadline and release time of a job, and $period_k$ the period which is the time difference between the two successive jobs of a $task_k$.

In the above discussed model, a job i released at time instant $r_k + (i-1).period_k$ needs to execute for e_k units before the time $r_k + (i-1).period_k + d_k$. In our task model, we concentrate on a constrained deadline model which assumes that $d_k \leq period_k, \forall k \in T$. Tasks preemption is not allowed and context switching overhead is subsumed into e_k. We also assume that $r_k = 0, \forall k \in T$, which means that feasibility of the tasks is checked when the system is most loaded.

We consider computing resource set CR such that $CR = \{CR_1, CR_2, \ldots, CR_r\}$. Each one is characterized by a computing power $CP_y(1 \leq y \leq r)$ such that $CP_y \in CP$, where $CP = \{CP_1, CP_2, \ldots, CP_r\}$, and measured in Millions of Instructions per Second (MIPS). The execution time of a $task_k$ on resource CR_y can be computed by

$$EET_{ky} = (e_k + e_{higher})/CP_y, \tag{2}$$

where e_{higher} is the execution time of higher priority tasks than $task_k$. Mathematically,

$$e_{higher} = \sum_{j=1}^{k-1} \lceil \frac{t}{period_j} \rceil e_j, \quad t \in PNP_j \tag{3}$$

where PNP set accumulates points or time instants on which task feasibility is analyzed. The PNP set is defined as follows.

Definition 1. *PNP_k is a set of positive and negative points for $task_k$ constituted by the relation $x.period_j$ such that $1 \leq j \leq k$ and $1 \leq x \leq \lfloor period_k / period_j \rfloor$, where $period_j$ represents periods of higher priority tasks than $task_k$. The point $t_P \in PNP_k$ is said to be positive if $task_k$ is declared feasible (i.e., completes its execution at or before the deadline) at some point t by considering all associated time and data constraints. The point $t_N \in PNP_k$ declares the $task_k$ infeasible when it misses the deadline. Each point in PNP_k is called rate-monotonic scheduling point. From Definition (1), it is concluded that the set PNP is the union of positive points set PP and negative points set NP where $PP = PNP - NP$ and $NP = PNP - PP$. In other words, $PNP = PP \cup NP$.*

2.2. Data Files Model

In our task model, a tasks set $T = \{task_1, task_2, \ldots, task_n\}$ consists of data-intensive real-time tasks where each task $task_k$ needs a set of data files DF_k for its execution. The set $DF_k = \{f_k1, f_k2, \ldots, f_km\} \subseteq DF$. The file $f_k x \in DF_k$ is stored on data storage resource dr_w, where $dr_w \in DR_k$ and $DR_k \subseteq DR$. The DR is the set of total storage resources in the HPC environment. In other words, files in DF_k are stored on DR_k storage resources. We assume that the data files are replicated on more than one data storage resources.
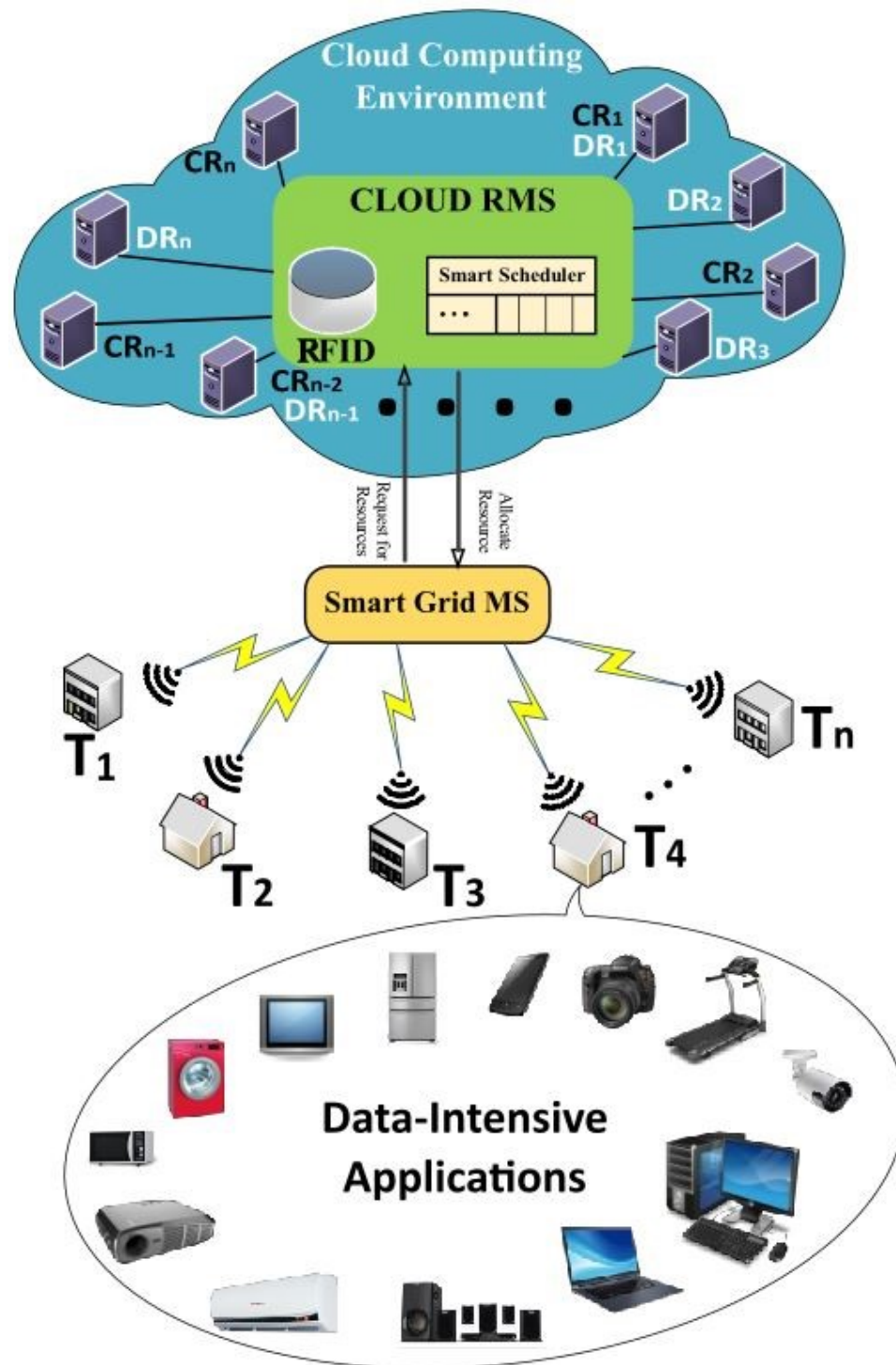
Figure 1. Model for offloading computation and data-intensive application from smart microgrid to the cloud.

The total execution time TT of a task $task_k$ is the sum of actual computation time EET of $task_k$ on computing resources CR_y and the transfer time taken by the required m data files in the set DF_k

by transferring from storage resources DR_k to the computing resource CR_k where task k is executed. Mathematically,

$$TT_k = EET_{ky} + \sum_{z=1}^{m} FT_{(f_k z)} \tag{4}$$

where $FT_{f_k z} = R_w + Size_{f_k z}/BW_{wy}$ is the transfer time of the file $f_k z$.

