A comparative analysis of resource allocation schemes for real-time services in high-performance computing systems

Muhammad Shuaib Qureshi¹,², Muhammad Bilal Qureshi³, Muhammad Fayaz², Wali Khan Mashwani⁴, Samir Brahim Belhaouari⁵, Saima Hassan⁶ and Asadullah Shah¹

Abstract
An efficient resource allocation scheme plays a vital role in scheduling applications on high-performance computing resources in order to achieve desired level of service. The major part of the existing literature on resource allocation is covered by the real-time services having timing constraints as primary parameter. Resource allocation schemes for the real-time services have been designed with various architectures (static, dynamic, centralized, or distributed) and quality of service criteria (cost efficiency, completion time minimization, energy efficiency, and memory optimization). In this analysis, numerous resource allocation schemes for real-time services in various high-performance computing (distributed and non-distributed) domains have been studied and compared on the basis of common parameters such as application type, operational environment, optimization goal, architecture, system size, resource type, optimality, simulation tool, comparison technique, and input data. The basic aim of this study is to provide a consolidated platform to the researchers working on scheduling and allocating high-performance computing resources to the real-time services. This work comprehensively discusses, integrates, analysis, and categorizes all resource allocation schemes for real-time services into five high-performance computing classes: grid, cloud, edge, fog, and multicore computing systems. The workflow representations of the studied schemes help the readers in understanding basic working and architectures of these mechanisms in order to investigate further research gaps.

Keywords
High-performance computing, resource allocation scheme, real-time systems, cloud computing, fog computing, edge computing, grid computing

Introduction
The high-performance computing (HPC) paradigm paves the way to the large number of resources with high computing power and storage capabilities distributed across a network. The HPC platform is focused by the research community, mainly for three basic reasons: (a) the parallel nature of applications, (b) the need of data distribution and communication with

¹Kulliyyah of Information & Communication Technology (KICT), International Islamic University, Kuala Lumpur, Malaysia
²Department of Computer Science, School of Arts and Sciences, University of Central Asia, Naryn, Kyrgyz Republic
³Department of Computer Science, Shaheed Zulfikar Ali Bhutto Institute of Science and Technology, Islamabad, Pakistan
⁴Institute of Numerical Sciences, Kohat University of Science & Technology, Kohat, Pakistan
⁵Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar
⁶Institute of Computing, Kohat University of Science & Technology, Kohat, Pakistan

Corresponding author:
Samir Brahim Belhaouari, Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Education City, Qatar Foundation, Doha 34110, Qatar.
Email: sbelhaouari@hbku.edu.qa
other nodes through communication channels, and (c) the need for continuous availability and reliability of the underlying resources. These facilities motivate the users toward using distributed HPC systems for scalability, reliability, availability, exchanging and sharing information, and achieving performance efficiency with low cost and high quality.1–5 Organizations and individuals increasingly generate and store huge amounts of data on a daily basis using Internet of Things (IoT) devices by executing different nature applications. Such applications need powerful resources for computation and storage. Due to the limited number of available resources, proper resource allocation (RA) schemes are used to cater with all types of applications. RA scheme in the HPC environment plays a vital role in managing limited resources among multiple competing applications in a fair way that guarantees providing agreed quality of service (QoS).6,7 In HPC platform, a resource is any computing, storage, or communication entity that takes part in user application execution. The RA schemes consider different performance parameters based on the nature of applications. Some of the parameters are makespan minimization, computation cost minimization, energy efficiency, bandwidth optimization, and so on. These are some of the common parameters that are considered by most computational-intensive applications. However, some applications prefer to complete execution within specific time duration. Such applications are attached a time parameter, commonly known as execution deadline. These types of applications are called as real-time (RT) applications. RT applications or services are bounded by time constraints. Executing such applications not merely depend on producing the correct output but the time at which the results are generated.8,9 We use an application, service, task, and system interchangeably in this article. Some common examples of RT applications can be found in chemical plants, robotics, anti-missile systems, pacemakers, multimedia systems, and embedded systems.10–12

**RT application model**

Consider a RT application \( T \) as a set of multiple tasks. Assume there are \( n \) number of tasks in \( T \) and each task is characterized by the following typical parameters as shown in Figure 1.

1. **Deadline**—the time instant at or before which the task must complete its execution. In RT applications, mainly two types of deadlines are used: absolute and relative deadlines. The absolute deadline is a time instant which must be followed by RT task for producing timely correct results. On the contrary, relative deadline is the time duration between the absolute deadline and the release time of a task.

   \[
   \text{Relative deadline} = \frac{\text{Absolute deadline}}{C_0} - \text{Release time}
   \]

2. **Release time**—the arrival time of a task in the ready queue.
3. **Execution time**—this is the worst-case computation time of a task for which a computational resource is required without any interrupt.
4. **Criticality**—the consequences of missing the deadline. It can be hard, soft, or firm.

![Figure 1. Real-time application taxonomy.](image-url)
5. Response time—the difference between the time when a request for the resource is made and the finishing time on the resource when it is allocated.

Apart from the aforementioned parameters, the RT tasks can be distinguished on the basis of activation times. If the instances (jobs) of a task are activated at a constant rate, then it is called as periodic task. On the other hand, if the jobs activations of a task are not regularly interleaved, then the task is called as aperiodic task.

The focus of RA schemes for RT services is to complete execution within specified time. On the basis of timing constraints, RT services are broadly categorized into soft and hard RT services. In hard RT services, there is no room for missing deadline while in soft RT services, deadline missing puts drastic effects on overall system performance.²⁻⁴,¹³,¹⁴

Before explaining the RA schemes for RT services in detail, Table 1 describes the terms used in this article.

**Brief comparison with the existing surveys**

The existing state-of-the-art literature shows surveys on RA for different nature applications in HPC domains. But as per our exploration, no-one has consolidated RA schemes in all the emerging HPC paradigms like grid, cloud, fog, edge, and multicore systems in a single platform for special-type applications, that is, RT services. Naha et al.¹⁵ surveyed RA techniques for IoT devices generating RT data. They have considered latency parameters only and analyzed their applications in fog computing environment. They have confined their study mainly to different fog computing aspects and definitions. W Shu et al.¹⁶ provided an overview of RA in edge computing by presenting case studies ranging from smart homes to smart cities, RT video analysis, augmented reality, healthcare monitoring, and virtual reality games. Hridita et al.¹⁷ surveyed mobility aware RA and scheduling algorithms. They have overviewed heuristic approaches to balance non-RT applications makespan and cloud resources monetary costs. Mao et al.¹⁸ surveyed RA techniques in mobile edge computing. They have considered the communication perspective of the intended applications. They have identified limitations like high infrastructure deployment and maintenance cost, changeable human activity interaction, and so on. Mangla et al.¹⁹ elaborated resource scheduling and allocation schemes in a cloud computing environment. They have evaluated RA schemes on the basis of administrative domains, virtual machine (VM) allocation and migration strategies, energy efficiency, service level agreements (SLAs), and cost-effectiveness. RT periodic systems were analyzed for schedulability on multicore systems in previous studies.²⁰⁻²² The authors identified basic schedulability parameters by utilizing static priority assignment algorithms. Energy-aware RA schemes in a cloud computing environment were studied by Beloglazov et al.²³ They have evaluated heuristics for efficient management of cloud data centers. The RA mechanisms for different non-RT applications on grid computing systems have been detailed by MB Qureshi et al.⁴ They have explained each mechanism in detail, but their study is limited to grid computing systems only. Another comprehensive survey on RA in distributed computing systems is given by H Hussain et al.² They have studied the RA problem for mixed applications with different QoS parameters.

| Table 1. Terms description. |
|----------------------------|
| Term          | Description                                      |
|----------------|--------------------------------------------------|
| HPC            | High-performance computing                        |
| QoS            | Quality of service                                |
| RA             | Resource allocation                               |
| DVS            | Dynamic voltage scaling                           |
| RDTA           | Real-time data-intensive tasks allocation         |
| GA-SS          | Steady state genetic algorithm                    |
| GA-ST          | Struggled genetic algorithm                       |
| GA-EG          | Elitist generational genetic algorithm            |
| PPDP            | Partition problem-based dynamic provisioning and scheduling |
| DSB            | Dynamic scheduling of bag of tasks               |
| MAHP           | Modified analytic hierarchy process               |
| BATS           | Bandwidth aware tasks scheduling                 |
| LEPT           | Longest expected processing time                  |
| AGA            | Adaptive genetic algorithm                        |
| EFTD           | Earliest finish time duplication                  |
| PRS            | Proactive and reactive scheduling                 |
| ATS            | Allocation-aware task scheduling                  |
| DGP            | Developmental genetic programming                 |
| DCLS           | Dynamic cloud list scheduling                     |
| DCMMS          | Dynamic Cloud Min-Min Scheduling                  |
| EDF            | Earliest deadline first                           |
| DVFS           | Dynamic voltage frequency scaling                 |
| HGCN           | Hybrid genetic and cuckoo search                  |
| PTPS           | Purely time-driven periodic server               |
| WCPS           | Work-conserving periodic server                   |
| CRPS           | Capacity reclaiming periodic server               |
| USG            | Unfair-semi greedy                                |
| EDZL           | Earliest deadline until zero-laxity              |
| EARH           | Energy aware rolling horizon                      |
| GCSM           | Green cloud scheduling model                      |
| FDHEFT          | Fuzzy dominance sort–based heterogeneous          |
| DA-EDF         | Data-aware earliest deadline first                |
| EDA-EDF        | Enhanced data-aware earliest deadline first       |
| RS             | Resource scheduling                               |
| RM             | Resource matching/rate monotonic                  |
| HCS            | Hybrid cuckoo search                              |
| LFS            | Least feasible speed                              |
| CATP           | Compatibility aware task partition                |
| G-CATP         | Group-wise compatibility aware task partition     |
| SCRUP          | Simple combined resource usage partitioning       |
| ENCAP          | Enhancing shared cache performance                |
| TDA            | Time demand analysis                              |
| DEFT           | Dynamic fault-tolerant elastic                    |
How is this survey different from the existing state-of-the-art surveys?

The following contributions highlight the novelty of this article:

1. The existing literature on RA shows techniques for mixed RT and non-RT applications on one or another distributed computing systems. There exists no comprehensive state-of-the-art comparative analysis conducted only for RT applications. This article gathers and compares RA schemes only for RT applications on HPC (grid, cloud, fog, edge, and multicore) systems.

2. The existing research gives a plethora of RA schemes by considering certain parameters which may not give proper comparison with respect to all aspects. Instead, in this article, the RA schemes are compared on the basis of the most common parameters that cover almost all of the features of all RA schemes.

3. This article surveys and compares RT applications on both distributed (grid, cloud, edge, and fog computing) and non-distributed (multicore) HPC platforms, while current surveys are conducted for only one or another HPC environment.

