Wireless Personal Communications (2023) 128:2983–3001
https://doi.org/10.1007/s11277-022-10082-9

On Developing Framework for Schedulable Priority-Driven Systems: A Futuristic Review

Sarvesh Pandey[1] · Udai Shanker[2]

Accepted: 28 September 2022 / Published online: 15 November 2022

Abstract
Over time, systems' real-time data access requirements evolved, e.g., Real-Time Systems and Real-Time Database Systems and their variants. Assigning priorities to tasks/transactions in such a system has always been a critical decision as it forms a basis for allocating the limited number of shared resources optimally. This survey article studies the resource scheduling mechanisms of such systems. For resource scheduling, a priority is assigned to the smallest execution unit of the application, depending on the underlying scenario. The already existing resource scheduling algorithms are compared to make future recommendations – further exploration of all such unresolved open priority assignment policy-related problems is critical. Finally, we identify some new target technologies where one could foresee the future possibility of integrating custom-designed priority assignment policies.

Keywords Priority assignment · Real-time systems · Databases · Heterogeneous computing · Complex workflow scheduling

1 Introduction

The demand for designing innovative custom data-driven real-time applications is at an all-time high in today's digital world. Our society is arguably approaching a maturity level (in terms of utilizing the internet in this highly interconnected world). One can see the trials of the 5G internet all around the world that will change the world in the next few years. The real-time aspects of today's multi-node applications make them more human – mobility, heterogeneity, predictability, transparency, time-constraint enabled, and distributed nature. In Real-Time Systems (RTS), tasks are scheduled based on their priority for access to finite resources.

✉ Sarvesh Pandey
pandeysarvesh100@gmail.com

[1] Computer Science, MMV, Banaras Hindu University, Varanasi, India

[2] Department of Computer Science & Engineering, Madan Mohan Malaviya University of Technology, Gorakhpur, India

The journey of priority scheduling scheme design, in specific, begins with ever-growing research in RTS [1] and now covers newer technologies (e.g., cloud computing and the Internet of Things). Multiple research areas are originated from RTS, e.g., Real-Time Database Systems (RTDBS) [2], Distributed Real-Time Database Systems (DRTDBS) [3], Mobile DRTDBS [4], Replicated DRTDBS [5], Active DRTDBS [6], and deductive databases [7].

One common attribute in all these research domains is scheduling the resources for tasks/jobs/transactions/processes based on their time constraints and other factors, as the resources required are finite and limited. Many scheduling algorithms were designed in the past, and still, active research is going on in this direction to match the ever-going complexity of today's applications [8]. It will remain a relevant research field for a long time to meet the challenges set forth by today's societal needs. Instead of becoming obsolete, its relevance in system design has increased over time. Priority Assignment policies are also studied in the context of cloud computing [9] and fog computing [10] research domains for the completeness of this study. This generalized idea of understanding the evolution of priority assignment schemes historically revealed many similarities in how this problem is addressed across the domain.

The contribution of this survey paper is threefold. First, it revisits RTS, RTDBS, DRTDBS, and mobile DRTDBS with a central focus on the design of priority assignment policies. Second, it discusses how the relevance of designing priority scheduling methods has increased with the adoption of priority-based decision-making in new technologies, particularly cloud computing. Third, future research directions are noted as an outcome of an exhaustive study of priority assignment policies across computing domains.

The organization of the rest of this survey paper is as follows. Sect. 2, entitled " Priority Scheduling in Real-Time Systems,“ discusses the various existing priority schemes in the field of RTS. Sect. 3, entitled " Priority Scheduling in Deadline Driven Database Systems,“ discusses the need to design an entirely new set of priority schemes that are compatible with the unique nature of the given database application. First, the literature on priority assignment policies in the RTDBS environment is presented (in sub-Sect. 3.1). Sub-Sect. 3.2 focuses on how the application's distributed nature makes it hard to schedule the transactions (and associated cohorts running at multiple sites). Sub-Sect. 3.3 concentrates on covering the literature work on priority assignment policies in mobile DRTDBS. Sect. 4 is dedicated to cloud computing-based services and their way of assigning priorities. The existing priority heuristics are a foundation for designing priority heuristics for any new technological domain. In the end, Sect. 5 concludes the survey paper.

2 Priority Scheduling in Real-Time Systems

The RTSs have been a driving engine for a wide range of dynamically evolving real-life applications for almost the last five decades [11]. More specifically, it all started in 1967 when a study on scheduling two periodic synchronous tasks with implicit deadlines was presented [12]. Modeling tasks in RTSs has never been easy because of their complex nature and multi-criteria decision mechanisms.

Let us explain the underlying RTS terminologies to sketch a simple RTS scenario. Task execution is the only way to make changes in RTS. The task can be further categorized as a single-instance task and a multi-instance task [13]. The single-instance task is an aperiodic task, where the task invocation can happen only once. Moreover, a multi-instance task can invoke its instances in a repeated manner. Job is nothing but an invocation of a multi-instance task. Repeated executions of the same task instance are facilitated — execute the given job repeatedly depending on the scenario.

The multi-instance task can be categorized as a periodic and sporadic task. If the next job of a given task is executed just after the previous period's elapsed, then the task is a periodic one. So, the next job of a periodic task begins its execution right away when the period-in-run finishes. In simple words, the execution of the job is continually repeated at the end of the period. The recurrence of the job of a periodic task is implemented using clock interrupts. A sporadic task is different in terms of its job arrival sequences. Like, the arrival of the next job of a sporadic task cannot be predicted — it might reoccur at a random instant after a long time [14]. The only mathematical guarantee one has is that it would be after the elapse of minimum inter-arrival time. A worse scenario separates each instance of a sporadic task by minimum inter-arrival time. The aperiodic task has no restrictions associated with it regarding task arrival time. Two aperiodic tasks may get invoked at the same time. It also cannot be instantiated more than once [15].

Furthermore, a task-set is defined as a set of concurrently executing multi-instance tasks requiring access to the shared resources in the system. The task-set can be classified into three types based on the task type — periodic task-set, aperiodic task-set, and sporadic task-set. Based on task arrival time, the task set can be categorized as synchronous as well as asynchronous. If all the tasks associated with the given task-set arrive at time t=0, then it is a synchronous task set — the arrival of all the associated tasks happens in sync with each other. Otherwise, if the tasks of a given task-set have offsets associated with them, then it is called an asynchronous task-set — the arrival of the associated tasks are not in sync (the first task may arrive at the time t_1 while the second may arrive at the time t_2). Based on the deadline parameter, the task sets can be categorized as implicit-deadline task sets ($D_i = T_i$), constrained-deadline task sets ($D_i \leq T_i$), and arbitrary-deadline task sets (the deadline of a task can be smaller than, equal to, or greater than its period).

Fixed Priority Scheduling of any given task-set workload is mainly done using one of the following three state-of-the-art scheduling schemes.