The R_w represents the response time of the data storage resource $dr_w \in DR_k$ where the data file $f_k z$ is stored. The response time is the time when the request to fetch the file is made at the time when the request is entertained. Algebraically, the response time of data resource dr_w is calculated as:

$$R_w = ST_{f_k z} + WT_{f_k z} \tag{5}$$

where $ST_{f_k z}$ is the service time and $WT_{f_k z}$ is the waiting time of the request respectively for accessing the file $f_k z$. Also, the $Size_{f_k z}$ denotes the size of the file $f_k z$, and BW_{wy} shows the link bandwidth between data storage resource dr_w and computing resource CR_y. The proposed model selects that storage resource for file access for which FT is minimum, i.e., $\min(FT)$.

2.3. Task Grouping

The data-intensive real-time tasks in T are grouped into x number of groups on the basis of common data files demands. The tasks in a group represent a subset of T or we can say each group is a set of tasks for easy understanding. The task grouping taxonomy is pictorially represented in Figure 2.
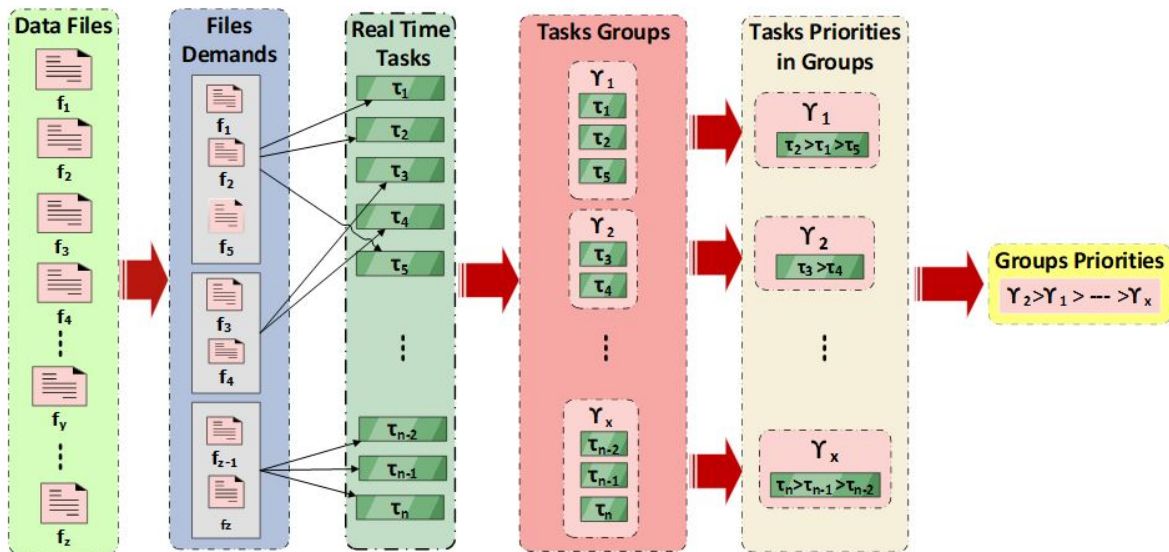

Figure 2. Task grouping taxonomy.

Based on real-time task grouping criteria, the group of tasks, its cardinality, and priority assignment is defined in the following sections.

Definition 2. *A group of real-time tasks Y_x is a subset of tasks, i.e., $Y_x \subseteq T(x \leq n)$ having common data files demands. Each group Y_x contains minimum one task which concludes that $Y_x \neq \{\}$.*

Definition 3. *The cardinality of a task group defines the total number of tasks in a group. Let there be total x number of groups, then cardinality of the original tasks set $T = \{task_1, task_2, \ldots, task_n\}$ can be defined as:*

$$card(T) = \sum_{l=1}^{x} card(Y_1) \tag{6}$$

The advantage of the task grouping mechanism is to reduce the total number of priority levels [25]. Additionally, $t_N \in NP$ for a higher priority task $task_j \in Y_x$ remains the member of NP for all lower priority tasks than $task_j$ in Y_x, since tasks in the same group are also sorted on the basis of RM priorities. In this way, the least number of points is tested on the PNP set which decreases the execution time.

From the above definitions, the following can be observed.

Observation 1. *Let each task group Y_l constituted from task set $T = \{task_1, task_2, \ldots, task_n\}$ contains a single task in a worst case scenario and x represents the total number of groups in the system, then $x = n$ where n represents $card(T)$.*

Each real-time task in our task model has deadline d, which demonstrates the maximum allowed time for a task to complete its execution on a single computing resource. Let the maximum and minimum deadlines of the tasks in a group Y_x are denoted by d_{max} and d_{min} respectively. Here an interesting observation can be made.

Observation 2. *d_{max} of the tasks $\{task_1, \ldots, task_l\} \in Y_x (l \leq n)$ sorted by RM priority assignment technique and following the implicit deadline model (period = deadline) is the period of the last task while d_{min} is the period of the first task in Y_x. The relation follows:*

$$d_{max} = period_l d_{min} = period_1 \tag{7}$$

where $period_l$ and $period_1$ represent periods of the last and the first task in Y_x, respectively.

Proof. The RM technique sort the tasks based on priority assignment criteria: the lesser the period of the task, the higher the priority. This means that the last task $task_l \in Y_x$ has the lowest priority and first task $task_1 \in Y_x$ has the highest priority among all tasks in Y_x. The $task_1$ is executed in the first and $task_l$ is executed in the last slot. In other words, $priority(task_1) \geq priority(task_2) \geq, \ldots, \geq priority(task_l)$ which follows that $period(task_1) \leq period(task_2) \leq, \ldots, \leq period(task_l)$. It also follows that $d_{max} = period_l$ and $d_{min} = period_1$, which completes the proof.
□

The authors in [24] discussed that RM assigns static priorities to tasks and considered an optimal scheduling algorithm among static priority assignment scheduling algorithms. By optimal they mean that RM should schedule a task, if any other static priority assignment algorithm can schedule that task. Following are the general characteristics of the RM scheduling technique which play a role in proving its optimality.

1. the system should consist of a fixed number of tasks;
2. the tasks should have execution times known in advance;
3. each task has a completion deadline equal to its period;
4. tasks should be periodic which means that instances or jobs of a task should arrive after a fixed time interval;

5. tasks should be independent;
6. all tasks should arrive at time 0. This time instant is also known as the critical instant and the system is considered as the most overloaded at this instant.

Definition 4. *The period of a task group is defined as the temporary period attached to the group of tasks which is the period of the last task in a group. In other words, $period_l$ is the group period because the tasks in a group are sorted using RM technique. For example, if a group Y_x accommodates tasks $\{task_1, task_2 \ldots, task_l\}$, then*

$$period(Y_x) = period_l. \tag{8}$$

Since the groups are sorted on the basis of RM priorities, so $period(Y_1) \leq period(Y_2) \leq, \ldots, \leq period(Y_l)$ which states that $priority(Y_1) \geq priority(Y_2) \geq, \ldots, \geq priority(Y_l)$. Equation 8 states that all tasks in Y_x must complete execution at or before $period_l$. It has been further evaluated that since the tasks in the same group Y_x are also sorted by RM priorities, so $t_N \in NP$ for a higher priority task $task_i \in Y_x$ remains the member of NP for all the lower priority tasks than $task_i$.