4. This article evaluates and portrays each RA scheme graphically in a convinced way that is easy to understand.

5. This survey can assist the research and development stakeholders in synthesizing and identifying their requirements for different emerging HPC paradigms encompassing different architectures and implementations. The target readers can be categorized into architecture-interested readers, algorithm-interested readers, and general readers.

**Article organization**

The article structure is pictorially presented in Figure 2. Section “Introduction” introduces basics of the HPC systems, the RT application model, brief comparison with the existing surveys, and overall article organization. This section also portrays RT application taxonomy. Section “HPC RA problem for RT services” evaluates RA problem for RT services. The criteria are provided for evaluating the RA schemes. Section “HPC RA problem for RT services” also shows a broad taxonomy of RA schemes for RT services on HPC (grid, cloud, fog, edge, and multicore) systems. Section “RA schemes for RT services in grid computing” demonstrates RA schemes for RT services in grid computing systems. Section “RA schemes for RT services in cloud computing” focuses on describing RA schemes in cloud computing systems. These systems are explained for accommodating RT applications. We review the edge computing resources allocation schemes for RT services in section “RA schemes for RT services in edge computing.” Section “RA schemes for RT services in fog computing” details fog computing RA mechanisms for executing RT applications, while section “RA schemes for RT services in multicore systems” presents a multicore environment for such type applications. Finally, section “Conclusion” concludes this survey. All of the RA mechanisms are presented pictorially which can help in understanding their working easily.

**HPC RA problem for RT services**

RA problem in HPC systems can be defined as an issue of assigning limited HPC resources in order to satisfy the performance requirements of the competing applications according to some predefined QoS criterion such as makespan minimization, profit maximization, cost and energy efficiency, load balancing, and deadline satisfaction.

The RT application T is composed of multiple tasks \( t_i \) (1 < i < n), where each task is characterized by a completion deadline \( d_i \). The T is mapped to a set of HPC resources \( J = \{ j_1, j_2, ..., j_k \} \) according to some predetermined criteria. Then, the general RA problem of assigning task \( i \) to HPC resource \( j \) can be defined as \( \{ t_i, j, \text{deadline is satisfied} \} \).

The feasibility of the RA scheme for RT application T to HPC resources J can be defined as a function

\[
RA(T) = \begin{cases} 
\text{feasible, each task deadline is fulfilled} & \text{if feasible} \\
\text{infeasible, otherwise} & \text{otherwise}
\end{cases}
\]

The HPC resource can be defined as a machine with different capabilities. The capabilities may be processing, storage, or communication, and so on. The machines with processing and communication capabilities are known as computing and network resources, respectively. Some of the HPC resources have both computing and storage capabilities. These resources have associated cost of processing. The allocation cost is calculated from the resource binding process that analyses the resource performance, architecture, utilization, and processing power. Based on the architecture, HPC resources can be classified into homogeneous or heterogeneous resources. Homogeneous resources have the same, while heterogeneous resources have different designing architectures. In case of heterogeneous resources, task allocation is performed by analyzing the allocation cost.

Figure 3 shows a broad taxonomy of RA schemes for RT systems on the aforementioned HPC platforms.
Evaluation criteria

The existing RA schemes for scheduling RT applications on HPC resources shown in Figure 3 can be compared and evaluated on the basis of some common parameters. The set of criteria we consider here for evaluating the work done so far include application type, operational environment, optimization goal, architecture, system size, resource type, optimality, simulation tool, comparison technique, and input data. We discuss the aforementioned aspects briefly in the following subsections and examine in detail in each RA scheme in Table 2.

Application type. The HPC paradigm supports various types of applications, and the RA schemes are modeled based on the application type requirements. Some of the application types include RT applications, deadlines oriented dependent applications, data-intensive RT dependent applications, workflow applications, and so on. In this survey, we only focus on analyzing RA schemes which are developed for RT applications. Some of the RT applications may be data intensive, which need data for complete processing. The dependent applications can be broadly categorized into workflow applications which consist of multiple divisible sub-applications with some sort of execution or data dependency. In case of dependency, there is communication between applications which use network bandwidth.

Operational environment. The environment consisting of executing resources on which the RA scheme is operated can be termed as operational environment. It can
be categorized on the basis of nature of resources and their operational taxonomies, for example, distributed systems, grid computing, cluster computing, cloud computing and multicore systems. These systems can be accumulated under the umbrella of HPC environment. The HPC environment can be broadly categorized into distributed HPC systems which include grid, cloud, edge, and fog computing systems, and non-distributed systems which consist of multicore systems.

**Optimization goal.** Every RA scheme is designed to achieve specific metric on the basis of which its performance is measured, for example, energy consumption,
| RA technique                | Publishing year | Application type                                                                 | Operational environment        | Optimization goal                                                                 | Architecture            | System size | Resource type                  | Optimality | Simulation tool | Comparison techniques | Input data          |
|-----------------------------|-----------------|----------------------------------------------------------------------------------|--------------------------------|------------------------------------------------------------------------------------|-------------------------|-------------|-------------------------------|------------|----------------|----------------------|---------------------|
| HGCS24                      | 2019            | Real-time data-intensive tasks                                                   | Cloud computing               | Makespan minimization and cost efficiency                                        | Heterogeneous           | 10, 20      | Computing, storage, bandwidth | NA         | MATLAB | GA, CS                           | Synthetic dataset   |
| Task Buffering and Offloading Policy25 | 2019         | Real-time tasks                                                                 | Fog computing                 | Maximizing throughput and tasks completion ratio, RA balancing                     | Heterogeneous           | Variant     | Computing, memory             | NA         | CloudSim | RR, MRU, AQW, FCFS, LTS, MOMIS, NMOMIS | Not mentioned       |
| USG, EDZL26                 | 2019            | Soft real-time tasks                                                            | Cloud computingSys            | Feasibility analysis, response time analysis                                        | Heterogeneous           | 4–64 nodes | Computing                       | NA         | Not mentioned | EDF                  | 10,000 synthetic tasksets Random synthetic workload |
| EDF-DVFS-AC27               | 2019            | Real-time workflow system                                                      | Cloud computing system        | Cost and energy efficiency, tasks feasibility analysis, SLA violation ratio         | Heterogeneous           | Not mentioned | Computing, communication      | NA         | C++          | EDF, EDF-DVFS                  |                     |
| FDHEFT28                    | 2019            | Deadline-constrained workflow applications                                      | IaaS Cloud                    | Cost and makespan minimization                                                    | Heterogeneous           | Not mentioned | Computing                       | NA         | JMetal | MOHEFT, NSPAS, SPEA2, ß-fuzzy PSO, RSH, GCP for RS and RMH, D-LAWS for RM | Real-world and synthetic workflows Dataset of SNAP |
| RS and RM29                 | 2019            | Latency-critical computation-intensive applications                              | Cloud and edge computing      | Network delay, user QoS                                                         | Heterogeneous           | 1 edge orchestrator and 10 ES | Computing, communication, storage | Yes        | Hadoop 2.7.1 consisting of YARN and MapReduce | ρ, ζ, σ |                     |
| ENCAP30                     | 2019            | Real-time tasks                                                                 | Multicore systems             | Tasks schedulability analysis                                                      | Heterogeneous           | 2–32 cores | Computing, communication       | NA         | OPENMP, EDF-EN | Calandrino, Bastoni, Anderson dataset Synthetic dataset | Cybershake, Montage, Epigenomics, Inspiral, SIPHT |
| Lg-TDA31                    | 2019            | Real-time systems                                                               | Multicore system              | Task schedulability analysis                                                      | Not mentioned           | Not mentioned | Computing                       | NA         | MATLAB | TDA                               |                     |
| PPDPS, K-mean32             | 2018            | Deadline-constrained dependent data-intensive tasks                             | Cloud computing system        | Deadline-constrained cost efficiency                                            | Heterogeneous           | Not mentioned | Computing                       | NA         | ANOVA | DPDS, IC-PCP                  | Cybershake, Epigenomics, LIGO Inspiral Analysis, Montage, SIPHT |
| DSB33                       | 2018            | Scientific workflows                                                            | Cloud computing system        | Cost efficiency, deadline meeting                                                | Heterogeneous           | 16          | Computing                       | NA         | WorkflowSim | WRPS, SCS, HEFT                 |                     |
| MAHP, BATS, BAR, LEPT, divide-and-conquer34 | 2018 | Scientific workflows                                                            | Cloud computing system        | Turnaround time, response time                                                  | Heterogeneous           | 20          | Computing, memory, bandwidth    | NA         | Real cloud environment | BATS, IDEA             |                     |