1. **Fixed Priority Preemptive Scheduling (FPPS)**: The preemption is permitted to respect the priority order. This scheme is based on real-time competitive scheduling — the one with lower priority must not cause trouble to the one with higher priority.
2. **Fixed Priority Non-preemptive Scheduling (FPNS)**: Preemption is not permitted. This scheme is based on the simple idea of First Come, First Serve — irrespective of its priority, the currently running task must be allowed to finish its execution. The FPNS, a real-time priority scheduling scheme, ensures that priority order is followed for transaction enqueuing. It is just that if a high-priority task comes and requests a resource that is held by already executing a low-priority task (the low-priority task here is in the middle of its execution), then preemption would not be allowed. This scheme is not a true real-time scheduling scheme.
3. **Fixed Priority Deferred-preemptive Scheduling (FPDS)**: The preemption may be deferred for some time in case of priority inversion. Priority inversion is when a high-priority task is asked to wait for a resource already being held by some other low-priority task(s). The FPDS sets the non-preemptive region for each task. If the task is in

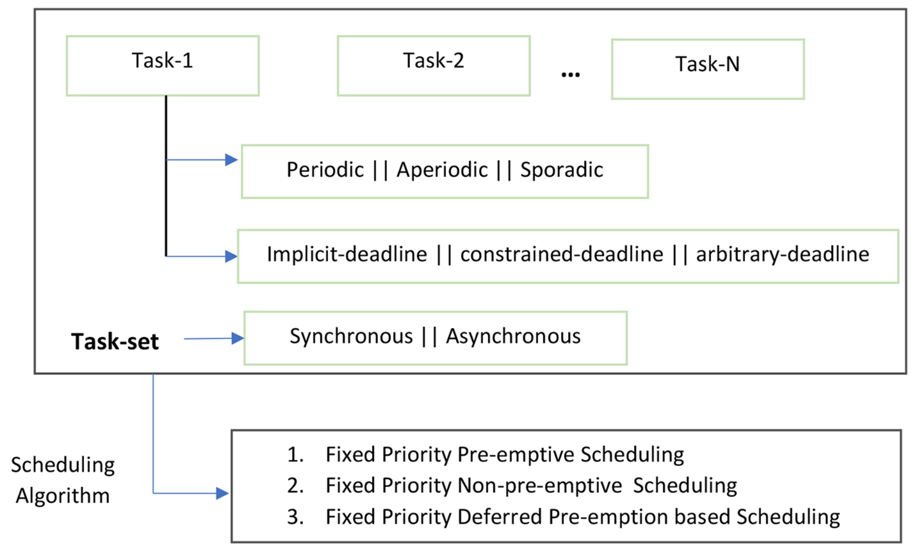
Task-1
Task-2
...
Task-N
Periodic || Aperiodic || Sporadic
Implicit-deadline || constrained-deadline || arbitrary-deadline
Task-set
Synchronous || Asynchronous
Scheduling Algorithm
1. Fixed Priority Pre-emptive Scheduling
2. Fixed Priority Non-pre-emptive Scheduling
3. Fixed Priority Deferred Pre-emption based Scheduling

Fig. 1 Terminologies & Variants of Scheduling Algorithms for RTS

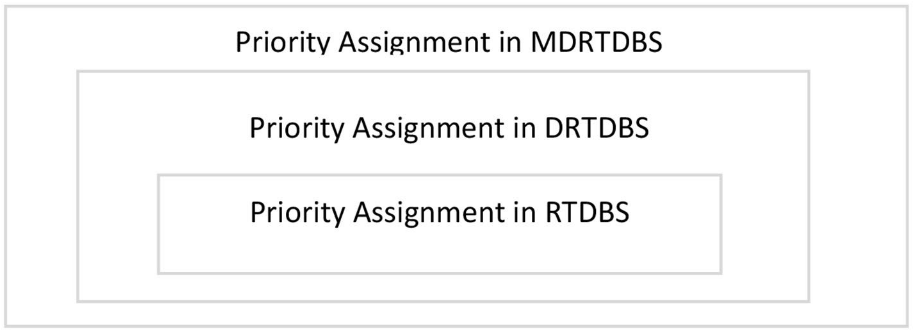
Priority Assignment in MDRTDBS
Priority Assignment in DRTDBS
Priority Assignment in RTDBS

Fig. 2 Logical Representation of Database Systems Design Complexity

a non-preemptive region, i.e., when it is close to its completion, it cannot be pre-empted even in case of priority inversion. The non-preemptive region is inclined towards the end of the task execution life cycle. The FPPS attempts to draw a fair trade-off between resource utilization and priority order.

One of the above-discussed priority scheduling algorithms, i.e., FPPS, FPNS, or FPDS, is utilized to schedule task-set workloads on a case-to-case basis for any real-life application. For any RTS, the variants of task-set and the widely used scheduling algorithms are pictorially represented in Fig. 1. The type of task heavily affects the performance of the priority scheduling algorithm employed and, ultimately, the system. For instance, one needs to consider whether the task is periodic, aperiodic, or sporadic and whether the associated deadline is implicit, constrained, or sporadic. The nature of the synchronous or asynchronous task set plays a crucial role in prioritizing schemes. Thus, one needs to consider this while choosing the priority scheduling algorithm in the RTS-based applications.

The foremost work on scheduling two synchronous periodic tasks with implicit deadlines on a single processor was reported in 1967 by Finberg and Serlin [12]. They considered the FPPS scheme for this purpose. The FPPS, a single-processor-based fixed-priority preemptive scheduling algorithm, was further extended by Liu and Layland in 1973. They came up with a new algorithm named Rate Monotonic Priority Ordering (RMPO) — this algorithm was designed for synchronous periodic task sets with implicit deadlines [16]. Almost a decade afterward, in 1982, a pioneer work on scheduling tasks was reported by Leung and Whitehead [17]. They argued that priority assignments should be done based on the deadline of a task instead of its period. The reason behind this argument was that, in practice, a task always has a deadline less than its period. Therefore, the driving parameter for priority assignment should be the deadline of a task. They named their algorithm the Deadline-Monotonic Priority Ordering (DMPO) algorithm.

The DMPO is a classical priority assignment scheme designed to prioritize tasks in the RTS. It is optimal for synchronous periodic task sets with constrained deadlines. Depending on the evolving nature of research problems, opportunities, and challenges, many variants of the DMPO algorithm are proposed. Last time, when this research area got full attention, several priority assignment policies were designed — mainly focusing on its enhancements [18]. The objective was to put more effort into getting the optimum possible schedulable sequence.

The optimality of DMPO can be broken up by adding an offset element in the scheduling scenario. Moreover, it is also not a suitable policy if one goes with arbitrary deadlines instead of constrained ones [19]. The optimality does not work with non-preemptive scenarios as well [20]. However, this research problem got very little attention afterward, particularly in the context of RTS.