Definition 5. *The group capacity can be defined as the total number of tasks in a group. Tasks in a group are added on the basis of common data files demand. The group capacity can be analyzed on the basis of resource utilization by a group of tasks called group utilization (GU) which is defined as the sum of the resource utilization of each task in the group. The computing resource utilization of each task is termed as task utilization (TU). Let n denotes the total number of tasks in a group Y_x, then GU and TU can be found as follows.*

$$GU_{Y_x} = TU_1 + TU_2 + \ldots + TU_n = \sum_{i=1}^{n} TU_i \tag{9}$$

where

$$TU_i = \frac{e_i}{period_i} \tag{10}$$

Theorem 1. *Let Y_x be a group of n periodic tasks, where each task is characterized by TU. The group Y_x is said to be RM feasible if the following condition holds:*

$$GU_{Y_x} \leq n(2^{1/n} - 1) \tag{11}$$

The inequality (11) is called the Liu and Layland (LL) test reported in [24]. The expression $n(2^{1/n} - 1)$ is the threshold value of a group which means that a group Y_x can accommodate tasks as for as the GU remains lower than or equal to the threshold value. Equation (11) is checked repeatedly when a new task is added to the group. If adding a task changes the inequality to $GU \geq n(2^{1/n} - 1)$, then the incoming tasks are added to another group. The authors in [25] refer the LL test as the sufficient condition test. They claim that it is not necessary that the tasks in a group are not schedulable if Equation (11) does not hold. This means that utilization-based tests are sufficient, but not necessary. Let us explain by the following example task set taken from [26,29].

Example 1. *Consider a task group $Y = \{task_1, task_2, task_3, task_4\}$ where tasks follow RM ordering and having following characteristics and utilizations mentioned in Table 1 and Table 2 respectively.*

Table 1. Task characteristics in Y.

Tasks	C_i	$period_i$
Tak_1	2	3
Tak_2	1.5	6
Tak_3	0.5	12
Tak_4	1	24

Table 2. Task utilizations.

TU	Value
TU_1	0.666
TU_2	0.250
TU_3	0.041
TU_4	0.041

For the above tasks set Y, the $GU_Y \cong 1$ and threshold = 0.756.

It shows that the aforementioned example task group Y is not schedulable by using LL test because it does not satisfy Equation (11). However, the gantt chart in Figure 3 shows the schedulability of these tasks within deadlines under RM technique. From the above discussion, it is clear that LL test is sufficient only, so we use LL test for checking group capacity only. For analyzing task or group feasibility, we use PNP test which is necessary and sufficient conditions test.
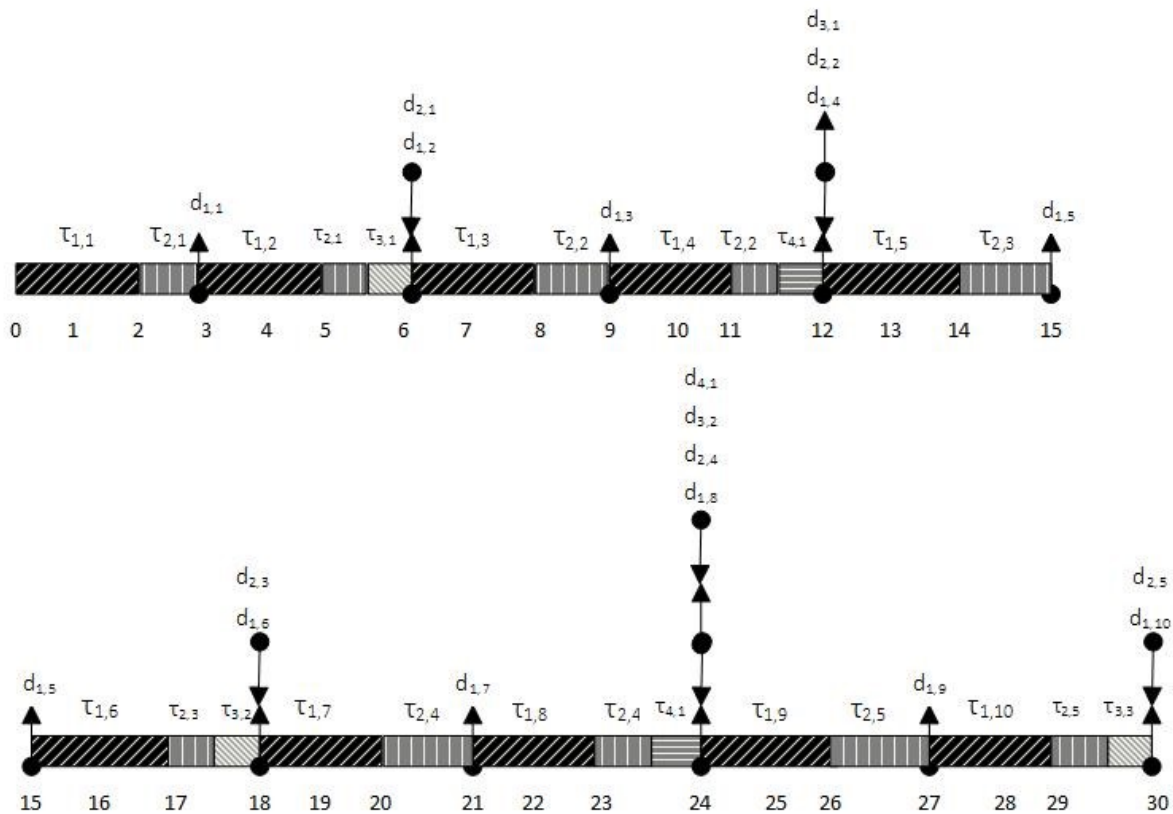

Figure 3. Gantt chart for scheduling tasks set in Example 1.

Theorem 2. *A group of real-time tasks* $Y_x = \{task_1, task_2, \ldots, task_l\}$ *is schedulable if all tasks in* Y_x *are schedulable.*

Theorem 3. *The batch of real-time tasks called periodic tasks set represented by* $T = \{task_1, task_2, \ldots, task_n\}$ *is deemed feasible if all task groups* $Y_1, Y_2, \ldots, Y_x$ *are schedulable.*

2.4. Cost Model

Scheduling decisions by integrating cost parameters change the way computational resources are selected to fulfill the user *QoS* criteria. The data-intensive real-time tasks are submitted to the broker which searches resources to process tasks within deadlines and user-specified budget constraints. The feasibility of tasks' groups on computational resources is checked by considering data transfer time, transfer costs, computational cost, tasks' deadlines, and computational power of the resources. The basic parameters considered for feasibility decisions in this research are:

(a) user-specified budget, and
(b) tasks deadlines.

Based on the above two parameters, the cloud computing and storage resources which can execute tasks within deadlines in a minimum cost by taking into account all data and processing constraints are selected.

By introducing cost model, Theorem 3 can be extended in Theorem 4 for checking schedulability of modified task set.