(continued)
| RA technique | Publishing year | Application type | Operational environment | Optimization goal | Architecture | System size | Resource type | Optimality | Simulation tool | Comparison techniques | Input data |
|--------------|-----------------|------------------|-------------------------|------------------|--------------|------------|--------------|------------|-----------------|---------------------|------------|
| Data-aware   | 2018            | Deadline-constrained bag-of-tasks applications with data requirements | Hybrid cloud computing system | User specific deadlines meeting, execution time | Heterogeneous | 19         | Computing, storage, bandwidth | NA         | Aneka           | Default and enhanced algorithms | Walliability index application |
| SCRUP       | 2018            | Hard real-time systems | Multicore system | Feasibility analysis and minimizing power dissipation | Homogeneous | 4, 16      | Computing, memory | NA         | Not mentioned   | HTTP, TATP           | 3000 and 12,000 synthetic datasets |
| Hybrid-EDF  | 2018            | Real-time IoT workflows | Fog and cloud computing systems | Deadline miss ratio, percent tasks executed on cloud, total momentary cost | Heterogeneous | 3 fog hosts and 30 cloud hosts | Computing, communication | NA         | C++             | Fog-EDF              | Synthetic workload |
| DEFT         | 2018            | Real-time applications | Cloud computing systems | Fault-tolerance, resource utilization efficiency maximization, no. of completed tasks | Heterogeneous | Not mentioned | Computing | NA         | CloudSim       | NDRFT, DRFT, NWDEFT | Google traces |
| RDTA         | 2017            | Data-intensive real-time tasks | Grid computing system | Makespan minimization | Heterogeneous | 11, 34     | Computing, storage, bandwidth | NA         | MATLAB          | PTA                 | Synthetic dataset |
| AGA          | 2017            | Hard real-time tasks | Cloud computing system | Execution time and cost efficiency | Heterogeneous | 30         | Computing | NA         | Wilcoxon-ranked-sum test | Greedy, GA | DAG's of varying sizes Random synthetic workload |
| MinMinMAC, MinMinEAC, MaxMinMAC, MaxMinEAC, SuffrageMAC, SuffrageEAC | 2017 | Real-time bag of tasks | SaaS cloud computing system | SLA violation ratio, average result precision, average cost per job | Heterogeneous | 128        | Computing | NA         | MinMin, MaxMin, Suffrage | Synthetic workload |
| DA-EDF, EDA-EDF | 2017 | Real-time data-intensive BoT applications | SaaS Cloud | Impact of data locality in terms of SLA violation percentage | Heterogeneous | 256 VMs   | Computing, storage | NA         | C++             | NDA-EDF              | Synthetic workload |
| SigLM-MMW    | 2016            | Real-time multi-workflows | Cloud computing system | Load balancing, average waiting time | Heterogeneous | Arbitrary  | Computing | NA         | CloudSim       | Not mentioned         | Real load traces collected on PlanetLab |
| RA technique                  | Publishing year | Application type            | Operational environment | Optimization goal                                                                 | Architecture | System size | Resource type | Optimality | Simulation tool | Input data                                                                 |
|------------------------------|-----------------|-----------------------------|-------------------------|----------------------------------------------------------------------------------|--------------|-------------|---------------|------------|----------------|-----------------------------------------------------------------------------|
| GCSM                         | 2016            | Deadline-constrained tasks  | Cloud computing system  | Energy efficiency, maximizing performance ratio                                   | Heterogeneous | 25–50       | Computing     | NA         | Not mentioned  | FCFS, SLA-based resource constraint VM scheduling, priority-based scheduling scheme. EDF, EDF-RAC, EDF-FAC, EDF-ADC, EDF-ADC-RAC, EDF-ADC-FAC. |
| Gang scheduling approach     | 2016            | Real-time parallel applications | SaaS Cloud               | Deadlines meeting, high quality results, total momentary cost, tackling with software failures | Homogeneous  | 64          | Computing, communication | NA        | C++            | Synthetic workload. Real-time workload traces available at Parallel Workload Archive. |
| RM with reduced priority levels | 2015            | Real-time tasks             | Multicore system        | Priority levels reduction, cumulative utilization                                  | Heterogeneous | 12 cores CPU | Computing     | NA        | MATLAB         | Traditional approach with scheduling points, Synthetic dataset.           |
| EFTD                         | 2015            | DAG tasks                   | Cloud computing system  | Normalized scheduling length (NSL) reduction, computational cost and time minimization | Heterogeneous | 10–130      | Computing     | NA        | CloudSim       | HEFTAEST, HEFT-TL, DAG tasks.                                              |
| ATS                          | 2015            | User requests               | Multi-cloud systems     | Makespan minimization, average cloud utilization                                    | Heterogeneous | 32          | Computing     | NA        | MATLAB         | RR, CLS. Two benchmark dataset consists of 512 tasks and a synthetic dataset. |
| Bandwidth-aware              | 2015            | Workflow                    | Cloud computing system  | Bandwidth efficiency, execution time                                               | Heterogeneous | Not mentioned | Computing     | NA        | Eclipse        | Not mentioned. Synthetic dataset.                                           |
| CATP, G-CATP                 | 2015            | Hard real-time tasks        | Multicore system        | Improving system schedulability under failures                                     | Heterogeneous | 4, 8 cores  | Computing     | NA        | Multicore system | HAPS, BFD. Workflow consisting of 4 tasks Task sets generated using UUniFast algorithm. |
| RA technique | Publishing year | Application type | Operational environment | Optimization goal | Architecture | System size | Resource type | Optimality | Simulation tool | Comparison techniques | Input data |
|--------------|-----------------|------------------|-------------------------|-------------------|--------------|-------------|--------------|------------|----------------|-----------------------|------------|
| EDF-BF-IC52  | 2015            | Real-time workflow | Paas and IaaS clouds    | Deadlines, cost efficiency, execution time minimization | Heterogeneous | 64          | Computing    | NA         | C++           | EDF                  | Real-world workflow application |
| PRS93        | 2015            | Real-time tasks   | Cloud computing system  | Energy efficiency, improving resource utilization, minimizing execution time | Heterogeneous | 5           | Computing    | NA         | Apache CloudStack 4.2.0, CloudSim | NMPRS, EDF, MCT, CRS | II-app, CloudSim-app |
| DVS54        | 2014            | Independent batch scheduling | Grid computing system | Makespan, flow time and energy efficiency | Heterogeneous | 64, 256    | Computing    | NA         | Sim-G-Batch Grid Simulator | GA, HGS-Sched, IGA | Kiviat Graphs, 1024 and 4096 tasks randomly generated by Gaussian distributions |
| DGP55        | 2014            | Soft real-time tasks | Cloud computing system | Cost optimization | Heterogeneous | 6, 4        | Computing, communication | NA         | Not mentioned | Not mentioned | Adaptive navigation system converted into TGs |
| PTPS, WCPS, CRPS56 | 2014 | Real-time applications | Cloud computing system | Calculating deadlines miss ratio and number of context switches | Heterogeneous | 6 cores    | Computing, memory | NA         | RT-Xen 2.0, Fedora 13 with para-virtualized kernel 2.6.32.25 | pEDF, gEDF, pDM, gDM | Synthetic workloads |
| PRS93        | 2014            | Real-time tasks   | Cloud computing system  | Energy efficiency | Heterogeneous | Infinite    | Computing    | NA         | Apache CloudStack 4.2.0, CloudSim | NMPRS, EDF, MCT, CRS | II-app, CloudSim-app |
| EArH57       | 2014            | Real-time application | Cloud computing system | Energy efficiency and tasks feasibility analysis | Heterogeneous | Infinite    | Computing    | NA         | NRHEARH, NMNEARH, NRHMRAARH, ProFRS, Greedy-R, Greedy-P, FCFS | NRHMRARH, ProfRS, Greedy-P, FCFS | Random synthetic tasks, Google cloud tracelogs |
| HCS58        | 2013            | Flow shop scheduling problem | Distributed non-HPC systems | Makespan minimization | Heterogeneous | Not mentioned | Computing    | NA         | MATLAB       | CS, ATTPSO, L-CPSO, HDE, OSA, PSOMA, PSOVNS, HGA, BEST (LR), MMAS, QDEA | 160 problems from OR library (8 = car1–car8 instances, 21 = rec01–rec41 instances, 120 instances, 11 = DMU instances) | (continued) |
### Table 2. Continued

| RA technique                                      | Publishing year | Application type | Operational environment                  | Optimization goal                                      | Architecture                  | System size                  | Resource type | Optimality | Simulation tool | Comparison techniques | Input data                      |
|---------------------------------------------------|-----------------|------------------|------------------------------------------|--------------------------------------------------------|-------------------------------|-----------------------------|----------------|------------|------------------|--------------------------|--------------------------------|
| Task shifting and task splitting\(^{59}\)          | 2013            | Real-time tasks  | DAG                                      | Load balancing                                         | Heterogeneous                 | 4 and 5 cores               | Computing      | NA         | MATLAB           | Not mentioned             | Synthetic dataset            |
| DCLS, DCMMS\(^{60}\)                              | 2012            | Independent      | Grid computing system                    | Makespan and energy optimization, dynamic load balancing | Heterogeneous                 | 64, 128, 256             | DVS-enabled computing | NA         | HyperSim-G         | FFS                      | Synthetic dataset using Gaussian distributions |
| GA-SS, GA-ST, GA-EG\(^{61}\)                       | 2012            | Independent      | Grid computing system                    | Cost efficiency                                        | Heterogeneous                 | Not mentioned              | Computing      | NA         | Not mentioned     | EDF, GUS, PP, Risk/Reward, PPOC, PPS | Randomly generated 1000 task sets |
| LFS\(^{62}\)                                      | 2012            | Real-time tasks  | Multicore system                         | Maximizing total utility                               | Heterogeneous                 | Not mentioned              | Computing      | NA         | Not mentioned     | Min-Min, RC TS            | Synthetic dataset            |
| Online accrued scheduling\(^{63}\)                 | 2011            | Real-time tasks  | Multicore system                         | Maximizing total utility                               | Heterogeneous                 | Not mentioned              | Computing      | NA         | Not mentioned     | EDF                      | Synthetic dataset            |
| EDF-greedy\(^{64}\)                               | 2011            | Real-time application | IaaS cloud computing system               | Cost efficiency                                        | Heterogeneous                 | Not mentioned              | Computing      | NA         | Not mentioned     | FFS                      | Synthetic dataset            |
| DVFS\(^{65}\)                                      | 2011            | Real-time application | Cloud computing system                    | Power efficiency                                       | Heterogeneous                 | 4                          | Computing      | NA         | CloudSim          | Not mentioned             | Synthetic dataset            |

HPC: high-performance computing; RA: resource allocation; HGCS: hybrid genetic and cuckoo search; GA: genetic algorithm; RR: round robin; USG: unfair-semi greedy; EDZL: earliest deadline until zero-laxity; EDF: earliest deadline first; EDF-DVFS: earliest deadline first-dynamic voltage frequency scaling; FDHEFT: fuzzy dominance sort–based heterogeneous earliest finish time; RS: resource scheduling; RM: resource matching/rate monotonic; ENCAP: enhancing shared cache performance; TDA: time demand analysis; PPDP: partition problem-based dynamic provisioning and scheduling; ANOVA: analysis of variance; DPDS: dynamic provisioning dynamic scheduling; IC-PCP: IaaS cloud partial critical path; DSB: dynamic scheduling of bag of tasks; MAHP: modified analytic hierarchy process; BATS: bandwidth aware tasks scheduling; LEPT: longest expected processing time; SCRUP: simple combined resource usage partitioning; DEFT: dynamic fault-tolerant elastic; RDTA: real-time data-intensive tasks allocation; AGA: adaptive genetic algorithm; DA-EDF: data aware earliest deadline first; EDA-EDF: enhanced data-aware earliest deadline first; GCSM: green cloud scheduling model; VM: virtual machine; EDF-RAC: earliest deadline first with restricted approximate computations; EDF-FAC: earliest deadline first with full approximate computations; EDF-ADC: earliest deadline first with application-directed checkpoints and full approximate computations; CPU: central processing unit; EFTD: earliest finish time duplication; DAG: directed acyclic graph; CLS: cloud list scheduling; BRS: best resource selection; CATP: compatibility aware task partition; G-CATP: group-wise compatibility aware task partition; PRS: proactive and reactive scheduling; NMR-PRS: non-migration-PRS; MCT: minimum completion time; CRS: complete rescheduling; DVS: dynamic voltage scaling; DPG: developmental genetic programming; PTPS: purely time-driven periodic server; WCPS: work-conserving periodic server; CRPS: capacity reclaiming periodic server; EARH: energy aware rolling horizon; HCS: hybrid cuckoo search; DCLS: dynamic cloud list scheduling; DCMMS: Dynamic Cloud Min-Min Scheduling; LFS: least feasible speed; FFS: first feasible speed; GA-SS: steady state genetic algorithm; GA-ST: struggled genetic algorithm; GA-EG: elitist generational genetic algorithm; WRPS: workflow responsive resource provisioning and scheduling; SCS: scaling-consolidation scheduling; BAR: bandwidth-aware routing; AEST: average of earliest start time; HEFT-TL: heterogeneous earliest finish time - level; NDA-EDF: non-data aware - earliest deadline first; PSO: particle swarm optimization; NSPSO: non-dominated sort particle swarm optimization; SPEA2: strength pareto evolutionary algorithm; MOHEFT: multi-objective heterogeneous earliest finish time; EDF-BF-IC: earliest deadline first best fit with imprecise computation; CISCO: San Francisco; PSOVM: particle swarm optimization-based memetic algorithm; OSA: Osman’s simulate annealing; PSOVS: particle swarm optimization for variable neighborhood search; HDE: hybrid differential evolution; GUS: generic utility scheduling; TATP: thermal aware task partitioning.
net profit gain, makespan, response time, system throughput, system utilization, applications execution within user budget constraints, and so on. Some of these metrics should be maximized such as net profit gain, throughput, system utilization, application processing within user budget, while some should be minimized such as total energy consumption, makespan, and response time.