Only milestone technical papers on the RTS's priority assignment policy have been reported, as this survey paper mainly focuses on the DRTDBS and other recent developments. In the next section (Sect. 3), the heuristics developed for the deadline-driven database systems are discussed/compared.

3 Priority Scheduling in Deadline Driven Database Systems

The complexity of database-based applications has been exponentially increasing and can be discussed in a sequence, from RTDBS [2], DRTDBS [3], to MDRTDBS [21]. The RTDBS is a live database system with tuned performance and throughput metrics for enhanced user experience [22], e.g., live streaming of audio/video through WhatsApp and banking. The RTDBS must be able to process/handle queries fetching temporal time-sensitive data through priority scheduling. The DRTDBS is a system with multiple RTDBS nodes connected through a network. In DRTDBS, the query can be even more complex as it may require data processing at multiple RTDBS nodes, e.g., processing sensor data stored in distributed databases.

The mobile DRTDBS logically connects multiple RTDBS mobile nodes through a network. Compared to the DRTDBS, in the MDRTDBS, dealing with mobile nodes for distributed data processing possesses additional challenges, such as computing nodes' energy needs and frequent disconnections. For example, Oracle Database Lite [23] involves portable mobile computing devices (e.g., smartphones and tablets) for data processing over a mobile network. It is not truly a mobile system as it has the feature of synchronization with stationary database nodes.

Figure 2 shows that designing a priority assignment policy for the MDRTDBS is more complex than the DRTDBS. This section covers the above three sub-areas, focusing on priority assignment policies. The design complexity of priority assignment policies depends on the application's complexity. Therefore, designing a suitable priority scheme (customized as per the needs) for complex applications is challenging.

3.1 Real-Time Database Systems

The real-time database systems are more complex than the real-time systems [2] [24], where one ensures consistency and meets deadlines. The smallest executable unit is the task for the RTS and the transaction for the RTDBS. Scheduling tasks is comparatively a less complex problem than a scenario where one needs to schedule transactions.

Most studies go with the Earliest Deadline First (EDF) policy for priority assignment of transactions. The EDF is the most straightforward and easy-to-implement policy, wherein the transaction with the earliest deadline is assigned the highest priority, i.e., the earlier the deadline of a transaction, the higher its priority will be. With time, researchers found that the EDF policy has some disadvantages requiring further investigation. At first, it favors short-lived transactions (or penalizes long-lived transactions). Second, the criticality aspect of a transaction is not considered in the EDF, as the criticality of a transaction has nothing to do with its size [25]. Third, it is known that the EDF does not perform well under high load conditions (confirmed by many researchers). It is because high-priority early deadline transactions are more prone to deadline miss, particularly when the workload is high. Such deadline misses can lead to system resource wastage [26].

The First attempt to improve the EDF is made in [27]; here, the adapted earliest deadline (AED) policy is proposed. As the name suggests, the AED improves the overall RTDBS performance using the mechanism based on adaptive admission control. In AED, the incoming lock requesting transaction is assigned to the HIT/ MISS group – the HIT group capacity is further dynamically tuned, resulting in improved system performance. Later, the AED policy was modified/improved, and the adapted earliest virtual deadline (AEVD) policy was proposed [28]. The AEVD explicitly addresses the problem of biases toward short-lived transactions. Datta et al. later developed an extension of AEVD; the adaptive access parameter (AAP) policy is presented [29]. The AAP is based on explicit admission control.

There is one more study available in the literature, by Dogdu et al., on balancing the biases of EDF towards short-lived transactions by utilizing transaction execution histories [30]. Here, the execution history of transactions forms a basis for priority assignment. The idea is that the miss rate for short-lived and long-lived transactions should be quantitatively similar. The Generalized Earliest Deadline First (GEDF) policy is proposed and compared against EDF [31]. The GEDF has the following salient features.

1. Introduction of the new parameter named "transaction importance" to be taken care of while assigning priority to transactions.
2. Use of Deadline and Importance Criterion in Scheduling Transactions

The GEDF policy is further extensively studied by Kaddes et al. from the perspective of tuning the "transaction importance" parameter named "SPriority" [32]. The performance of the RTDBS system is evaluated under varying values of SPriority, and optimal values of system parameters are suggested [33].

3.2 Distributed Real-Time Database Systems

The DRTDBS is a logical integration of multiple geographically separated RTDBSs [34]. The transaction processing domain in the DRTDBS can be divided into the following sub-domains – priority assignment, concurrency control, and commit processing. This survey paper focuses on studying priority assignment schemes. For a complete understanding of all the components of transaction processing in DRTDBs, readers can refer to [35] [36] [3] [37] [6] [8] [38].

Assigning priority to transactions in the DRTDBS setting is a complex job compared to the RTDBS setting. A transaction can be distributed in nature (instead of being a local one, as in the RTDBS). Therefore, processing a distributed transaction requires a mechanism to divide it into sub-transactions (cohorts) depending on the associated data access requirements. This results in a scenario where the local processing by cohorts at multiple distant locations affects the overall global transaction execution life cycle.

In [39], it has been proved that the DRTDBS application's performance can be improved by performing priority assignment activities at a cohort level. Four priority schemes were suggested – (i). Ultimate deadline (UD) (ii). Effective deadline (ED), (iii). Equal slack (EQS), and (iv). Equal Flexibility, considering the sequential execution model. Although these schemes consider the deadlines of participating transactions, all four schemes fail to address increased data contention level that shoots up in an uncontrolled manner at a high load.

The data contention issue with the above four schemes is studied in [40]. As a solution to this problem, modified and more flexible schemes were proposed. These schemes covered both the aspects of assigning priority to transactions – deadline requirements and data contention. The simplest scheme proposed was the NL (Number of Locks held) priority scheme. Here, the priority to the current cohort is assigned based on the total number of locks held by its parent transaction at the time of its invocation. Another scheme proposed was the static EQS (SEQS), which is nothing but the extended version of the EQS to reduce the negative impact of data contention on the system caused by the EQS. Further, a hybrid priority assignment scheme named Mixed Method (MM) is proposed that considers both the criteria in priority assignment – time-constraint and data contention level. The MM scheme was the first of its kind that attempted to come up with a balanced multi-criterion priority assignment approach. However, all the above-discussed schemes, except the UD priority scheme, are not suitable for the parallel transaction execution framework.

The parallel transaction execution framework has distributed deadline dependency – overall transaction execution time depends on a cohort that finishes its execution at last. Late execution of any of the cohorts may lead to a deadline miss of the associated global transaction. This makes it hard for a global transaction to meet its deadline because of multi-party involvement in completing its execution – it is likely that one of its cohorts might be running at an overloaded site.

The heuristic to assign priorities to cohorts of a distributed transaction (with parallel running cohorts) is proposed in [41] [42]. This heuristic favors the cohort with a larger number of locks – the higher the number of locks held, the higher the priority. The parallel cohort requiring more locks is assigned a comparatively higher priority. The fairness aspect of transaction scheduling is also touched upon in this study. Whenever a data conflict occurs, a near-to-completion cohort is favoured if possible. This reduces the wastage of resources and might lead to a scenario where the conflicting transaction may cooperatively complete their execution.