Theorem 4. *The batch of real-time tasks called periodic tasks set represented by* $T = \{task_1, task_2, \ldots, task_n\}$ *is deemed feasible with minimum cost if all task groups* $Y_1, Y_2, \ldots, Y_x$ *are schedulable by following all tasks constraints and holding inequality (12).*

$$cost_T \leq Budget \tag{12}$$

where $cost_T$ *is the total cost incurred by the batch of tasks, and Budget is the total user-specified budget. The cost of a resource can be expressed as execution cost per Millions of Instructions (MI), processing cost per unit time, processing cost per task, or simulation cost per unit time, etc. The cost for a single task is the sum of task execution cost and the data files transfer cost.*

3. Time- and Cost-Efficient Scheduling Algorithm

The Algorithm 1 determines the schedulability of real-time independent tasks set consisting of tasks with different data files and timing constraints. The execution procedure of the tasks involves checking task group feasibility which cumulatively constitutes tasks set. The *m* number of tasks in a group are checked on *r* number of distributed computing resources where $r \gg m$. Depending on the user budget and tasks scheduling preferences, the main objective of this algorithm is to execute distributed data-oriented applications by selecting computing and storage resources such that the tasks are processed with minimum total execution time and cost while tasks' deadlines are respected. The proposed algorithm works in three parts: (a) task initial feasibility checking which predicts a task's basic feasibility within a deadline by searching initial feasible computing resources, (b) task final feasibility and cost analysis which determines a task's schedulability after considering all the associated constraints, and finally (c) task dispatching to the best suitable resources after fulfilling all the pre-requisites. The first two parts are the matching and mapping parts which create set of time- and cost-efficient computing–storage resources pairs. The third part is the dispatching part which ensures that the selected resources can process tasks within time and budget constraints. By cost we mean the sum of a task's execution cost and data-files transfer cost. Similarly,

the total execution time to be minimized is the sum of tasks actual execution time and the transfer time incurred by transferring data-files from the storage resources to the computing resources where the task is executed. The data-files are replicated on multiple storage resources and the resource which has the minimum transfer cost is selected for data-file transfer. The computing resource capability for executing a task is checked by analyzing task feasibility on PNP points. As a result, the computing resource that can execute a task by maintaining the deadlines is initially selected from the list of available computing resources. The selected resource is called an initially feasible resource. The service requests are provisioned according to the described scheduling strategy; i.e., the total execution time of the task set and the incurred cost should be minimized. To ensure the fulfilment of the aforementioned two objectives, the set of storage resources are demonstrated which accommodate data-files needed for the task $task_i$ after identifying the initially feasible resources. A single file is assumed to be replicated on more than one storage resource, so the resource which has less transfer time and cost is selected. All such computing–storage resources pairs are further checked for calculating total execution time. The total execution time is the sum of all time factors. If the total time is within the task $task_i$ deadline and the total cost is within the user-specified budget, the compute–storage resources pair is declared feasible for assigning $task_i$. After selecting all such pairs for all tasks in a group, the tasks are then dispatched to the qualified resources by the dispatcher and all required files are transferred. The tasks are scheduled and computations are carried out. In this way, if all tasks in a group Y_x are scheduled, then the group Y_x is said schedulable by the Algorithm 1. Furthermore, if all groups are scheduled, then the original task set T is declared schedulable with minimum time and cost. The resource allocation procedure completes when all the tasks are dispatched to the resources and the unmapped queue becomes empty. The pseudocode of the tasks mapping and dispatching procedures is given in Algorithm 1.

The purpose of Algorithm 2 is to find suitable compute–storage resource pairs for each data-file required by a task. For each task, sets of required data-files and initially feasible computing resources are passed from Algorithm 1 as input arguments to the file transfer time calculating function. For each data file, the storage resources are identified and the best data storage resource which qualifies the minimization criteria (transfer time and cost) are selected for retrieving data file. For each data file, all possible combinations of initially feasible compute–storage resource pairs are tried and finally, the right combination is returned with decreased transfer time and execution cost.

Algorithm 1: Time- and cost-efficient assignment of real-time data-intensive group of tasks to the HPC resources

---

1 **Input:** Computing resources sorted in descending order of processing capacities, and a group Y_x of unmapped real-time tasks ordered by RM priorities and having budget constraints.;

2 **Output:** Time- and cost-efficient real-time data-intensive tasks schedule on HPC resources.
  Procedure

3 **for** *all* $task_i \in Y_x$ **do**

4   compute $PNP_i = \{x.period_l | l = 1, \ldots, i; x = 1, \ldots, \lfloor period_i / period_l \rfloor\}$;

5   // **Determining task initial feasibility for** *all available computing resources* $CR_r \in CR$ **do**

6   **for** *all* $t \in PNP_i$ **do**

7   calculate EET_{ir};   //gives minimum EET because resources are already sorted

8   **if** $EET_{ir} \leq t$ **then**

9           $CR_i \leftarrow CR_r$;   // CR_i is set of comp resources on which $task_i$ is initially feasible

10   **Break;**   //break if t_p is found

11   End

12   End

13   End

14   **if** DF_i *do not locally exist* **then**

15       $DF_i \leftarrow FT(CR_i, DF_i)$;   //Call to Algorithm 2. CD_i is comp-storage resource pairs set for which DF_i has min transfer time and cost

16   End

17   calculate TT_{ir};   //TT on CD_i

18   // **Determining the task final schedulability and cost analysis**

19   **if** $TT_{ir} \leq t$ & & $cost_i \leq Budget$ **then**

20   mark CD_i feasible for $task_i$;

21   End

22   End

23 // **Dispatching tasks to the feasible computing resources**;

24 **for** *schedulable tasks* $task_i$ **do**

25   submit $task_i$ to CR_r;

26   transfer all required files to CR_r;

27   update resource information directory;

28   remove $task_i$ from unmapped tasks list;

29   End

30 initialize computing resources to maximum processing powers and update resource information directory;

31 **End Procedure**

Algorithm 2: Selection of time- and cost-efficient compute–storage resource pairs for each data-file

---

1 Specify DR_i; //storage resources on which files DF_i are stored;

2 **for** *all $f_x \in DF_i$* **do**

3   **for** *all $CR_z \in CR_i$* **do**

4   //compute resource z on which $task_i$ is initially feasible;

5   **for** *all $dr_j \in DR_i$* **do**

6              $C_{zj} \leftarrow (FT_{zj}, cost_x)$ //transfer time and cost pair for file f_x in matrix C;

7   End

8   End

9      $A_x \leftarrow (zdr, min(C))$ /pair of $comp - storage$ resources for which transfer time and cost for f_x is min;

10   End

11 Return (A); //return comp-storage resources vector on which transfer time and cost for f_x is min;

---

4. Performance Evaluation

This section discusses the experimental set-up, the input data, and performance metrics used to evaluate the proposed resource allocation technique.

4.1. Experimental Setup

The proposed RA technique and the existing counterparts were simulated using synthetic data sets. These experiments were carried out in MATLAB R2016a on Intel Core i5 processor, 2.50 GHz CPU and 8 GB RAM running on Microsoft Windows 10 platform. The reason for using MATLAB is that it provides a multiprocessing environment for solving complex mathematical problems demanding powerful computations. The HPC systems are difficult to implement practically due to the lack of real life experimentation environment and multiple domain administration problems which make it difficult to acquire stable configuration for evaluation. In addition, acquiring a practical HPC environment is almost impossible due to the dynamic variations in the number of users and resources at a particular moment, their characteristics, limited access, and inconsistent network conditions over the public network [1]. In addition, effective evaluation needs the study of RA technique using different user inputs and varying resource conditions. Therefore, we have created the same HPC simulated environment by managing predefined resource and network configurations such as the number of computing and storage resources connected by network links of various bandwidths randomly assigned within the range {1024, 2048} MB.