**Architecture.** System architecture defines the nature of resources or configuration of resources within operational environment. The resources can be homogeneous or heterogeneous. Homogeneous resources have the same configuration, and application execution on a single resource shows the same status of the application on all resources. Heterogeneous resources have different configurations. The diversity may be due to the resource power, short-term storage capacity, or instruction sets. In a heterogeneous environment, different resources give different execution statistics of applications.

**System size.** The system size shows the total number of resources in the operational environment. In distributed HPC environment, this number is the count of total computing and storage resource while in non-distributed HPC system, it is the total number of cores in the system. The RA scheme takes into account the resource types required for the nature of executing application. If the executing application needs computing and storage resources, then the RA strategy is developed by considering the processing specifications of the resources. For example, if a system is composed of limited computing resources and a huge application needs more computing resources then the RA scheme implements proper load balancing mechanism to uniformly distribute application load on available limited resources.

**Resource type.** The HPC environment is composed of different types of resources like computing, storage, and network resources. The computing resources only perform computation tasks. They cannot give storage facilities. The storage resources provide data storage capabilities. The network resources offer communication facilities and work for transferring data from one resource to another resource. The RA mechanism is developed by considering resource-type configuration, for example, the data-intensive application engages both computing and storage resources. If these resources are located far away from one another, then the RA scheme considers computing, storage, as well as network resources for execution and data exchange among nodes.

**Optimality.** The RA scheme that gives better results against performance metrics is called as optimal scheme. Every RA scheme strives to achieve the defined performance metric. Some of the RA schemes are near optimal or non-optimal. The near optimal RA schemes have acceptable results like optimal and very little margins exist for improving its performance, but in case of non-optimal, the RA scheme needs revisiting. This parameter takes values like optimal, near optimal, non-optimal, or not applicable (NA) in case if it is not mentioned in the proposed scheme.

**Simulation tool.** The RA scheme is tested using some virtualized environment that gives the reflection of the actual physical environment. The simulation tool consists of the same operational architecture like a real environment. Different simulation tools are used for implementing RA schemes depending upon the nature of executing applications. The results obtained by simulating application on virtualized environment have much closer accuracy of the results obtained on the actual physical environment.

**Comparison techniques.** This parameter reveals the validity of the proposed techniques in comparison with the other existing state-of-the-art counterparts. The results are compared, and a conclusion is drawn in the form of numerical values or graphical charts.

**Input data.** The input data attribute indicates the dataset which is considered for experimentation or simulation of a specific task scheduling and allocation scheme. In all of the above mentioned parameters, the RT application execution within specified deadline is a common criteria for all HPC RA schemes.

**RA schemes for RT services in grid computing**

Grid computing is the integration of different hardware and shared use of computing resources, that is, shared infrastructure over a network for solving complex problems. In grid computing, the data are moved among different computing resources. So, managing and running distributed workflows automatically is a core feature of the grid computing.

The core essentials of grid technology are shared heterogeneous infrastructure, support of collaboration, distributed workflow management, and secure access to shared data. The general architecture of grid computing consists of a user, grid information service, resource broker, and resources. The user sends tasks to the grid for processing to speed up the execution of the application. The grid information service is a system that
collects information about the available grid resources and sends this information to the resource broker. The resource broker distributes jobs to the available grid resources based on the user’s requirements for execution. The grid resources are the computing entities that execute the user jobs.

The general architecture and taxonomy of grid computing environment is shown in Figures 4 and 5.

The grid computing RA schemes for RT services are described as follows.

**Dynamic voltage scaling**

Joanna Kołodziej et al.\(^{54}\) addressed exertion of energy spent on scheduling in grid environment, taking into account various grid scenarios. Grid system consists of multi-layer architecture with hierarchical management system, namely, grid fabric layer, grid core middleware, grid user layer, and grid application layer. For resource management and scheduling, two user and middleware layers are important. The three features of scheduling to define the task are static environment, batch task processing, and task interrelations. The authors addressed batch scheduling in a static environment where tasks are grouped into batches and executed independently in hierarchical order. For power supply analysis, two main energy-aware scheduling scenarios max-min and power supply mode are considered. The genetic algorithm is applied to solve scheduling issues by considering its six features, namely, single population in risky mode GA(R), single population in secure mode GA(S), multi-population in risky mode HDS-Sched(R), multi-population in the secure mode HGS-Sched(S), multi-population island in risky mode IGA(R), and multi-population island in secure mode IGA(S). Finally, single and multi-population are compared using empirical analysis in grid environment. The high-level scenario is shown in Figure 6.
RT data-intensive tasks allocation technique

In real-time data-intensive tasks allocation (RDTA) technique, the authors have tested the feasibility of RT tasks on grid computing resources. The tasks need data files for execution, which are transferred prior to or during the task processing. Initially, the basic execution demand of a task is checked on computing resources and a list of basic feasible resources is formed. Then, the data file transfer time of the required files is calculated from data storage resources to the grid computing resources. After this calculation, the total execution time is calculated and if a task can be executed within its deadline by considering all the time constraints, then the task is termed as schedulable, otherwise, unschedulable. The tasks set is schedulable only, if all the tasks in a set are schedulable. The task schedulability analysis process is portrayed in Figure 7.

Energy efficient genetic-based scheduling

To achieve efficient energy in computational grids (CGs) is a core concern nowadays. Joanna Kołodziej et al. proposed tasks in batch mode with zero dependencies between them. The authors considered two effective scheduling functions in hierarchy mode. There are three levels of hierarchical mode which communicate over the Internet. Only one task is executed at one CG. No tasks are allowed to preempt the entire process. The two optimization goals used in scheduling are makespan minimization and average energy consumption. In idle mode, each machine takes minimal energy and maximal power supply to reload the process. Three genetic methods are used to solve difficult scheduling issues. Initially, the population is generated. After selecting a parent node, crossover and mutation are applied to the individual root node and replace the parent node with the new population. Finally, an optimal individual population is generated. The proposed model outperforms the other existing grid scheduling techniques such as relative cost, Min-Min, and Tabu search.

RA schemes for RT services in cloud computing

Cloud computing is the deployment of servers located remotely on the Internet to store, manage, and process the data. Such servers have large processing powers rather than a local server.

Cloud computing is on-demand services delivery on a pay-as-you-go basis over the Internet. There are two basic types of cloud models: service models and deployment models. Service model refers to the kind of services the cloud offer, while deployment model refers how to deploy the application on the cloud. The service model is further sub-divided into three categories: software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS). SaaS are the cloud hosted applications and virtual desktops; PaaS are operating systems, database management, and deployment tools; and IaaS are physical data centers, servers, network, VMs, storage, and load balancers. The three types of services are provided after establishing SLA between consumer and provider. The basic SLA determines the time at which the service will be provided and the corresponding usage cost.

The cloud deployment models are further divided into three main categories: public, private, and hybrid clouds. Cloud computing is an automatic, pool of resources, on-demand service (pay-per-use), secure, economical, and easy maintenance system. The main benefits of cloud include its flexibility, security, accessibility, recovery from disaster, increased collaboration, document control, and automatic software adaptable system. The RA problems in cloud computing are mainly driven by monitoring, analyzing, and thoroughly checking performance of the deployed resources.
which ensure the agreed QoS to the intended user applications.

The general architecture of the cloud environment is presented in Figure 8, and a taxonomy is provided in Figure 9 taken from agileanswer.blogspot.com.

The cloud computing RA schemes for RT services are detailed as follows.

**Partition problem–based dynamic provisioning and scheduling scheme**

An innovative, cost-efficient technique known as partition problem–based dynamic provisioning and scheduling (PPDPS) for scheduling deadlines constrained workflow application was proposed by Vishakha Singh et al. The PPDPS algorithm works mainly in two phases, namely, subset-sum problem, and k-means clustering. The degree of heterogeneity of different machines is first decided, and then the speed information of VMs in millions of instructions per second (MIPS) is provided by the caching service provider (CSP). The K-mean algorithm is used to regulate the speed of VMs. The subset-sum problem approach is used to schedule the workflow to meet its deadlines at a low execution cost. It’s an NP-Complete problem which determines whether a set can further be divided into subsets, such that the sums of the elements in the subsets are equal. The greedy approach is used which solves the problem in polynomial time. If any of the VMs get fail during task execution, then additional storage–based model transfers the data to another VM. This model helps in easy data recovery. The PPDPS technique is compared against IaaS cloud partial critical path (IC-PCP) and dynamic provisioning dynamic scheduling (DPDS) techniques. The working of the proposed PPDPS model is portrayed in Figure 10.

**Dynamic scheduling bag of tasks**

Nazia Anwar and Deng proposed a dynamic scheduling of bag of tasks (DSB) method for scheduling scientific workflows dynamically and elastic provisioning of VMs. The main objective of this study is to maximize the total utilization of computing resources with minimum execution cost satisfying the task deadline constraints. The proposed technique works in five steps. Initially, all tasks are assigned priorities in order to guarantee task dependencies using heterogeneous earliest finish time (HEFT) approach. The tasks are then grouped horizontally in the same level to provide parallelism. The execution time of a task is pre-calculated that determines whether the task will fulfill the deadlines during processing. If the tasks execution exceeds deadline, the next cost-efficient VM is used to process the task. After mapping the tasks on VMs, all tasks are put into a ready queue. The task is considered ready for execution if all the precedence tasks can be
completed successfully. Furthermore, the elastic resource provisioning method is used to dynamically adjust the number of VMs instances to ensure the completion of workflow within its deadlines. It is claimed that the DSB outperforms WRPS, SCS, and HEFT techniques. Figure 11 shows the working flow of the DSB algorithm.