The communication delay in sending/receiving messages should also be considered in designing a priority assignment scheme. With global transactions, access to remote data items is facilitated through communication between the parent cohort and the cohort where the given remote data item physically resides. The role of communication delay in assigning priority to transactions was first studied by Chen Hong-Ren et al. [43]. They proposed a priority scheme named Flexible High Reward (FHR). The FHR aims at reducing transaction deadline miss percentage by favoring the remote cohorts, i.e., assigning comparatively higher priority to the remote cohorts. It is an attempt to balance the intrinsic communication delay involved in accessing remote data items.

The existing deadline-based priority assignment heuristics for time-constrained databases are studied and concluded that such heuristics do not perform better in terms of the percentage of transactions missing their deadlines; they fail miserably in the high data contention environment. Therefore, to address the issue of high data contention, the Most Dependent Transaction First (MDTF) heuristic has been proposed [44]. The MDTF utilizes the dependent transactions sizes of all directly competing transactions in the computation of their priority value. Performance studies have shown that MDTF provides a trade-off between the NL and EQS heuristics from the performance perspective due to its better handling of data contention. As an extension to the MDTF, the contention-aware equal slack (CA-EQS) policy [45] is also proposed. The CA-EQS is an advanced version of MDTF — it performs better than the MDTF. The MDTF is purely based on dealing with data contention, while the CA-EQS considers real-time constraints as well as checks on the data contention level.

3.3 Mobile Distributed Real-Time Database Systems

The MDRTDBS research area requires utmost attention by the database community as, to date, only a little work has been reported [21]. Though developing custom concurrency control protocols for the MDRTDBS has gained some attention in recent years [4], only a few efforts have been made to develop custom priority assignment policies. The readers are referred to [46], [47], and [48] for the noted development in the field of concurrency control (one of the key sub-domains of transaction processing) – all the three research articles considered the Earliest Deadline First (EDF) for priority assignment. This makes it hard to claim that the existing mobile-based concurrency control algorithms are effective under real-life mobile scenarios.

The policies, i.e., NL [40] and CA-EQS [45], designed specifically for the DRTDBS, are incompatible with the MDRTDBS applications for obvious reasons – the additional complexities intrinsically added with consideration of mobile nodes. In [49], a modified version of the already existing NL heuristic is proposed. The heuristic is designed considering the MDRTDBS environment. In this heuristic, the data contention is linked with the type of lock

Table 1 Evolution of Priority Assignment Policies in Databases

Database Environment	Priority Assignment Policy	Pros	Cons
RTDBS	EDF [26]	• widely accepted • simple to implement	• Favors short-lived transactions • Criticality aspect not considered, and • Not suitable for high workload scenarios.
	AED [27]	• Improved EDF variant • Enhanced performance through the integration of adaptive admission control.	Biases toward short-live transactions present
	AEVD [28]	• Extended version of AED • adverse effects of biases on long-lived transactions reduced.	Criticality aspect not addressed in AEVD.
	AAP [29]	• The modified version of AEVD • Inclusion of explicit admission control.	Criticality is not addressed
	GEDF [31]	• Utilizes the transaction execution history to solve the bias in EDF • Addresses the criticality aspect as well.	Working with transaction execution history creates extra overhead.
DRTDBS	FHR [43]	• is based on the simple idea that communication delay should be counted in designing a priority heuristic. • Remote cohorts are favored as their execution also involves communication costs.	No consideration of the contention caused by hot data items.
	MDTF [35]	Based on how many transactions depend on the requesting transaction – the transaction with the highest dependency length would have the highest priority.	• This heuristic considers only one aspect – contention. • The deadline of a transaction is ignored.
	CA-EQS [45]	• An extended version of the MDTF heuristic. • Considers both the deadline requirements and the data contention in a balanced manner.	Fine parameter tuning is required.
MDRTDBS	NWL [43]	• Designed for the MDRTDBS environment. • The idea is that write locks contribute more to data contention than read locks.	No consideration of mobility aspects.

(read or write) on any given data item. It is claimed that, contrary to the NL heuristic, write locks contributes more to the data contention and therefore need to be handled differently [50]. This heuristic is further enriched by considering one more aspect of priority computation – the size of the prerequisite set of data items required by a cohort of any global transaction [51]. However, the existing MDRTDBS priority assignment policies are not mature as the intrinsic mobility issue is not yet efficiently handled. The protocols discussed in this section are summarized in Table 1.

The above table summarizes the advancements in designing priority assignment schemes with a sole focus on real-time database applications

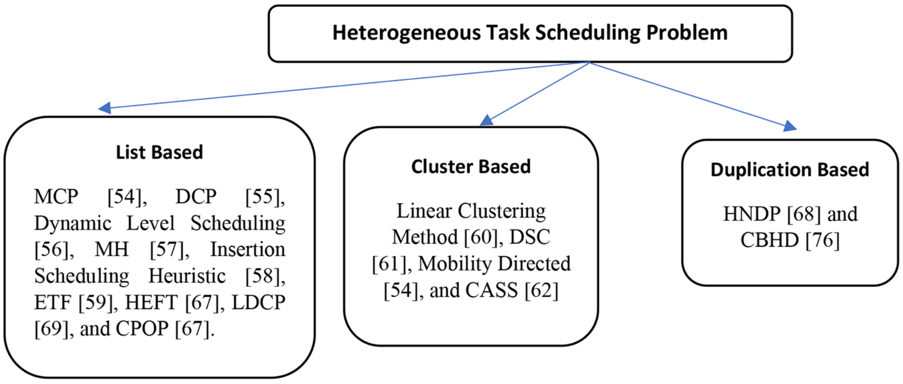
Heterogeneous Task Scheduling Problem
List Based
MCP [54], DCP [55], Dynamic Level Scheduling [56], MH [57], Insertion Scheduling Heuristic [58], ETF [59], HEFT [67], LDCP [69], and CPOP [67].
Cluster Based
Linear Clustering Method [60], DSC [61], Mobility Directed [54], and CASS [62]
Duplication Based
HNDP [68] and CBHD [76]

Fig. 3 Broad Categorization of Algorithms Solving Heterogeneous Task Scheduling Problem

4 Priority Scheduling of Tasks in Cloud-Based Real-Time Database Applications

Soon after the advent of the internet in the early nineties, the beginning of the 21st century marked extensive research in cloud computing [9]. Many applications have migrated to the cloud recently and were previously deployed on on-prem dedicated node-based set-ups. Businesses/individuals started relying more on cloud resources for their unpredictably varying IT infrastructure needs rather than owning the costly hardware infrastructure [52]. This resulted in the broad adoption of cloud-related services across industries [53]. In the interest of not losing track of studying priority assignment policies, the discussions are restricted to studying the priority schemes only.