The heterogeneity in the modeled simulation environment was carried out by randomly generated resource characteristics, network bandwidths connecting computing and storage resources, file sizes, task workloads, and number of files required for each task. The data files requirements for each task were also randomly assigned in the range {x, y} showing minimum and maximum values respectively where x is assumed 1. This means that a task can demand at least 1 and at most y number of files. The files sizes are fixed at 100 MB each. To model the data files distribution, each of the data file was replicated on more than one storage resources. In our experimentation, we assume that there exist maximum 5 copies of any data file on the storage resources. The storage resources were decoupled from the nodes where computing resources are deployed. The computing resources were initially equipped with the full processing capabilities randomly chosen within the range {10,000, 40,000} Millions of Instructions per Second (MIPS). The files required by a task are either pre-fetched or transferred during execution.

When the data file fetched for some higher priority task on the same computing resource is used by the lower priority task or when the required file is locally available, the file transfer time is taken as zero. This technique exploits both temporal and spatial locality of data access. This file transfer incurs communication cost and time.

As discussed above, we evaluate our proposed technique on synthetic data sets where each task (data-intensive application) gets the required computing resource within the deadlines. The applications are scheduled for getting cloud resources using the RM scheduling algorithm. The workload for the smart devices represented in Figure 1 is generated in such a way that each device generates a job periodically after the interval of $\{\alpha, \beta\}$ seconds, where $\alpha = 100$ and $\beta = 10{,}000$. This means that each smart device sends a request for the cloud resources after the aforementioned time interval. The execution time e_i for each application $task_i$ was generated in the range {a, b} using normal distribution function representing the best and worst-case execution times respectively, and virtual machines are allocated accordingly. We have considered worst-case execution time equal to the $period_i$ for $task_i$ in order to ensure the tasks schedulability in any changing environment. In our task model, the period of the $task_i$ is equal to its deadline, i.e., $period_i = d_i$. Initially, tasks were assigned RM priorities such that the task with high rate has higher priority, where $rate = \frac{1}{period}$. The tasks from the superset are grouped into subsets on the basis of data files demands. It is understood that when the tasks set size increases, the number of VMs needed also increases.

The task groups as well as tasks inside each group are sorted on the basis of the RM priority assignment technique. Each task in the group has respective computation requirements and is entitled to get computational resource no later than the deadline. The tasks in a group are scheduled on the HPC system and computing resources are allocated on the basis of RM priorities. Each task generates multiple jobs. Each job is generated after an interval of $\{\alpha, \beta\}$ seconds. The experimentation was carried out by considering ifferent number of computing and storage resources. The above-discussed setting is subsumed in Table 3.

Table 3. Simulation parameters settings.

Parameters	Values
Bandwidth	1024~2048 MB
Task data files demand	$\{x, y\}$
File size	100 MB
Computing resource capacity	10,000~40,000 MIPS
e_i	$\{a, b\}$
$period_i$	100~10,000
Data files transfer cost	$\{1, 5\}$
Computing resource unit processing cost	$\{5, 50\}$

The computation time and cost of each job is summed into the computational requirements of each task. It is assumed that the communication cost of each job is minimal and is merged with the computation demand e_i of the task in our experimental setup. The data file is supplied to the task when it is requested from the storage resource and hence response time is zero. The transfer cost per unit size of the data file between data storage and computational resource was randomly generated between 1 and 5. In addition, the unit processing cost of the computing resources was generated between 5 and 50 depending upon the resource computing power. It is assumed that the file transfer within the same node incurs 0 transfer cost.

4.2. Performance Metrics

The proposed RA approach evaluates the HPC resource set for each task, and the overall objective is to minimize the total execution time and cost. The total execution time is the cumulative time consumed by the task set after assigning all task groups to the available computing resources. This time is also known as makespan and mathematically defined as follows.

$$Makespan = max(TT_1, TT_2, \ldots, TT_l), \tag{13}$$

where TT_j represents the total execution time of task group j. Similarly, the cost of a task set T is the overall cost incurred by all task groups. Mathematically,

$$Cost_T = max(Cost_1, Cost_2, \ldots, Cost_l), \tag{14}$$

and

$$Cost_j = \sum_{i=1}^{x} cost_i \tag{15}$$

The $cost_i$ is a combined cost incurred by a task processing on a computing resource and data files transfer taken by a $task_i$ in a group j. The time and cost for tasks context switching is negligible in our experiments and hence not included in the objective function.

5. Results and Discussion

In this section, we evaluate the performance of our proposed algorithm by comparing it with two methodologies, RDTA [12] and Greedy.

The makespan and cost minimization behavior of the proposed and the aforementioned two techniques was checked for the randomly generated task sets consisting of 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000 tasks. The plots reported in this paper are the average values of 300 runs of all the task sets. According to the task grouping criteria discussed in Section 3, the task sets are grouped into 5, 7, 7, 4, 8, 5, 7, 9, 9, and 10 groups respectively. Each group accommodates different number of tasks based on applied grouping criteria. The task grouping details are given in Table 4. The experiments were performed by checking system behavior on different number of computing resources. The number of computing resources was randomly generated within the range $\{10, 100\}$. We assume that the data storage resource gives a response immediately when a request is made by a task for data file access and hence the response time is ignored. The time delay in preparing the computing resource is also taken as zero because in our system, the computational resource is supposed to be ready for task execution as soon as the task arrives at that resource.

It was observed that for small task sets, fewer number of computing resources was involved as compared to the larger task sets. It is also understandable that choosing the proper number of computing resources can contribute to maintaining tasks' deadlines. If a lower number of computing resources is selected as compared to a large number of task sets, then it is likely that some tasks may not be RM feasible due to long waiting queues, which is crucial in real-time systems.

The main objective of this evaluation is to reduce the makespan and execution cost of the application while tasks deadlines are intact. Figure 4a,b depicts the normalized values of the makespan. The variation in magnitude depends upon the total number of tasks per task set, number of data files demands, and the computation and deadline requirements of each task. The lower the makespan value, the better the performance of the RA scheme. The other performance measurement criterion is the execution cost minimization. From Figure 5a,b, it is evident that decrease in makespan results in reduced processing cost.

It is known from Figure 4a,b, that the proposed technique continues to make scheduling decision by analyzing tasks feasibility on searching PNP sets and checking each scheduling point until some positive point t_P is found. Although, the size of PNP set for task group Y_x becomes large if the ratio between the periods of the first and the last task $\frac{period_n}{period_1}$ in Y_x is large which consumes time because large number of inequalities are tested, but this procedure enhances the chance of task feasibility because more $positive - negative$ points become available for testing tasks schedulability. Furthermore, all the initial feasible computing resources are encountered and the resource having minimum cost for the task execution is selected for task processing. The RDTA approach merely deals with executing tasks within deadlines and hence does not consider the cost parameters, which are considerably high in that case. In the case of the Greedy technique, the graph is steeply higher because a feasible resource is selected at random without taking into account the low time and cost constraints. So, the resources with high computing power are selected when termed feasible.
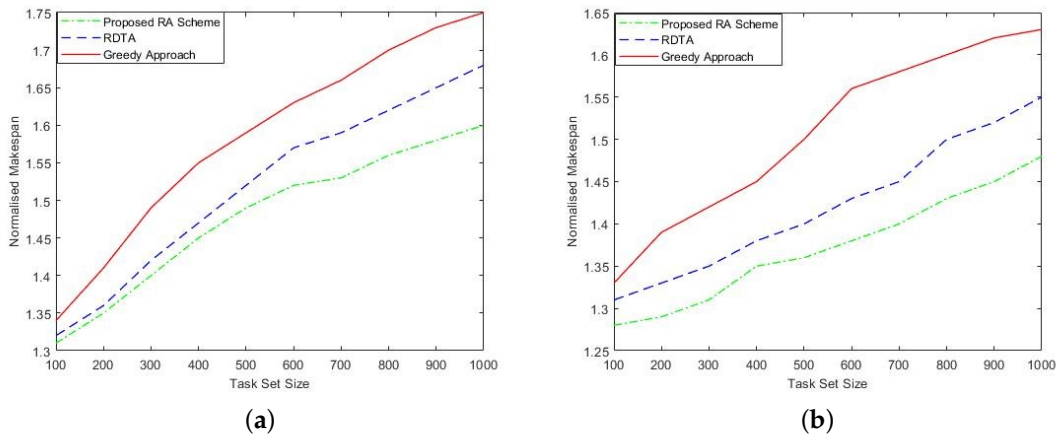