Heuristic-based RA schemes
Gawali and Shinde\textsuperscript{34} proposed heuristic methods to accomplish RA and task scheduling efficiently in a cloud computing environment. Numerous size and type of data is offloaded to the cloud for execution. The analytical hierarchy process (AHP) assigns a rank to each task on the basis of priorities. Then, each task is assigned to the VM using bandwidth aware tasks scheduling (BATS) and BAR optimization techniques. The load on the VM is continuously checked using longest expected processing time (LEPT) algorithm. If resources are unavailable, the task has to wait for its turn in the waiting queue. If a resource is overloaded, the tasks are distributed on other resources using divide and conquer algorithm. The proposed heuristic methods improved the performance in terms of RA. The task allocation process of the aforementioned heuristic is depicted in Figure 12.

Data-aware RA scheme
To reduce the cost of execution while meeting the deadline constraints is one of the important considerable factors in hybrid clouds. A data-aware scheduling methodology is proposed in Nadjaran et al.\textsuperscript{35} to support deadline requirements of data-intensive applications. Data intensive is one of the applications which is used to analyze a large number of datasets. The proposed method takes into account the data locality, transfer time, and network bandwidth and then checks if current private cloud resources are enough to complete the task within a specific time interval. It calculates the extra resources required to execute tasks within deadlines. The remaining time in meeting the deadlines is first calculated, and then the remaining number of needed resources is computed. The remaining tasks are scheduled on dynamic resources. The proposed algorithm executes data-intensive independent tasks within strict deadlines while minimizing total execution cost and the total number of required resources. Figure 13 shows the working of the proposed data-aware scheduling methodology.

Earliest finish time duplication approach
Fully utilization of the selected resources is an important factor in cloud task scheduling. Delay in communication between resources degrades overall system performance. To overcome these limitations, Zhaobin

---

**Figure 9.** Cloud computing taxonomy.
Liu et al. proposed earliest finish time duplication (EFTD) approach to pre-process the cloud resources effectively. Directed acyclic graph (DAG) method is used to improve the utilization of resources. In this method, tasks with the highest priority are assigned to the selected resource for further processing. Three steps are performed to assign resources to specific tasks: calculating earliest start time (EST), earliest finish time (EFT), and task allocation. In this approach, initially EST is calculated to regulate processing unit and an EST value is assigned to processing resource. Then, the EFT is calculated to choose processing unit and assign it smallest EFT value. Finally, the results of both times (EST and EFT) from ready tasks are compared. The high priority task is assigned to that resource whose both times are same. The next task with lower priority which has both times different is chosen from the ready queue.

To reduce the processor scheduling time, parent nodes of two key tasks are duplicated. In this way, the communication process is improved as compared to the other counterparts (HEFT, AEST, HEFT-TL). The whole scenario is depicted in Figure 14.

**Task scheduling and load balancing technique**

Load balancing is an important factor while dealing with numeral requests sent to the server through a network. Arun Sangwan et al. proposed task scheduling algorithm in cloud computing based on load balancing to meet user demands and make full use of resources. The whole process considers the above mentioned two aspects in four phases. Initially, when new workflow arrives, it is submitted to the pre-processor component for computing different attributes of all ready tasks. Then, the ready task is placed in a ready queue for further processing. When services become available, the scheduler executes all tasks available in the ready queue and when the task completes its execution, the executor notifies the pre-processor of the completed task. This technique tries to balance the load on all available resources so that tasks can be completed in prespecified deadlines. The whole process is briefly summarized in Figure 15.
Proactive and reactive scheme

To meet RT cloud computing environment, there was a need to reduce system's energy consumption. To resolve this issue, Chen et al.\textsuperscript{53} proposed proactive and reactive scheduling (PRS) procedure to make efficient utilization of system resources with reduced energy consumption. When a new task is arrived in the system, the PRS algorithm checks whether the task has urgency. If it has urgency parameter, then it is referred to the urgent task queue. Otherwise, task is further sent to waiting queue in ascending order by their laxity. In addition, PRS checks the requirements of waiting task not to exceed the available system resources. Later on tasks are scheduled to VMs for execution. The whole scenario is illustrated in Figure 16.

Allocation-aware task scheduling

Allocation-aware task scheduling (ATS) algorithm is proposed by Sanjaya K Panda for multi-cloud environment.\textsuperscript{48} Each cloud consists of different datasets to deploy VMs. The proposed model comprises three steps, that is, matching, allocating, and scheduling. In matching phase, cloud manager preserves a universal queue to insert an incoming request from users. The requests are served as first in first out (FIFO). The manager matches the task with other VMs to find optimized VM among all. Then, the request is removed and the completion time of task on a specific VM is computed. Thus, one task is selected from the global queue at a time. In addition, the VM is found by manager to grasp minimum completion time (MCT) for task supervised by cloud. Task scheduling is decided in allocation phase. In allocation phase, resources are assigned to tasks. The scheduler executes all tasks to carry out the computation. Each cloud resource executes more than one task concurrently. The experimental results demonstrate better performance than round robin (RR) and cloud list scheduling (CLS) algorithms. The entire process is portrayed in Figure 17.

Bandwidth-aware RA

QoS attainment is incredible essential of all the users. In cloud computing, processing a task in a flow is vital. A non-linear programming model\textsuperscript{76,77} for task scheduling is proposed in Sindhu.\textsuperscript{39} In this model, several tasks are ready to be scheduled. Various resources are assigned to each task. Among all, some resources are used and the remaining are being idle. To avoid resource power wastage, limited network bandwidth is considered. Based on accessible bandwidth, each task is forwarded to every VM. The proposed model produced better accuracy than other existing models by taking full utilization of resources and reducing the waiting time. The working flow of this technique is shown in Figure 18.
S Raghavan et al.\textsuperscript{50} proposed BAT approach for the optimum solution of workflow scheduling. The workflow scheduling contains two chunks, namely, task scheduling and mapping tasks and resources. In this approach, task and resources are mapped together to reduce the entire cost of execution. The model contains three resources and four tasks, each resource has different execution costs. Furthermore, the cost of each task is computed on each resource. With the help of mapping function, a list is maintained when every task is mapped with resource based on minimal value. Each resource doesn’t contain more than one task. Finally, the result is computed and compared to the best resource selection (BRS) algorithm. The model showed overall nominal cost. The working of BAT algorithm is shown in Figure 19.

\textbf{Developmental genetic programming}

In RT computing, practicing cloud infrastructure is a new concept. Resources are assigned to the tasks through cloud infrastructure. It is very important to design an efficient and effective procedure for RA through cloud. Deniziak et al.\textsuperscript{55} analyzed the problem of cloud RA to minimize execution cost in RT applications. The existing methodologies, that is, iterative improvement\textsuperscript{46} and constrained logic programming,\textsuperscript{49} briefly explained and resolved the application cost problem, but still there is a chance of improvement to
minimize the execution cost. An efficient genetic procedure is proposed to minimize the cost of applications with high QoS in a cloud computing environment. Initially, the system comprises distributed methods considering the worst-case scenario where all tasks are started at equal time. All tasks are scheduled in a fixed order with a definite time frame. In addition, the methods are converted into several task graphs due to static scheduling. Also, primary population of genotypes is created and solved through genetic procedure. Here, the developmental scheduling algorithm is used for assigning and scheduling all tasks which outperform the ideal solution. The whole process is shown in Figure 20.

**Federation-based RA scheme**

Jiayin Li\(^6\) proposed a new approach in a cloud computing environment online for task scheduling on IaaS cloud. In this research, various data centers meet with the help of federated method. Each data center contains manager server which holds information about VMs and communicate with each other, respectively. The tasks are stored in a database in the cloud, then cloud collects tasks and distributes it in the entire cloud system considering resource accessibility in the other clouds. The resource scheduling is being checked by monitory infrastructure. The infrastructure, producer, and consumer are communicated with each other in two modes, namely, push and pull mode. While making decisions, consumer pulls information about resources from other existing clouds. After finishing task execution, producer pushes the data to the consumer. Two
distinct methods, namely, advance reservation (AR) and best effort (BE) are used for hiring capacities from cloud computing infrastructure. Both push and pull modes are used in the basic RA model. Finally, DAG is used in the application model. Energy-aware local mapping significantly reduces energy consumption in the federated cloud computing environment.\textsuperscript{78,79} This procedure is illustrated in Figure 21.

\textbf{Adaptive genetic approach}

Allocation and scheduling of tasks on a VM is a difficult job in a hard RT environment. To resolve this problem, Amjad Mahmood and Khan\textsuperscript{40} proposed adaptive genetic algorithm (AGA) in a cloud computing environment. The model consists of numerous tasks having the work pressure to complete its tasks while meeting the deadline. Tasks have a precedence relationship when communicate with each other on VM denoted by a DAG. The scheduling of tasks is done using chromosomes to generate the population. Multiple crossover is performed on randomly selected chromosomes without troubling the scheduler. Furthermore, mutation is accomplished by selecting a gene from chromosome. Each gene is capable of changing bits with some probability. The proposed AGA model is compared with the other techniques such as DAG technique, greedy search approach, and non-adaptive GA. Better outcomes are achieved by the proposed algorithm. This process is shown in Figure 22.

\textbf{Dynamic fault-tolerant elastic scheduling}

It is vital to improve the resource utilization in cloud computing environment. Hui Yan et al.\textsuperscript{38} proposed a fault tolerant algorithm to accomplish resource utilization. The data center comprises multiple hosts, each having its own VM.\textsuperscript{80,81} A user maintains task flow in a task queue. The system schedules the task which keeps record of both task scheduler and performance monitor. The monitor is responsible to retain the status of the system performance. On the other hand, task...
scheduler schedules task when getting feedback from monitor. The task scheduler, data center, and monitor communicate with each other using star topology. The performance of the model is verified using CloudSim simulator through Google tracelogs. The following Figure 23 shows the whole process of dynamic fault-tolerant elastic scheduling (DFES) algorithm.

**Earliest deadline first greedy approach**

Karthik Kumar et al.\(^6\) proposed a VM allocation procedure for RT tasks. The main objective of the proposed technique is to decrease the cost of tasks allocation while meeting deadlines. The authors proposed a greedy approach based on earliest deadline first (EDF) algorithm. Initially, tasks are allocated separately to access VMs. When VMs became inadequate to accomplish the task on time, then based on lookup table, other low-priced VMs are selected by greedy approach to complete the tasks before expiry of the time constraint. When tasks overlap each other and produce high cost, then identifying overlapping tasks and allocating resources together create outcomes in polynomial time. The proposed model shows the allocation of tasks. This process is shown in Figure 24.