Scheduling of tasks in heterogeneous computing systems is an exciting research area – lots of work has been done in this direction in the last two decades. This topic remains relevant even today and continues to grab the attention of the research community at large. Applications based on the static model can be represented as the Directed Acyclic Graph (DAG), where nodes represent tasks and edges represent dependencies amongst tasks. Following are the characteristics of all such applications – dependency between tasks, the execution time of each task on each processor, and communication cost between two adjacent tasks connected through an edge.

In a simple way, the objective here is to schedule tasks belonging to an application to one of the available heterogeneous compute resources (heterogeneous processors) in such a way that the overall performance can be improved and task-precedence requirements are met. The parameter to assess the performance can be the overall completion time. Researchers have extensively studied the static task scheduling problem, and their solutions can be widely categorized as list-based algorithms, clustering-based algorithms, duplication-based algorithms, and guided random search based algorithms. Figure 3 briefly represents algorithms under each of these categories.

The list-based scheduling algorithms generally provide an ordered list of tasks. Here, priority is assigned to each task. Further, a task is selected for execution based on its priority. Once it's decided which task will execute, the next question is which processor the selected task will be with. This decision is based on the fact which processor is going to minimize the

Table 2 Chronological Developments in the Cloud Scheduling Literature (2002–2020)

Year	Cloud Scheduling Algorithm	Objectives Considered	The Pros and Cons
2020	DRHEFT [74]	MTTF	— provides a better SER-LTR trade-off using fuzzy dominance. — Based on the single fault tolerance assumption that further requires generalization to increase its applicability.
2018	E-HEFT [72]	Makespan and load balancing	— Reduces makespan of a workflow and improves load balance among VMs. — Virtual machine selection is facilitated using many-to-one game matching, considering tasks and VMs as players.
2018	Modified HEFT [73]	Makespan and load balancing	— Makespan time is reduced compared to HEFT. — The overload issue as well is addressed. However, the uniformity of load distribution remains a cause of concern.
2015	CEAS [71]	Execution Cost and energy consumption	— The evaluation is done using Cloudsim. Four real-time workflows are considered for performance evaluation. — Outperforms existing algorithms considering the cost involved (in $) and energy consumption (in Kilowatt).
2014	MPQGA [75]	Makespan and SLR	— A GA-based stochastic search method for heterogeneous computing systems. — Utilized GA for priority assignment while heuristic-based HEFT search for task-processor mapping.
2012	CBHD [76]	Makespan, Load Balancing, and Processor Utilization	— Intelligent use of clustering and replication techniques with the HEFT resulted in reduced makespan, improved load balancing, and increased processor utilization. — Adding clustering and replication components increased the system's complexity.
2011	ECS [70] and ECS + idle [70]	Energy Consumption, Makespan	— Energy saving goals are achieved using the DVS technique. Better power management is facilitated by recent processors (Intel and AMD). — The shorter the makespan value, the lesser would be the energy consumption requirements. This is because idle processor slots also add to energy consumption.
2008	LDCP [69]	Normalized Schedule length and speedup	— The new LDCP attribute is defined, which helps assign priorities to tasks in a workflow. — Best suited for workflows with high communication costs. — Possible integration of the LDCP algorithm with already existing optimizations, for instance, task duplication, should be explored and evaluated.
2005	HNDP [68]	Schedule length ratio	— Priorities are assigned to tasks based on a decisive path. — If the processor is idle, an attempt is made to duplicate the predecessors' currently running tasks in the order of most to least favorite. — HNDP outperforms HEFT. The performance improvement becomes even more evident on increasing the CCR value. — Requires slightly higher compute resource (approx. 5%) to achieve the same makespan value as it is a task duplication algorithm.
2002	HEFT [67] and CPOP [67]	Makespan	— Both HEFT and CPOP algorithms were proposed in the same research article. The performance results with HEFT are better than that of CPOP. — HEFT is a pioneer list-based heuristic algorithm. Many researchers validated their results against HEFT even when it is almost two decades old. The HEFT algorithm suffers from the resource overload problem.

pre-defined cost. The algorithms of this category were found to be more practical and promising than those of other categories. Some of the list scheduling heuristics in this category are the Modified Critical Path (MCP) [54], Dynamic Critical Path (DCP) [55], Dynamic Level Scheduling [56], Mapping Heuristic (MH) [57], Insertion Scheduling Heuristic [58], and earliest Time First (ETF) [59].

The clustering-based algorithms map the DAG graph tasks with the unbounded number of clusters. Then, these unbounded number of task clusters are merged iteratively to the point where the number of clusters becomes equal to the number of processors. Further, the finalized task clusters are mapped with and scheduled on a bounded number of processors. Tasks belonging to the same cluster must be executed using the same processor. Moreover, within each processor, task executions are ordered as well. Some of the scheduling heuristics in this category are the Linear Clustering Method [60], Dominant Sequence Clustering (DSC) [61], Mobility Directed [54], and Clustering & Scheduling System (CASS) [62].

The task duplication based algorithms generally assume that the system consists of an unbounded number of identical processors. The algorithms in this category were not practically viable because of the significantly higher time complexities compared to other categories [63]. As the name suggests, the guided random search algorithms use randomized search in a guided manner to navigate through the problem space [64]. This is an iterative process where knowledge gained in the previous iteration is combined with a randomized feature to generate new results [65]. The algorithms based on the idea of guided random search could be the right choice for some applications as they provide good quality DAG schedules. However, they require a significantly higher execution time to reach a solution compared to the alternatives available [66].

4.1 The HEFT Algorithm and its Variants

The HEFT algorithm [67], proposed in 2002 by Topcuoglu et al., is undoubtedly a founding pillar for most list-based scheduling algorithms. This subsection particularly focuses on how the scheduling 'heterogeneous computing resources' research is carried out with the HEFT algorithm kept in the centre. The heterogeneous computing resources are nothing but 'heterogeneous virtual machines hosted in a cloud environment. The objective of any such workflow scheduling can include cost, makespan, load balancing, reliability awareness, security awareness, keeping a check on energy consumption, and abiding by service level agreements (SLAs) – different customer has a different set of SLA requirements. The HEFT algorithm is designed with only one objective: makespan.

The same research article also discussed another algorithm named Critical Path on a Processor (CPOP). HEFT performs better than CPOP; therefore, most researchers considered it a base protocol while proposing their version of cloud workflow scheduling algorithm for multiprocessors. The Heterogeneous N Predecessor Decisive Path (HNDP) algorithm is also proposed further to improve the system's performance [68]. It is an extension of the CPOP protocol. This algorithm injects/ integrates the task-duplication based approach with HEFT. Whenever the processor resource becomes idle, it is utilized to execute the predecessors of tasks. Efficient utilization of idle slots further improves performance. Later, a Longest Dynamic Critical Path (LDCP) algorithm is proposed [69]. In line with HEFT, CPOP, and HNDP, this algorithm is of single objective nature and attempts to reduce the makespan of a workflow.