Figure 4. Average makespan for different task's requirements scenarios: (**a**) Normalised makespan for scenario 1. (**b**) Normalised makespan for scenario 2.
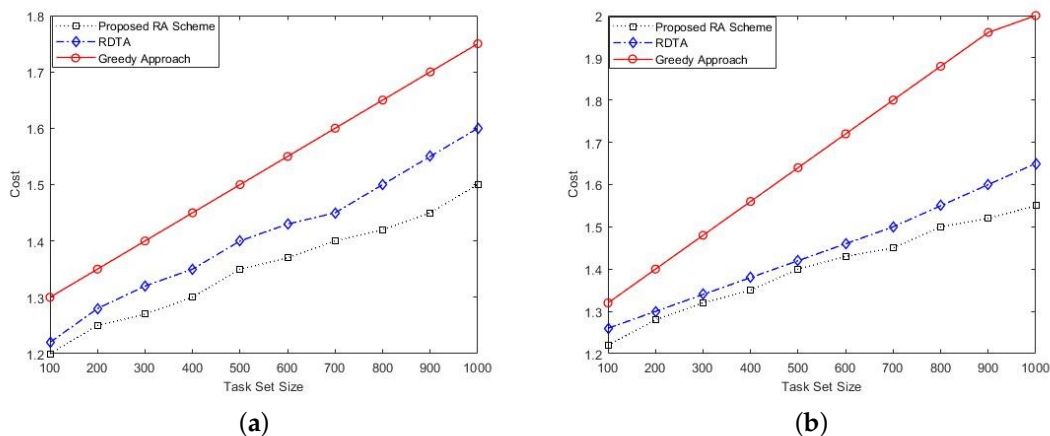

Figure 5. Average cost on two different task's requirements scenarios: (**a**) Cost for scenario 1. (**b**) Cost for scenario 2.

To further investigate the effectiveness of the proposed technique, we have conducted more experiments with different system settings. It is also noticeable from Figure 4a,b that the time taken

by all tasks test also increases uniformly as the number of tasks increase because more tasks are tested. It is obvious that the makespan of some task sets is high although the computing resources were operated at full speed because they need data files from remote storage resources which increase the total completion time. The resources when operating on full capacities consume high energy, but currently energy efficiency is out of the scope of this research. The situations where makespan is low demonstrates that the data files are locally exist or perfected for some higher priority tasks and do not need re-fetching for the lower priority tasks, which adds zero file transfer time to the overall execution time. The plots show that as the task set size increases, the makespan of the Greedy and RDTA grow, as opposed to the proposed approach. This growth in case of Greedy approach is because of making a greedy selection for the data storage and computing resources among multiple choices for data files accessing and task execution. This selection does not intelligently consider the minimization criteria. The RDTA mechanism also encounters high execution time and hence cost as shown in Figure 5a,b because the data file replication is not taken into account when making a choice for data files fetch among storage resources. In the case of the proposed approach, the ratio $\frac{period_n}{period_1}$ results in a larger value which constitutes larger PNP set that provides more points for schedulability checking. This phenomenon provides more opportunities for task scheduling and hence results in large number of tasks meeting the deadlines constraints.

Table 4 shows the formation of task groups in our experimental evaluation on the basis of randomly generated data files demands. The task groups are created as for as the inequality $GU_{Y_x} \leq n(2^{1/n} - 1)$ in Theorem 1 holds.

Table 4. Task groups.

Task Set Size	Group (No. of Tasks)
100	TG1(25), TG2(10), TG3(30), TG4(8), TG5(27)
200	TG1(31), TG2(5), TG3(53), TG4(12), TG5(48), TG6(32), TG7(19)
300	TG1(4), TG2(64), TG3(20), TG4(25), TG5(40), TG6(126), TG7(21)
400	TG1(101), TG2(32), TG3(130), TG4(137)
500	TG1(10), TG2(23), TG3(116), TG4(67), TG5(50), TG6(39), TG7(120), TG8(75)
600	TG1(146), TG2(3), TG3(43), TG4(201), TG5(207)
700	TG1(2), TG2(133), TG3(108), TG4(7), TG5(211), TG6(120), TG7(119)
800	TG1(234), TG2(21), TG3(233), TG4(41), TG5(19), TG6(115), TG7(123), TG8(5), TG9(9)
900	TG1(112), TG2(21), TG3(34), TG4(321), TG5(232), TG6(18), TG7(116), TG8(29), TG9(17)
1000	TG1(12), TG2(109), TG3(120), TG4(32), TG5(19), TG6(129), TG7(127), TG8(245), TG9(21), TG10(186)

5.1. Effect of Data Files Transfer on Performance

One of the basic components of calculating execution time is the data files transfer time incurred by transferring data files from the remotely located storage resources to the decoupled computing resources if the required files are not locally available. In addition to the makespan and cost values, two more performance measures considered in the evaluation results are the percent share of the data transfer time and local data access.

In our experiments, the percent share of the data files transfer time in the makespan calculation is evaluated. The Figure 6a,b plot the impact of the average data transfer time for the task sets. The lower value can put significant impact on reducing the overall makespan of the task set. As it is known from the task workload, the lower priority tasks scheduled on the same computing resource can utilize the same data files retrieved for the higher priority tasks in case the data requirements of the tasks are same. In that case, the data transfer time is zero. Additionally, the transfer time is also zero if the required file resides on the same node locally where the task is being executed. In this case, the more locally accessed files decrease the impact of remote data files transfer on the performance. It is less likely that the task is scheduled on the same computing resource for which all the required data files locally exist.