**Deadline-oriented VM allocation approach**

Kyong Hoon Kim et al.\(^6\) proposed an algorithm not only to reduce the task allocation cost but also increase the system consistency for both hard and soft RT services. In hard RT service, if a task does not meet the timing constraint, then penalty is charged. In soft RT service, if a task does not meet time constraints, then there is no penalty and results are accepted, but the QoS is decreased. Depending on the deadline type, the user requests hard or soft VM for further processing. Initially, the user requests a VM by providing all the information to the vendor about RT applications. Then, the provided information is first analyzed for creating one real-time VM (RT-VM) request and the
vendor offers a VM from the cloud computing environment. The VM meets the task’s requirement constraint, and its information is submitted to the user later. Finally, the user executes the RT applications. The whole scenario is depicted in Figure 25.

**Approximate computation–based RA scheme**

In a cloud computing environment, QoS provision is preferable requirement for the consumer. Georgios L Stavrinides and Karatza\(^\text{27}\) proposed approximate computation–based scheduling technique for RT system. To schedule tasks in a ready queue, heuristic is applied to select task and VM. In the beginning, tasks are prioritized on the basis of EDF policy. In case of tie, task with the highest cost is selected. Once the task is selected by the scheduler, it is assigned to a VM based on the earliest estimated finish time (EEFT) approach. The proposed model is made energy efficient through the utilization of dynamic voltage frequency scaling (DVFS) technique during the VM selection process. Also, the model is cost effective when it reduces the idle time of the VM. The proposed model provides better results than other existing techniques considering total energy consumption, SLA violation ratio, average result precision, and total monetary cost as performance parameters. The working of the proposed model is represented in Figure 26.

**Periodic server scheduling scheme**

Numerous systems share single computing platform to enrich flexibility and reduce the execution cost through virtualization equipment instead of using separate hosts. To overcome this issue, Sisu\(^\text{56}\) proposed novel scheduling structure in RT environment. All the
incoming tasks are prioritized considering pre-emptive fixed scheduling. Then, these tasks are scheduled according on the virtual processing units. Every operating system is liable on tasks scheduling. The processing is executed in two conditions: runnable and non-runnable. Xen scheduler schedules dependent functions on distinct VM and spreads its other parts. The jobs continued to execute on a single core hierarchy considering soft RT tasks. Also, tasks are implemented on sporadic server where
do-schedule method is called when virtual processing differs from running tasks. The proposed model performance is computed on different quanta whose range lies in 1–10 ms. The proposed model achieved better results at fine-grained quantum considering tasks overhead.

**Heuristic-based EDF scheme**

In software services, the data locality problem exists nowadays. There is a need to reduce the problem of workload during execution of multiple tasks. Georgios L Stavrinides and Karatza\(^{42}\) presented a solution to overcome heavy workload problem. In this scenario, tasks are dynamically scheduled in cloud computing under different circumstances. Three heuristics are applied on a RT tasks set such as non-data-aware earliest deadline first (ND-EDF), data-aware earliest deadline first (DA-EDF), and enhanced data-aware earliest deadline first (EDA-EDF). In ND-EDF, tasks are entered in global queue for central scheduling. Then, tasks are sorted in descending order of mean computational costs. Tasks having largest cost are selected first for scheduling. After scheduling, tasks are assigned to the VM with the EST. The VM executes tasks according to the earliest deadline. In DA-EDF, tasks are scheduled like NDA-EDF via the largest mean computational costs. The tasks are assigned to the VM having EST. But the tasks are executed with their job’s deadline. In EDA-EDF, the procedure of tasks selection is same as other two procedures. Here, tasks are executed according to the EST employing EDF policy. NDA is matched with the other two heuristics DA and EDA-EDF to aware the system performance on data locality. In the proposed model, data locality is considered during scheduling.

**Application-directed check pointing and approximate computation scheme**

Georgios L Stavrinides and Karatza\(^{45}\) focused on reducing cost and fault tolerance in SaaS cloud computing environment to achieve good quality within time constraints. The six scheduling techniques EDF, earliest deadline first with restricted approximate computations (EDF-RAC), earliest deadline first with full approximate computations (EDF-FAC), earliest deadline first with application-directed checkpoints (EDF-ADC), earliest deadline first with application-directed checkpoints and restricted approximate computations (EDF-ADC-RAC), and earliest deadline first with application-directed checkpoints and full approximate computations (EDF-ADC-FAC) are compared to compute the performance of overall system. In EDF, tasks are in queue according to earliest estimated start time. The tasks are engaged in selected queue considering EDF. In EDF-RAC, tasks in queue are allocated through EDF. In EDF-FAC, tasks selection is same as EDF and EDF-RAC procedures. The tasks are scheduled only to execute the mandatory portion. In EDF-ADC, tasks are scheduled to execute mandatory portions. The comparison among scheduling algorithm proved that EDF-FAC produced better results and EDF-ADC gave the best results in worst scenarios.

**EDF and unfair semi-greedy approach**

H Alhussian et al.\(^{26}\) proposed cloud computing architecture that consists of two parts: master node (MN)
and VM. In this scenario, it is assumed that MN has VMs pool. When RT job comes to the MN, the master scheduler picks a VM resource from the pool and assigns tasks to it until VM is fully loaded. When the minimum required number of VM is determined, the MN scheduler assigns RT services to them. Each VM node has deadline look-a-head module that gives information regarding deadline-miss event. In case of such event, the event is removed from scheduling queue and put into urgent queue. The MN scheduler at that point looks for a VM node with an unoccupied processor and assigns the urgent tasks to any such node that is found unoccupied. Sometimes, the MN scheduler also makes another VM node and appoints the urgent tasks to it. When VM deadline is suspected, it removes those tasks from waiting queue and sends the suspected tasks to the MN. By following this procedure, tasks are expected to meet the deadlines.

**Dynamic proactive reactive scheduling**

H Chen et al.\(^5\) proposed PRS algorithm that dynamically exploits PRS methods for scheduling RT tasks. The tasks are assumed aperiodic and independent. The authors examined how to reduce the system’s energy consumption while guaranteeing the RT constraints for green cloud computing where uncertainty of task execution exists. The proposed PRS algorithm achieved improved performance as compared to four typical baseline scheduling algorithms: non-migration-PRS (NMPRS), EDF, MCT, and complete rescheduling (CRS).

**Energy-aware RA**

Energy conservation is a big issue in cloud computing systems. If it is handled properly, then reducing operating costs, system reliability, and environmental protection factors can be achieved effectively. Existing power-aware scheduling approaches provide promising way to achieve this goal, but the issue is that they are not RT tasks oriented and thus lacking the ability of guaranteeing system schedulability in such situations. To solve the aforementioned problem, X Zhu et al.\(^57\) proposed rolling-horizon scheduling architecture for RT tasks scheduling in virtualized clouds and energy-aware scheduling algorithm for RT tasks. The tasks are aperiodic and independent. Two other strategies “scaling up” and “scaling down” are proposed for making trade-offs between task’s schedulability and energy conservation.

**Bag of tasks scheduling with approximate computation**

The authors in GL Stavrinides and Karatza\(^41\) devised six different techniques, that are MinMinMAC, MinMinEAC, MaxMinMAC, MaxMinEAC, Sufferage MAC, and SufferageEAC for scheduling batch of real-time tasks (BoT) on SaaS cloud computing systems. All tasks are independent and non-pre-emptive which avoid performance degradation. Each task is assigned some weight which shows its total number of required computations. If a task fails to finish on time, it takes non-completed status and gets out of the scheduling queue because its deadline is non-extendable. A task gets completed status when all its mandatory parts (jobs/sub-tasks) are executed on time. The completion of optional sub-tasks may refine the results. In the proposed model, a task can get a finished, partially finished, or skipped status. If a task is partially finished, it is called as approximate and the output may get affected. In any case other than approximate, a job is lost.

In the proposed model, when a task arrives at the central scheduler, it enters the queue and scheduler is invoked. The load on VM is calculated and the following steps are performed:

a. Estimate the completion time of a task.
b. Calculate idle time of a processor.
c. Arrange queue according to the EDF algorithm.
d. Calculate MCT of a task.

The detailed working of the proposed framework is depicted in Figure 27.

**Green cloud scheduling approach**

T Kaur and Chan\(^44\) proposed a technique to efficiently utilize the nodes in a cloud computing system to save energy consumed during the process. With the help of virtualization, heterogeneous types of tasks are efficiently assigned to the nodes within their deadline limits that are energy efficient using green cloud scheduling model (GCSM). Running tasks, resources utilization, and energy statistics information are stored in a database. Cloud user submits the task(s) and provides the deadline information for the task(s). The tasks handler and analyzer section checks whether the current task along with the constraints provided by the user exists in the database. If it is there, the task is assigned to the required resource. If the task is not found in the current database, then a special unit called green cloud scheduler (GCS) efficiently performs scheduling of tasks on appropriate nodes within the deadline limits of the tasks. The GCS also tries to eliminate idle node(s) within the system, and thus minimizes the energy consumed by unnecessary nodes. The proposed system saves energy up to 71%, while 82% of the tasks are completed within their deadline constraints.
Fuzzy dominance sort–based RA technique

X Zhou et al.\textsuperscript{28} minimized cost and makespan simultaneously for workflows deployed and hosted on IaaS clouds. This scheme proposes a fuzzy dominance sort–based heterogeneous earliest finish time (FDHEFT) algorithm which closely integrates the fuzzy dominance sort mechanism using HEFT list scheduling heuristic. The proposed scheme achieved significantly better cost-makespan tradeoff fronts with remarkably higher hyper volume and can run up to hundreds of times faster than the state-of-the-art algorithms. Two sets of simulation experiments are implemented on real-world workflows and synthetic applications to validate the effectiveness of the proposed FDHEFT. The experiments are based on the actual pricing and resource parameters of Amazon EC2. The produced results show supremacy of the FDHEFT approach with respect to cost-makespan trade-offs and a lower CPU runtime when compared to the other peers approaches like e-fuzzy PSO, NSPSO, SPEA2, and MOHEFT.

Best fit with imprecise computations

G Stavrinides and Karatza\textsuperscript{52} proposed variant of EDF scheduling scheme called EDF-BF-IC for the RT workflow applications scheduling in heterogeneous PaaS cloud that combines inaccurate bin techniques and computations. This scheduling technique consists of two main objectives. First, to ensure that all applications are not exceeding the deadlines and that results in high quality output. Second, to decrease the time spent by execution of every workflow that causes cost to the user. The EDF-F-IC scheduling heuristic consists of two phases: tasks selection and VM selection. In the first phase, priority is assigned to each task according to EDF policy. The task having the highest EDF priority is the task which has earliest end to end deadline. All the tasks are organized in ascending order of priorities. If there are two tasks and both have same deadlines, then both tasks are organized in descending order. In the second phase, when task selection is completed, then task is assigned to that VM which delivers it with the earliest estimated start time. The authors first find the initial position on which the ready task is located in the VM queue, agreeing to their priority so that least priority task does not precede the least one. Then, they evaluate if the time saved by avoiding some part of the ready task is equal or greater than the total average time forced on the ready task sub-tasks. The proposed EDF-BF-IC technique is compared to EDF by considering communication workflow, intensive, and moderate application parameters.