It is to note that task-duplication approaches are best suited to communication-intensive applications only. The reason is simple: task-duplication helps reduce communication overhead, resulting in reduced makespan value. However, from an energy consumption perspective, the benefits of task duplication come at the cost of increased energy consumption. Running a precedence-constrained parallel workflow on a multiprocessor computing node can be better facilitated when the energy consumption aspect of it is considered as well, in addition to factors like makespan and processor utilization. The HEFT protocol is further improved, and an energy-conscious component is incorporated. Two energy-conscious heuristics – Energy Conscious Scheduling (ECS) and ECS + idle – were proposed for workflow scheduling [70]. Both the heuristics are different in the way that they measure energy consumption differently. The Dynamic Voltage Scaling (DVS) technique, in-built into recent processors, is utilized by the above heuristics. These heuristics are a bi- criterion in nature as they consider two objectives: makespan and energy consumption. The Cost and Energy Aware Scheduling (CEAS) algorithm [71], as well, is developed for workflow scheduling on a multiprocessor system with a focus on execution cost and energy consumption. Here, the energy consumption is reduced through custom task merging methods. In addition to energy consumption, it is also tuned to take care of budgetary requirements.

Researchers found that the HEFT protocol does not work well in terms of balancing loads amongst virtual machines. An Enhanced HEFT (E-HEFT) algorithm is proposed to address the load balancing problem [72]. The E-HEFT is a bi-criterion algorithm that attempts to fulfill two objectives, i.e., makespan and load balancing. The modified HEFT (M-HEFT) algorithm is a bi-criterion algorithm that considers makespan and load balancing [73]. However, no comparative study is available to date to showcase which one among the above two (E-HEFT and M-HEFT) is superior from a performance perspective. The effort should be made in this direction as leveraging positive features of both schemes through logical integration might result in an improved version of the algorithm.

For any real-time embedded applications, a heterogeneous Multiprocessor System-on-chips (MP-SoCs) environment can be utilized as it can facilitate parallel processing while keeping a check on power consumption. Any hardware system whatsoever could not be free from faults. MP-SoCs are vulnerable to two types of faults – transient and permanent. Transit fault occurs only for a very small period, and the system recovers on its own from this type of failure. However, as the name suggests, the impact of permanent faults cannot be averted until the hardware causing the fault is replaced/ repaired. The measure of transit fault is done using the reliability component named soft-error reliability (SER). The measure of permanent fault is done using the reliability component named Lifetime Reliability (LTR). The system must address the reliability-related issues simultaneously. The HEFT algorithm does not consider the SER-LTR co-optimization problem. The deadline-constrained reliability-aware HEFT (DRHEFT) protocol has been developed to address this problem [74]. The performance results show that the DRHEFT does well in terms of forging a better trade-off between SER and LTR compared to other existing state-of-the-art reliability-aware HEFT algorithms.

Researchers tried exploring a few other less common techniques as well to address the multiprocessor workflow scheduling problem. The Multiple Priority Queues Genetic Algorithm (MPQGA) is developed for scheduling tasks of heterogeneous computing systems [75]. Here, a genetic algorithm-centered approach is utilized for the priority assignment of tasks. Similarly, the Clustering Based HEFT with Duplication (CBHD) algorithm attempts to integrate the positive features of both clustering and task-duplication [76]. This reduces makespan, balances load among processors, and increases processor utilization.

Table 2 briefs cloud scheduling protocols discussed in this section, their objectives, and pros/cons. As seen in Table 2, the cloud computing research community has kept the clock ticking by coming up with improved versions of cloud resource scheduling algorithms regularly to meet the constantly changing customer requirements. Anticipating that in coming years, with a boost in big data and the internet of things (IoT) domain, more applications would require massive computing power. That means the computer science community still has a long way to go. Readers interested in more detailed discussions in this regard may refer to [77].

5 Conclusion and Future Work

The choice of priority assignment scheme has always been critical in assessing the performance of time-constrained applications. In this survey article, the progress made in the field of priority assignment techniques is presented in a time-series manner from a holistic point of view. An attempt is being made not to tie the core priority assignment problem with any specific application; instead, the focus was to investigate the already existing and near-future application domains that prioritize their processes in one way or another. To conclude, it is expected that in the near future, more custom applications will be integrated with the prioritization concept to make them address the real problems end customers face these days.

5.1 Future Research Directions

As it can be concluded from the discussions, the research domain on designing a priority assignment policy is not new. However, the design requirements change with the change in application nature. As a result, this topic has been studied across knowledge domains. Unfortunately, as far as our literature survey is concerned, researchers put little to no effort into looking at this problem holistically and investigating the pros and cons of priority assignment schemes across domains.

We are of the firm opinion that leveraging cross-domain algorithmic ideas has the potential to benefit the research in this direction. In the future, after reading extensive research literature on priority assignment schemes, we anticipate following key research directions requires further extensive investigation by the computer science community in the coming days.

1. Designing priority assignment schemes for complex distributed real-time systems is critical since most of the applications we rely on are distributed [11]. Managing the execution of a task distributed across a number of processors depends on multiple factors.
2. The research area of priority assignment policies for transaction scheduling has been widely discussed. The RTDBS and DRTDBS research domains considered designing the priority assignment scheme as a soul part of their framework [3]. However, very

little effort has been made to develop custom priority schemes for the MDRTDBS. This issue needs to be addressed as the community will rely more on mobile devices in the next few years.

3. In the context of priority inversions and overload conditions, the growing complexity of applications warranted designing priority schemes to provide increased concurrency among competing transactions [6].
4. Maurya 2018: An extensive effort is needed in the context of designing a general benchmark to assess the performance of workflow scheduling algorithms in a heterogeneous computing environment. Some effort in this direction has been made in [78]. However, developing an accurate benchmark for this purpose requires further exploration.

Open-source benchmarks will not only fasten the research in these directions but also attract more interest from around the globe – this will help researchers focus on improving the existing algorithms instead of trying to implement what is already implemented. We can achieve this only through a more open, transparent, and collaborative approach.

Funding The financial support from Banaras Hindu University (BHU), India, under IoE Grant is acknowledged.

Data Availability This research work utilizes no data or dataset.

Code Availability Since the article is of survey type, coding the already existing idea was not of our interest. However, an attempt is made to ensure that all the articles surveyed are from authentic sources with pointers to code repositories. The past research articles are assumed to be correct regarding conclusions drawn and the results presented.

Declarations

Following declarations are made to ensure transparency.

Conflict of Interest The authors of this paper have no conflict of interest regarding the publication of this research article.