The above two factors can be correlated with the makespan calculation to indicate the impact of resource selection made by the RA scheme on achieving the decided objective. It is evident from Figure 6a,b that the Greedy and RDTA schemes do not intelligently adapt for the data files locality of access procedure and hence contribute to high data transfer percentage. The percentage of locality of access rises with the increase in the task set size. In comparison to the RDTA and Greedy counterparts, there is a high chance for the lower priority tasks to reuse the pre-transferred data files by using the proposed RA scheme. In addition, it is more likely that the assigned tasks find the required data files locally. The Greedy approach exhibits degraded performance because there is a very less probability of finding appropriate computing resource for tasks assignment.
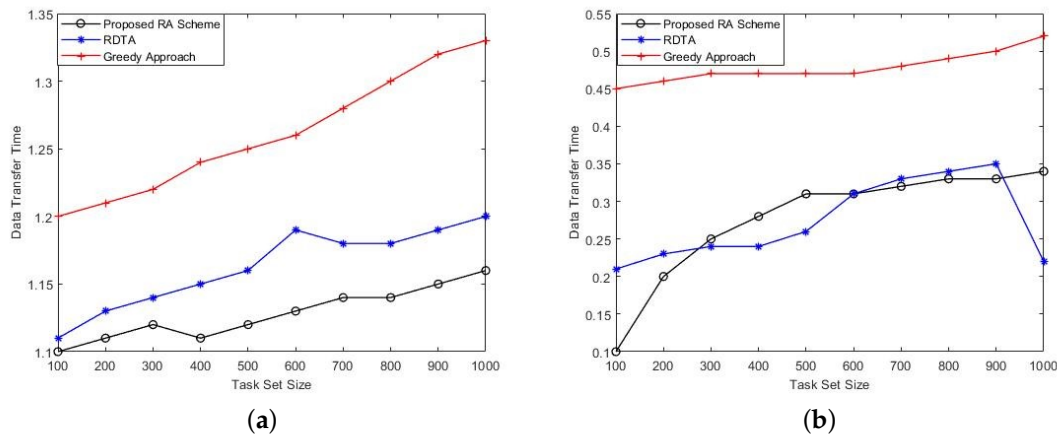

Figure 6. Effect of data transfer time on task sets in two different scenarios: (**a**) Data transfer time for scenario 1. (**b**) Data transfer time for scenario 2.

5.2. Impact on Resource Utilization

The utilization of the proposed RA scheme is measured on the basis of computing resources utilization in the HPC system. The resource utilization is directly related with the computation workload; when the task workload increases, the resource utilization also increases. The cumulative resource utilization can be calculated by the following equation.

$$Util_{cum} = \sum_{j=1}^{r} \frac{tasks\ workload\ processing\ time\ by\ a\ resource}{resource\ active\ time} \tag{16}$$

where $Util_{cum}$ represents the cumulative utilization of all computational resources spent on processing tasks workload, and r represents the total number of computing resources engaged in processing tasks sets.

It is observed from Figure 7 that the proposed RA scheme improves the resource utilization by keeping resources as busy as possible. The resource utilization is lower for the tasks sets having less number of tasks, but as soon as the number of tasks increases the resource utilization also increases. This means that the resource will be 100% utilized for the large task sets. This is an understandable phenomenon, because when the workload increases, more computational power is needed to complete tasks by their respective deadlines. If the computational power of the resources is relaxed for energy-efficient allocation, then it is very likely that some of the task groups may not be feasible. Moreover, this also will pertain to an unfair comparison. When the number of tasks increases, the proposed procedure pushes the system power to grow rapidly in order to accommodate more tasks to maintain the deadline constraints. This behavior results in high energy consumption but in this research we do not deal with the energy-efficient

perspective. Figure 7 reveals that implementing the proposed approach, the minimum system utilization is between 70% and 72% for small task sets but touches 100% when the task computational demands increase. The maximum system utilization approaches 85% by the other counterparts.

5.3. Failure Ratio

Failure ratio determines the ratio of tasks missed their deadlines, or the ratio of tasks which finished their execution after the deadlines. Mathematically,

$$Failure - ratio = \frac{Number\ of\ tasks\ missing\ deadlines}{Total\ number\ of\ tasks} \tag{17}$$
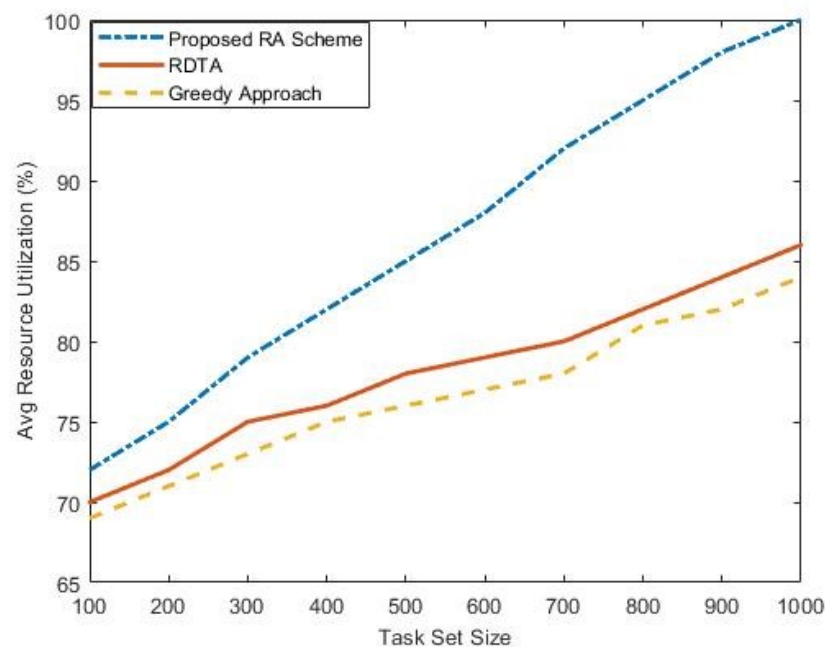

Figure 7. Effect on resource utilization.

The proposed algorithm is reactive in the context of deadlines missing by the tasks due to the efficient utilization of cloud resources, testing more and more PNP points, and task grouping on the basis of similar data files demands which skip the re-fetching of the same data files from the storage resources on the same computing resource again and again, which adds transfer time overhead to the makespan calculation. It is also obvious that the RM technique gives higher priorities to the tasks with shorter periods. In our case, since we assume that periods of the tasks are equal to the deadlines, so tasks with short deadlines have higher priorities.

The proposed RA methodology incorporates the scheduling points test as feasibility criteria for determining initial feasible resources, which is slower in performance as compared to the other scheduling techniques. This performance gap is due to testing task feasibility on all scheduling points for all tasks in a task set to offer more opportunity for finding feasible points. The other techniques, such as the iterative one, have the advantage of skipping large number of scheduling points in the feasibility analysis and concludes the task's feasibility very earlier. The proposed approach also does not consider the resources

power consumption and the energy perspectives when operating with high power to execute more tasks within deadlines.

6. Conclusions and Future Work

This paper presented the problem of resource reservation for smart systems in the cloud computing environment. The real-time systems deadlines generated by smart devices were respected by modifying the task model by incorporating data constraints. A task grouping technique was introduced that reduces the priority levels and execution time. The scheduling and resource allocation decision is driven by the need to improve the traditional performance parameters, such as resource utilization, and decrease the total application execution time and cost. In a cloud computing environment, the providers are incentivized by profit motives; while consumers would have a limited budget and would, therefore, like to execute the application at resources that provide services within the budget. In such an environment, both provider and consumers aim to improve their utility. This research is user-centric, which selects computing and data storage resources in such way that the makespan and cost is minimized while keeping the deadlines of the real-time system intact. The results were obtained through mathematical formulations by modifying the original task model to incorporate the task's data files requirements. The proposed resource allocation scheme was compared with RDTA and Greedy approaches and celebrated results were achieved.

As a future work, it will be interesting to rank the cloud resources on the basis of different criteria such as resource computational powers, storage capacities, imbalance workloads, and cost and allocate them using machine learning techniques. It is also desirable to include energy-efficient perspectives in the proposed research when resources operate with high power to execute more tasks within deadlines.