\textbf{Figure 27.} Workflow of BoT with approximate computation scheduling approach.
**Hybrid genetic and cuckoo search algorithm**

Min-Allah et al.\textsuperscript{24} proposed a hybrid approach for scheduling RT applications on cloud computing resources. The main performance parameters are total cost and makespan minimization. The schedulability of application is checked on cloud VMs. If application is feasible for execution, then the cost of execution is checked. The proposed hybrid genetic and cuckoo search (HGCS) algorithm selects low cost cloud computing resources where RT applications are executed with total minimum execution time. The performance of the HGCS is compared with the other two well-known algorithms, the genetic, and cuckoo search.

**RA schemes for RT services in edge computing**

When various things (devices and so on) and information are connected to the Internet, it refers to the IoT. Nowadays, billions of IoT devices are connected to the Internet and huge amount of data produced by these devices need to be processed in a very short span of time. One solution to the huge data storage and processing is the cloud environment. As the distance between the IoT device and cloud increases, the transmission latency also increases which increases the response time. Furthermore, smart handheld devices have limited storage and computation capabilities which cannot accommodate applications demanding high storage and RT processing.\textsuperscript{52} The solution to this problem is the edge computing platform. The mobile edge computing technology helps in overcoming the limited memory and storage constraints problems by providing cloud computing facility adjacent to the smart IoT devices. The edge computing platform allows some application processing to be performed by small edge servers (ESs) located between the IoT devices and the cloud in location curiously physically closer to the IoT device.

In edge computing, server is located in the edge network. The distance between the IoT devices and the computation resource is a single hop, due to which the latency is low, and the jitter is also very low. Edge computing is geo-distributed with location awareness and mobility support. Edge computing provides limited service scope with limited hardware capabilities to the mobile users. The storage capacity and computational power are also limited in edge computing.

Figure 28 shows the general architecture of edge computing while Figure 29 portrays the taxonomy of edge computing presented by Ejaz Ahmed et al.\textsuperscript{83}

**Resource matching–based allocation scheme**

Due to the low latency in RA, edge computing is becoming essential nowadays. There is a need to provide optimal solution in a cloud computing environment. Hengliang Tang et al.\textsuperscript{29} proposed an algorithm for dynamic RA in edge computing environment. The RA comprises resource matching (RM) and scheduling processes on ESs. To reduce network traffic and improve the user experience is a main part of every application. All information related to data is stored in memory for further use. In resource scheduling algorithm, the resources are scheduled with the help of edge orchestrator (EO). The traffic data are scheduled into the disk of ESs. All the jobs are submitted to the corresponding ES by the access point (AP) where every job contains multiple tasks. Now, the corresponding container for the tasks is configured by the ES. Every ES can present multiple containers containing RAM and CPU. Furthermore, each container executes single task at a time. Finally, every ES matches its tasks and containers by an RM algorithm. The proposed model presented better results than the existing techniques.

**RA schemes for RT services in fog computing**

With the increasing volume of IoT devices, the main problem faced by cloud computing is the latency. IoT devices need sufficient bandwidth for transferring huge amount of data for communication as these devices send all the data to the cloud for processing and storage, which consume bandwidth and energy. The processing of RT data meeting the deadline constraints is challenging for the cloud due to its high latency.

The cloud model has insufficient ability to handle the IoT requirements in case of bandwidth, latency, and huge data volume produced by the IoT devices. Hence, there is a need to bring the cloud facility closer to the IoT devices to minimize the use of bandwidth and energy consumption and to reduce the latency.
CISCO introduced fog computing to overcome the deficiencies of the cloud computing. Conceptually, fog is the intermediate layer between the IoT devices and the cloud. The general architecture of the fog layer is represented in Figure 30.

Fog is not the replacement of cloud due to the limited number of resources but its extension which is dominant in terms of lower service delay, processing cost, and response time. Similarly, augmented reality, virtual reality, and time sensitive applications which have rigid service delivery deadlines are also not efficiently execute on remote cloud resources. Using fog computing, the produced data are pre-analyzed on the asset, minimize the volume of data, and efficient management of runtime behavior is carried out. Conversely, due to the wireless connectivity, power failure, and devolved management, the failure ratio is high in the fog computing. Naha et al. presented taxonomy of fog computing based on the requirements of application, infrastructure, and platform which is presented in Figure 31.

**Hybrid EDF approach**

Georgios L Stavrinides and Karatza proposed hybrid technique to scheduled RT workflows in a cloud computing environment. In fog computing, IoT workflows for dynamic scheduling in three-tiered architecture. The IoT comprises devices and sensors which transmit data via Wi-Fi network to the fog layer. The data arrive dynamically at central scheduler toward Poisson stream. Each job is non-preemptible and denoted by a DAG. The tasks are initially prioritized using EDF policy. Then, the tasks are scheduled and assigned to the VM according to EFT technique. The proposed model produced improved results as compared to fog-EDF. This general process is shown in Figure 32.
T asks buffering and offloading policy

A task shows poor performance on the execution nodes due to limited capacity. Cloud computing resolves this issue by utilizing powerful resources which produce better quality results in reduced time. But it increases chances of data loss because several tasks transfer information over the network. To overcome this issue, Lei Li et al.\textsuperscript{25} proposed an algorithm for processing heterogeneous RT tasks. It comprises three tiers to increase the QoS. Initially, the end tier is accessible by the data source of fog system. Second, the fog tier shows a minor latency during transmission between the fog and end devices. Third, the cloud tier contains unlimited resource capacity to attain extra-ordinary performance. A single parallel virtual queue at the fog node is made to reduce time complexity. The proposed model performance is compared with the two RA techniques, that is, the RR and maximum resource utilization. The results showed that greater value is achieved by a task when task buffering and offloading policy is adopted. This scenario is depicted in Figure 33.

Tasks buffering and offloading policy

RA schemes for RT services in multicore systems

The RA schemes for RT services in multicore environment are mainly categorized into online, offline, and mixed approaches.\textsuperscript{86} The online RA approaches map RT applications dynamically at runtime without any
prior knowledge of the system. In offline RA approaches, the applications have advanced knowledge of the whole system and the resources status. These approaches have drawback of lack of knowledge how to solve variant application workload problem at runtime dynamically. In mixed approaches, application mapping and selection is pre-calculated at design time.

**Rate monotonic scheduling with reduced priority levels approach**

In reduced priority levels technique, the hard RT tasks are grouped into multiple classes. Priority is assigned to a class instead of individual task which means that if there are total $n$ tasks in a set, then the number of priority levels will be less than $n$ which shows that one of the class has more than one task. The tasks within a class are prioritized based on the rate monotonic scheduling (RMS) approach. This procedure reduces the assignment of priority levels.

**Hybrid cuckoo search–based algorithm**

Xiangtao Li and Yin proposed hybrid cuckoo search (HCS)-based algorithm to resolve permutation flow shop scheduling problem (PFSSP). The core purpose of the PFSSP scheduling is to minimize the makespan. Based on a random key, largest rank value (LRV) rule is proposed to create an appropriate cuckoo search. In addition, with the combination of Nawaz Enscore Ham (NEH) heuristic algorithm, a population with an assured value and range is generated. A fast local search (FLS) is injected to increase the local utilization for consideration. Finally, comparison is carried out for better performance of the HCS model with the PSOMA, OSA, PSOVNS, and HDE algorithms.

**Online accrued scheduling scheme**

Shuo Liu et al. proposed scheduling algorithm for RT system on the task model online with the utility functions. Two different utility functions that are profit and penalty are associated with the task. If the task completes its execution within a specific time period, it is considered as profit. On the other hand, if the task misses its deadline, then it must pay a penalty. To schedule several RT services requests, two non-preemptive heuristic scheduling are used. The purpose of using heuristics is to accept, schedule, or abort RT services to maximize the sing time utility function. Initially, all the incoming RT tasks are accepted to assess the requirement for the system. Then, all the incoming tasks in a ready queue are scheduled for execution. Finally, reject awaiting requests and terminate the task when accomplished. The proposed model shows better results than EDF, GUS, risk-reward, and
PP-aware scheduling algorithms. The high-level working of the proposed scenario is depicted in Figure 34.

**Least feasible speed technique**

In Min-Allah et al.,62 the authors introduced least feasible speed (LFS) technique for checking the schedulability of the RT systems with minimum energy consumption procedure on multicore systems. The authors suggested that the first point in a scheduling points set cannot guarantee the schedulability of the tasks with minimum power consumption. So, each task feasibility is checked by multiple scheduling points in a set. The proposed LFS technique outperforms the existing first feasible speed (FFS) counterpart. Furthermore, the authors have also suggested a strategy for load balancing on cores.

**Load balancing by tasks splitting and tasks shifting strategy**

Load balancing plays a vital role in improving multicore systems efficiency. In Hussain et al.,59 the authors distribute the RT tasks on multicore systems in a way that the computational demand of tasks is fulfilled in specified deadlines and the total utilization of the core remains in bound. They perform utilization-based tests. When a core utilization becomes greater than 100%, then tasks are shifted to the other underutilized core.

**Compatibility-aware task partitioning scheme**

It is essential to improve task execution within deadline constraints on multicore platforms. Qiushi Han et al.51 provide efficient tasks scheduled using RMS on multiple core platforms under fault tolerant requirements. In the proposed scheme, multicore scheduling is divided into two groups, namely, global and partitioning scheduling. In global scheduling, every task is executed on any core, while in partitioning scheduling, every task is assigned to a core and all tasks are executed on that specific core. The authors considered partitioning scheduling because of low overhead. Only one task is allowed to be allocated at a time. Also, a task is assigned to a respective core which computes the compatibility. The task is re-executed when fault is identified. A check pointing scheme is merged with the compatible task to increase system utilization. Figure 35 shows this phenomenon.

**Simple combined resource usage partitioning**

The usage of reserve memory on the CPU is increased by some task running on the system due to which latency is also increased during the execution of tasks. To overcome this problem, Baquero and Gustavo36 focused on the memory resource division for RT systems. Initially, fixed priority is assigned to a number of tasks and then the tasks are scheduled on multiple core platforms which share a single memory known as cache. The scheduling of hard RT tasks is improved by co-scheduling memory and CPU. Then, memory bandwidth and memory latency are progressing using three-dimensional systems with on-chip dynamic random access memory (DRAM). This is how tasks are scheduled to meet the deadline for the memories. The proposed model scheduled 19.5% more tasks than the other existing classic scheduling methodologies such as simple combined resource usage partitioning (SCRUP) and TATP.