References

1. Xu, J., & Parnas, D. L. (2000). Priority scheduling versus pre-run-time scheduling. *Real-time systems*, *18*(1), 7–23
2. Kao, B., & Garcia-Molina, H. (1993). An overview of real-time database systems. *Real Time Computing*, *127*, 261–282
3. Shanker, U., Misra, M., & Sarje, A. K. (2008). Distributed real time database systems: Background and literature review. *International Journal of Distributed and Parallel Databases, Springer Verlag*, vol. 23, no. 02, pp. 127–149
4. Lam, K., & Kuo, T. (2002). Mobile distributed real-time database systems. *Real-Time Database Systems* (pp. 245–258). Boston, MA: Springer
5. Arun, A., Pandey, S., & Shanker, U. (2021). A Multi-Replica-Centered Commit Protocol for Distributed Real-Time and Embedded Applications. *International Journal of System Dynamics Applications (IJSDA)*, *10*(4), 1–19
6. Pandey, S., & Shanker, U. (2020). Transaction Scheduling Protocols For Controlling Priority Inversion: A Review. *Computer Science Review*, *35*, 100215
7. Minker, J. (2014). *Foundations of deductive databases and logic programming*. Morgan Kaufmann

8. Pandey, S., & Shanker, U. (2021). Performance Issues in Scheduling of Real Time Transactions. *Proceedings of the 26th International Conference on Database System for Advance Applications (DASFAA-2021), Taipei, Taiwan*
9. Buyya, R., Broberg, J., & Goscinski, A. M. (2010). *Cloud computing: Principles and paradigms* (87 vol.). John Wiley & Sons
10. Yi, S., Li, C., & Li, Q. (2015). A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pp. 37–42
11. Davis, R. I., Cucu-Grosjean, L., Bertogna, M., & Burns, A. (2016). A review of priority assignment in real-time systems. *Journal of systems architecture*, *65*, 64–82
12. Fineberg, M. S., & Serlin, O. (1967). Multiprogramming for hybrid computation. *In Proceedings of fall joint computer conference*, pp. 1–13, November 14–16
13. Choi, S., & Agrawala, A. (1998). Scheduling aperiodic and sporadic tasks in hard real-time systems.
14. Jeffay, K., Stanat, D., & Martel, C. (1991). On non-preemptive scheduling of periodic and sporadic tasks. *In IEEE real-time systems symposium*, pp. 129–139
15. Isovic, D., & Fohler, G. (2000). Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. *In Proceedings 21st IEEE Real-Time Systems Symposium*, pp. 207–216
16. Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, *20*(1), 46–61
17. Leung, J. Y. T., & Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, *2*(4), 237–250
18. Goossens, J., & Devillers, R. (1997). The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, *13*(2), 107–126
19. Lehoczky, J. P. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proceedings 11th IEEE Real-Time Systems Symposium*, pp. 201–209
20. George, L., Rivierre, N., & Spuri, M. (1996). Preemptive and non-preemptive real-time uniprocessor scheduling. *Doctoral dissertation, Inria*
21. Swaroop, V., & Shanker, U. (2010). Mobile distributed real time database systems: A research challenges. *IEEE International Conference on Computer and Communication Technology (ICCCT)*, pp. 421–424
22. Warren, W. (2022). 9 Attributes of Live Real Time Databases [Online]. Available: https://raima.com/live-real-time-databases/
23. Oracle Database Lite Documentation Library. (2010). [Online]. Available: https://docs.oracle.com/cd/E12095_01/index.htm
24. Kim, Y., & Son, S. (1995). Predictability and consistency in real-time database systems. *Advances in real-time systems*, pp.509–531
25. Baruah, S. (2019). Mixed-Criticality Uniprocessor Scheduling. In Y. C. Tian, & D. Levy (Eds.), *Handbook of Real-Time Computing*. Singapore: Springer
26. Yu, P. S., Wu, K., Lin, K., & Son, S. H. (1994). On Real-Time Databases: Concurrency Control and Scheduling. *Proceedings of the IEEE*, vol. 82, no. 01, pp. 140–157
27. Haritsa, J., Livny, M., & Carey, M. (1991). Earliest deadline scheduling for real-time database systems. *Proceedings Twelfth IEEE Real-Time Systems Symposium*, pp. 232–242
28. Pang, H., Livny, M., & Carey, M. J. (1992). Transaction Scheduling in Multiclass Real-Time Database Systems. *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, p. 23–34
29. Datta, A., Mukherjee, S., Konana, P., Viguier, I., & Bajaj, A. (1996). Multiclass transaction scheduling and overload management in firm real-time database systems. *Inf Syst*, *21*(1), 29–54
30. Dogdu, E. (2006). Utilization of execution histories in scheduling real-time database transactions. *Data & Knowledge Engineering*, *57*(2), 148–178
31. Semghouni, S., Amanton, L., Sadeg, B., & Berred, A. (2007). On new scheduling policy for the improvement of firm RTDBSs performances. *Data & Knowledge Engineering*, *63*(2), 414–432
32. Kaddes, M., Amanton, L., Berred, A., Sadeg, B., & Abdouli, M. (2013). Enhancement of Generalized Earliest Deadline First Policy. In *Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS)*, pp. 231–238
33. Kaddes, M., Abdouli, M., Amanton, L., Sadeg, B., Berred, A., & Bouaziz, R. (2020). A probabilistic analysis of transactions success ratio in real-time databases. *International Journal of Computer Aided Engineering and Technology*, *12*(4), 405–422
34. Hong, D., Johnson, T., & Chakravarthy, S. (1993). Real-time transaction scheduling: a cost conscious approach. *ACM SIGMOD Record 22*(2), 197–206
35. Shanker, U., Misra, M., & Sarje, A. (2006). Some performance issues in distributed real-time database systems. *Proc. VLDB Ph.D. Work,Conv. Exhib. Cent. (COEX), Seoul, Korea*
36. Shanker, U. (2008). Some Performance Issues in Distributed Real Time Database Systems. PhD Thesis. Indian Institute of Technology Roorkee