Nomenclature

Notation	Description
T	Task set
r_k	Release time of task k
e_k	Execution time of task k
d_k	Deadline of task k
CP_y	Computing power of resource y
PNP	Positive-negative points set
DF_k	Data files set required by task k
EET_{ky}	Execution time of task k on resource y
Y_x	Task group x
GU_{Y_x}	Group utilization of task group x
TU_i	Utilization of task i
CD	Compute and storage resource pair
CR	Computing resource set
t_P	Positive point
t_N	Negative point
TT	Total execution time
DR	Data storage resource set
R_w	Response time of the storage resource w
f_{kz}	File z needed by task k
$FT_{f_{kz}}$	Transfer time of the file $f_k z$
$card(T)$	Cardinality of task set T
dr_w	Data storage resource w

References

1. Venugopal, S.; Buyya, R. An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids. *J. Parallel Distrib. Comput.* **2008**, *68*, 471–487. [CrossRef]
2. Min-Allah, N.; Qureshi, M.B.; Alrashed, S.; Rana, O.F. Cost Efficient Resource Allocation for Real-Time Tasks in Embedded Systems. *Sustain. Cities Soc.* **2019**, *48*, 101523. [CrossRef]
3. Martel, S. How Cars Have Become Rolling Computers. Available online: https://www.theglobeandmail.com/globe-drive/how-cars-have-become-rollingcomputers/article29008154/ (accessed on 2 June 2020).
4. STATISTA. Number of Internet of Things (IoT) Connected Devices Worldwide in 2018, 2025 and 2030. Available online: https://www.statista.com/statistics/802690/worldwideconnected-devices-by-access-\technology/ (accessed on 25 May 2020).
5. Amazon Elastic Compute Cloud (Amazon EC2). Available online: http://aws.amazon.com/ec2/ (accessed on 5 June 2020).
6. Li, J.; Qiu, M.; Ming, Z.; Quan, G.; Qin, X.; Gue, Z. Online optimization for scheduling preemptable tasks on IaaS cloud systems. *J. Parallel Distrib. Comput.* **2012**, 72, 666677. [CrossRef]
7. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.H.; Konwinski, A.; Lee, G.; Patterson, D.A.; Rabkin, A.; Stoica, I.; et al. Above the clouds: A Berkeley View of Cloud Computing. Available online: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf (accessed on 12 June 2020).
8. Amadeo, M.; Molinaro, A.; Paratore, S.Y.; Altomare, A.; Giordano, A.; Mastroianni, C. A Cloud of Things framework for smart home services based on Information Centric Networking. In Proceedings of the IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), Calabria, Italy, 16–18 May 2017.
9. Stergioua, C.; Psannis, K.E.; Kimb, B.G.; Gupta, B. Secure Integration of IoT and Cloud Computing. *Future Gen. Comput. Syst.* **2018**, *78*, 964–975. [CrossRef]

10. Chen, H.; Zhu, X.; Guo, H.; Zhu, J.; Qin, X.; Wu, J. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *J. Syst. Softw.* **2015**, *99*, 20–35. [CrossRef]

11. Lee, Y.S.; Son, Y. A study on the smart virtual machine for executing virtual machine codes on smart platforms. *Int. J. Smart Home* **2012**, *6*, 93–106.

12. Qureshi, M.B.; Alqahtani, M.A.; Allah, N.M. Grid Resource Allocation for Real-Time Data-Intensive Tasks. *IEEE Access* **2017**, *5*, 22724–22734. [CrossRef]

13. Zorkany, M.; Hussein, M.; Kader, N. Real Time Operating System for the Internet of Things, Vision, Architecture, and Research Directions. In Proceedings of the World Symposium on Computer Applications & Research (WSCAR), Cairo, Egypt, 12–14 March 2016. [CrossRef]

14. Dickerson, R.; Gorlin, E.; Stankovic, J, Empath: A continuous remote emotionalhealth monitoring system for depressive illness. In Proceedings of the Wireless Health, San Diego, CA, USA, 10–13 October 2011.

15. Hishama, A.A.B.; Ishaka, M.H.I.; Teika, C.K.; Mohameda, Z.; Idrisb, N.H. Bluetooth-based home automation system using an android phone. *Jurnal Teknologi (Sci. Eng.)* **2014**, *70*, 57–61.

16. Pavana, H.; Radhika, G.; Ramesan, R. PLC based monitoring and controlling systemusing WiFi device. *IOSR J. Electr. Commun. Eng.* **2014**, *9*, 29–34.

17. Khalid, A.; Aslam, S.; Aurangzeb, K.; Haider, S.I.; Ashraf, M.; Javaid, N. An efficient energy management approach using fog-as-a-service for sharing economy in a smart grid. *Energies* **2018**, *11*, 3500. [CrossRef]

18. Mckinsey, By 2025 Internet of Things Applications Could Have 11 Trillion Impact, 2015. Available online: http://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internetof-thingsapplications-could-have\-11-trillion-impact/ (accessed on 28 May 2020).

19. Soliman, M.; Abiodun, T.; Hamouda, T.; Zhou, J.; Lung, C. Smart Home: Integrating Internet of Things with Web Services and Cloud Computing. In Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2–5 December 2013; pp. 317–320.

20. Son, J.Y.; Park, J.-h.; Moon, K.-d.; Lee, Y.-h. Resource-Aware smart home management system by constructing resource relation graph. *IEEE Trans. Consum. Electr.* **2011**, *57*, 1112–1119. [CrossRef]

21. Davidson, J. *Apple Homekit: The Beginner's Guide. Van Helostein*, 1st ed.; CreateSpace Independent Publishing Platform: Scotts Valley, CA, USA, 2017.

22. The Ambient. Available online: https://www.the-ambient.com/guides/samsung-smartthings-guidesmart-\home-163 (accessed on 20 May 2020).

23. Google Home Review (2018): The Smart Speaker that Answers almost Any Question, The Guardian. Available online: https://www.theguardian.com/technology/2017/may/10/google-homesmart-speaker-\review-voice-controll (accessed on 10 June 2020).

24. Liu, C.L.; Layland, J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM* **1973**, *20*, 40–61. [CrossRef]

25. Qureshi, M.B.; Alrashed, S.; Allah, N.M.; Kolodziej, J.; Arabas, P. Maintaining the Feasibility of a Hard Real-time Systems with Reduced Number of Priority Levels. *Int. J. Appl. Math. Comput. Sci.* **2015**, *25*, 709–722. [CrossRef]

26. Min-Allah, N.; Khan, S.U. A hybrid test for faster feasibility analysis of periodic tasks. *Int. J. Innov. Comput. Inf. Control* **2011**, *7*, 1–10.

27. Sha, L.; Goodenough, J.B. *Real-Time Scheduling Theory and Ada, CMU/SEI-88-TR-33*; Software Engineering Institute, Carnegie-Mellon University: Pittsburgh, PA, USA, November 1988; p. 15213.

28. Suciu, G.; Vulpe, A.; Halunga, S.; Fratu, O.; Todoran, G.; Suciu, V. Smart Cities Built on Resilient Cloud Computing and Secure Internet of Things. In Proceedings of the 2013 19th International Conference on Control Systems and Computer Science, Bucharest, Romania, 29–31 May 2013; pp. 513–518.

29. Liu, J.W.S. *Real Time Systems*; Prentice Hall: Upper Saddle River, NJ, USA, 2000.
30. Ye, X.; Huang, J. A framework for Cloud-based Smart Home. In Proceedings of the 2011 International Conference on Computer Science and Network Technology, Harbin, China, 24–26 December 2011; pp. 894–897.

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.