37. Pandey, S., & Shanker, U. (2018). Priority Inversion in DRTDBS: Challenges and Resolutions. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (CoDS-COMAD '18)*, pp. 305–309
38. Pandey, S. (2020). Resolving Conflicts amongst Distributed Real Time Transactions, PhD Thesis, Dept. of CSE, M. M. M. University of Technology, Gorakhpur-273010, 2016-20, June 12.
39. Kao, B., & Garcia-Molina, H. (1997). Deadline assignment in a distributed soft real-time system. *IEEE transactions on parallel and distributed systems*, *8*(12), 1268–1274
40. Lee, V., Lam, K., & Kao, B. (1999). Priority scheduling of transactions in distributed real-time databases. *Real-Time Systems*, *16*(1), 31–62
41. Shanker, U., Misra, M., & Sarje, A. K. (2005). Priority assignment heuristic to cohorts executing in parallel. *Proceedings of the 9th WSEAS International Conference on Computers, World Scientific and Engineering Academy and Society (WSEAS)*, pp. 01–06
42. Shanker, U., Misra, M., & Sarje, A. K. (2005). Priority Assignment Heuristic and Issue of Fairness to Cohorts Executing in Parallel. *WSEAS Transactions on COMPUTERS*, *4*(7), 758–768
43. Chen, H. R., Chin, Y. H., & Tseng, S. M. (2001). Scheduling value-based transactions in distributed real-time database systems. *In Internationa Parallel and Distributed Processing Symposium. IEEE Computer Society.*, vol. 1, pp. 978–979
44. Pandey, S., & Shanker, U. (2020). MDTF: A Most Dependent Transactions First Priority Assignment Heuristic. In Mehdi Khosrow-Pour, Ed., *Encyclopedia of Organizational Knowledge, Administration, and Technologies* (1st ed., pp. 742–756). IGI Global
45. Pandey, S., & Shanker, U. (2020). A contention aware EQS priority assignment heuristic for cohorts in DRTDBS. *.The Journal of Supercomputing 77(7),* 6629-6663
46. Lam, K., Kuo, T., Tsang, W., & Law, G. (2000). Concurrency control in mobile distributed real-time database systems. *Information Systems*, *25*(4), 261–286
47. Lee, V. C., Lam, K. W., & Kuo, T. W. (2004). Efficient validation of mobile transactions in wireless environments. *Journal of Systems and Software*, *69*, 1–2
48. Lei, X., Zhao, Y., Chen, S., & Yuan, X. (2009). Concurrency control in mobile distributed real-time database systems. *Journal of Parallel and Distributed Computing*, *69*(10), 866–876
49. Singh, P. K., & Shanker, U. (2017). Priority Heuristic in Mobile Distributed Real Time Database Using Optimistic Concurrency Control. *23RD IEEE Annual International Conference in Advanced Computing and Communications (ADCOM), Bangalore, India*, pp. 44–49
50. Singh, P. K., & Shanker, U. (2018). A New Priority Heuristic Suitable in Mobile Distributed Real Time Database System. *In International Conference on Distributed Computing and Internet Technology. Springer.* pp. 330–335
51. Singh, P. K., & Shanker, U. (2018). A priority heuristic policy in mobile distributed real-time database system. *Advances in data and information sciences* (pp. 211–221). Singapore: Springer
52. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58
53. Grossman, R. L. (2009). The case for cloud computing. *IT professional*, *11*(2), 23–27
54. Wu, M. Y., & Gajski, D. D. (1990). Hypertool: A programming aid for message-passing systems. *IEEE transactions on parallel and distributed systems*, *1*(3), 330–343
55. Kwok, Y. K., & Ahmad, I. (1996). Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE transactions on parallel and distributed systems*, *7*(5), 506–521
56. Sih, G. C., & Lee, E. A. (1993). A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE transactions on Parallel and Distributed systems*, *4*(2), 175–187
57. El-Rewini, H., & Lewis, T. G. (1990). Scheduling parallel program tasks onto arbitrary target machines. *Journal of parallel and Distributed Computing*, *9*(2), 138–153
58. Kruatrachue, B., & Lewis, T. (1988). Grain size determination for parallel processing. *IEEE software*, *5*(1), 23–32
59. Hwang, J. J., Chow, Y. C., Anger, F. D., & Lee, C. Y. (1989). Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, *18*(2), 244–257
60. Kim, S. J. (1988). A general approach to mapping of parallel computations upon multiprocessor architectures. In *Proc. International Conference on Parallel Processing. IEEE Computer Society*, vol. 3
61. Yang, T., & Gerasoulis, A. (1994). Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, *5*(9), 951–967
62. Liou, J. C., & Palis, M. A. (1996). An efficient task clustering heuristic for scheduling dags on multiprocessors. In *Workshop on resource management, symposium on parallel and distributed processing*, pp. 152–156

63. Ahmad, I., & Kwok, Y. (1994). A new approach to scheduling parallel programs using task duplication. In *IEEE Internatonal Conference on Parallel Processing*, vol. 2, pp. 47–51
64. Hou, E. S., Ansari, N., & Ren, H. (1994). A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed systems*, *5*(2), 113–120
65. Correa, R. C., Ferreira, A., & Rebreyend, P. (1996). Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, pp. 462–469
66. Braun, T. D., Siegal, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J., Theys, M., Yao, B., Hensgen, D., & Freund, R. (1999). A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *IEEE Proceedings of the Eighth Heterogeneous Computing Workshop*, pp. 15–29
67. Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, *13*(3), 260–274
68. Baskiyar, S., & Dickinson, C. (2005). Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. *Journal of Parallel and Distributed Computing*, *65*(8), 911–921
69. Daoud, M. I., & Kharma, N. (2008). A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and distributed computing*, *68*(4), 399–409
70. Lee, Y. C., & Zomaya, A. Y. (2011). Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems*, *22*(8), 1374–1381
71. Li, Z., Ge, J., Hu, H., Song, W. H. H., & Luo, B. (2015). Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. *IEEE Transactions on Services Computing*, *11*(4), 713–726
72. Samadi, Y., Zbakh, M., & Tadonki, C. (2018). E-HEFT: enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing. In *IEEE International Conference on High Performance Computing* & *Simulation*, pp. 601–609
73. Dubey, K., Kumar, M., & Sharma, S. C. (2018). Modified HEFT algorithm for task scheduling in cloud environment. *Procedia Computer Science*, *125*, 725–732
74. Zhou, J., Zhang, M., Sun, J., Wang, T., Zhou, X., & Hu, S. (2020). Drheft: Deadline-constrained reliability-aware heft algorithm for real-time heterogeneous mpsoc systems. *IEEE Transactions on Reliability, 71*(1)*,* 178-189
75. Xu, Y., Li, K., Hu, J., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, *270*, 255–287
76. Abdelkader, D. M., & Omara, F. (2012). Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. *Egyptian Informatics Journal*, *13*(2), 135–145
77. Kumar, M., Sharma, S. C., Goel, A., & Singh, S. P. (2019). A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, *143*, 1–33
78. Maurya, A., & Tripathi, A. (2018). On benchmarking task scheduling algorithms for heterogeneous computing systems. *The Journal of Supercomputing*, *74*(7), 3039–3070

Dr. Sarvesh Pandey is presently Assistant Professor in the Computer Science - MMV, BHU, Varanasi, India. He received his Ph.D. degree (2020) in Computer Science & Engineering from M. M. M. University of Technology, Gorakhpur-273010, India. His broad areas of research include distributed real-time database systems, cloud computing and advanced data systems. He has published more than 25 research papers in various journals/conferences. He is active review member of various reputed journals, conferences, and book series.

Dr. Udai Shanker is presently Professor in the Department of Computer Sc. & Engineering of M. M. M. University of Technology, Gorakhpur-273010. For his imitation of the most modern of approaches and also for his exemplary devotion to the field of teaching and sharing his profound knowledge with students to make better future citizen of India, he has been a role model for the new generation of academicians. Besides introduced radical and revolutionary changes that have positively impacted the database world and student community, he is a man well versed with all the intricacies of academics.