**Enhancing shared cache performance–based approach**

PP Kumar et al.30 proposed task reprocessing scheme to enhance the performance of a common cache memory on a RT multicore system (with quantum-placed universal recurring co-scheduling model) and to reduce the overhead in RT scheduling by encouraging eligible task sets to reprocess based on heuristics called enhancing shared cache performance (ENCAP).
By utilization, processing rates are determined by splitting tasks into sub-tasks having pseudo cutoffs (intermediate cutoffs). These sub-tasks are processed using EDF scheme. Among sub-tasks, there exist cutoff miss or tie with the same cutoffs. The tie is broken by ENCAP.

The ENCAP is used to reschedule loads which are ignored due to the low priority load. Loads are assigned statically and processed based on a cache-aware RT EDF. Every load in a task is discharged regularly which is called an activity of the load. A periodic load can be derived by a slot which is partitioned between its consecutive load discharges and reprocessing cost. It implies the largest reprocessing time. Every activity of a load has a cutoff corresponds to the discharge time.

**Large time demand analysis technique**

In scheduling points test, the basic feasibility of a task is determined by testing scheduling points for the lowest priority task first. But main drawback of such tests is that is not possible to address the total execution demand of such tasks at a small number of points. Instead, the feasibility is determined at very large scheduling points set. In Min-Allah, the author performs the feasibility analysis of periodic RT tasks by testing large number of scheduling points. At each scheduling point, the cumulative demand of a task is tested. The task is declared unschedulable if at any point, the task execution demand is greater than the total processor capacity.

**Conclusion**

The RA scheme in HPC systems plays a vital role in allocating resources to the user applications, especially when such applications have associated time parameters and need resources within specified time and user budget constraints. In this article, we have surveyed RA schemes for RT services in HPC paradigms, including grid, cloud, fog, edge computing, and multicore systems. Working of these schemes has been evaluated theoretically and presented pictorially so that a new researcher can be facilitated on a single platform. The detailed comparison on the basis of common parameters provides an opportunity to easily find architectural and implementation-related similarities and differences among different RA schemes. We have provided comprehensive analysis, which distinguishes this article from the existing surveys in the RA domain. This survey specifically consolidated RA schemes only for RT services which involve execution deadlines. In this article, RA schemes are classified on the basis of RT workloads and an overview of the most commonly used schemes was presented in large-scale HPC systems. Each scheme was uncovered on the basis of some common performance parameters. We believe that this article will help researchers working in the HPC and RT systems domains to identify and select well-suited RA scheme according to a particular RT service.

In this article, the state-of-the-art RA schemes in different HPC domains are covered comprehensively, but still ample opportunities exist for upcoming investigators, for example, how to scale the discussed RA schemes efficiently to accommodate big data applications with diverse workloads and QoS requirements demanding computationally intensive and powerful distributed resources. Another dimension is to explore the reliability factor of the HPC resources because it can put direct impact on the performance of RT applications.

**Acknowledgment**

The authors would like to thank Qatar National Library (QNL) for supporting in publishing this research work.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

**Funding**

The author(s) received no financial support for the research, authorship, and/or publication of this article.

**ORCID iDs**

Muhammad Shuaib Qureshi https://orcid.org/0000-0002-3908-6302

Samir Brahim Belhaouari https://orcid.org/0000-0003-2336-0490

**References**

1. Zhang Y, Tian Y, Fidge C, et al. Data-aware task scheduling for all-to-all comparison problems in heterogeneous distributed systems. *J Parallel Distr Com* 2016: 93–94: 87–101.

2. Hussain H, Malik SU, Hameed A, et al. A survey on resource allocation in high performance distributed computing systems. *Parallel Comput* 2013: 39(11): 709–736.

3. Xie T and Qin X. Enhancing security of real-time applications on grids through dynamic scheduling. In: Feitelson D, Frachtenberg E, Rudolph L, et al. (eds) *Job scheduling strategies for parallel processing*, vol. 3834. Berlin: Springer, 2005, pp.219–237.

4. Qureshi MB, Dehnavi MM, Min-Allah N, et al. Survey on grid resource allocation mechanisms. *J Grid Comput* 2014: 12(2): 399–441.

5. Amir Y, Awerbuch B, Barak A, et al. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE T Parall Distr* 2000: 11(7): 760–768.
10. He L, Jarvis S, Spooner D, et al. Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids. In: Proceedings of the 5th IEEE/ACM international workshop on grid computing, Pittsburgh, PA, 8 November 2004, pp.402–409. New York: IEEE.

11. Caron E, Chouhan P and Desprez F. Deadline scheduling with priority for client-server systems on the grid. In: Proceedings of the 5th IEEE/ACM international workshop on grid computing, Pittsburgh, PA, 8 November 2004, pp.410–414. New York: IEEE.

12. Liu JWS. Real-time systems, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2000.

13. Bini E and Buttazzo GC. Measuring the performance of schedulability tests. Real-Time Syst 2005; 30(1–2): 129–154.

14. Casanova H, Berman F, Obertelli G, et al. The apple parameter sweep template: user-level middleware for the grid. In: Proceedings of the 2000 ACM/IEEE conference on supercomputing, Dallas, TX, 4–10 November 2000, pp.60–60. New York: IEEE.

15. Naha RK, Garg S, Georgakopoulos D, et al. Fog computing: survey of trends, architectures, requirements, and research directions. IEEE Access 2018; 6: 47980–48009.

16. Shu W, Cao J, Zhang Q, et al. Edge computing: vision and challenges. IEEE Internet Things 2016; 3(5): 637–646.

17. Hridita BA, Irfan M and Islam MS. Mobility aware task allocation for mobile cloud computing. Int J Comput Appl 2016; 137(9): 35–41.

18. Mao Y, You C, Zhang J, et al. A survey on mobile edge computing: the communication perspective. IEEE Commun Surv Tutor 2017; 19(4): 2322–2358.

19. Mangla N, Singh M and Rana SK. Resource scheduling in cloud environment: a survey. Adv Sci Technol Res J 2016; 10(30): 38–50.

20. Braun TD, Siegel HJ, Beck N, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Dist Com 2001; 61(6): 810–837.

21. Min-Allah N, Khan SU, Ghani N, et al. A comparative study of rate monotonic schedulability tests. J Supercomput 2012; 59(3): 1419–1430.

22. Nasri M. On flexible and robust parameter assignment for periodic real-time components. ACM SIGBED Rev 2017; 14(3): 8–15.

23. Beloglazov A, Abawajy J and Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comp Sy 2012; 28(5): 755–768.
40. Mahmood A and Khan SA. Hard real-time task scheduling in cloud computing using an adaptive genetic algorithm. *Computers* 2017; 6(2): 1–20.

41. Stavrinides GL and Karatza HD. Scheduling real-time bag-of-tasks applications with approximate computations in SaaS clouds. *Concurr Comput: Pract Exp* 2020; 32: e4208.

42. Stavrinides GL and Karatza HD. The impact of data locality on the performance of a SaaS cloud with real-time data-intensive applications. In: *Proceedings of IEEE/ACM 21st international symposium on distributed simulation and real time applications (DS-RT)*, Rome, 18–20 October 2017, pp.1–8. New York: IEEE.

43. Sangwan A, Kumar G and Gupta S. To convalence task scheduling in a decentralized cloud computing environment. *Rev Comput Eng Res* 2016; 3(1): 25–34.

44. Kaur T and Chan I. Energy aware scheduling of deadline-constrained tasks in cloud computing. *Clust Comput* 2016; 19(2): 679–698.

45. Stavrinides GL and Karatza HD. Scheduling real-time parallel applications in SaaS clouds in the presence of transient software failures. In: *Proceedings of 2016 international symposium on performance evaluation of computer and telecommunication systems (SPECTS)*, Montreal, QC, Canada, 24–27 July 2016, pp.24–27. New York: IEEE.

46. Qureshi MB, Alrashed S, Min-Allah N, et al. Maintaining the feasibility of hard real-time systems with a reduced number of priority levels. *Int J Appl Math Comput Sci* 2015; 25(4): 709–722.

47. Liu Z, Qu W, Liu W, et al. Resource preprocessing and optimal task scheduling in cloud computing environments. *Concurr Comput: Pract Exp* 2015; 27(13): 3461–3482.

48. Panda SK, Gupta I and Jana PK. Allocation-aware task scheduling for heterogeneous multi-cloud systems. *Proc Comput Sci* 2015; 50: 176–184.

49. Sindhu S. Task scheduling in cloud computing. *Int J Adv Res Comput Eng Technol* 2015; 4: 3019–3023.

50. Raghavan S, Sarwesh P, Marimuthu C, et al. Bat algorithm for scheduling workflow applications in cloud. In: *Proceeding of 2015 international conference on electronic design, computer networks & automated verification (EDCAV)*, Shillong, India, 29–30 January 2015, pp.139–144. New York: IEEE.

51. Han Q, Wang T and Quan G. Enhanced fault-tolerant fixed-priority scheduling of hard real-time tasks on multicore platforms. In: *Proceedings of IEEE 21st international conference on embedded and real-time computing systems and applications*, Hong Kong, China, 19–21 August 2015, pp.21–30. New York: IEEE.

52. Stavrinides GL and Karatza HD. A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds. In: *Proceedings of IEEE 3rd international conference on future internet of things and cloud*, Rome, 24–26 August 2015, pp.231–239. New York: IEEE.

53. Chen H, Zhu X, Guo H, et al. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *J Syst Softw* 2015; 99: 20–35.

54. Kołodziej J, Khan SU, Wang L, et al. Security, energy, and performance-aware resource allocation mechanisms for computational grids. *Fat Gener Comput Syst* 2014; 31: 77–92.

55. Denizzi S, Ciopinski L, Pawinski G, et al. Cost optimization of real-time cloud applications using developmental genetic programming. In: *2014 IEEE/ACM 7th international conference on utility and cloud computing*, London, 8–11 December 2014, pp.774–779. New York: IEEE.

56. Sisu X. Real-time virtualization and cloud computing. *All Theses Dissert (ETDs)* 2014; 1366, https://openscholarship.wustl.edu/cgi/viewcontent.cgi?article=2366&context=etd.

57. Zhu X, Yang LT, Chen H, et al. Real-time tasks-oriented energy-aware scheduling in virtualized clouds. *IEEE Trans Cloud Comput* 2014; 2: 168–180.

58. Li X and Yin M. A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem. *Int J Prod Res* 2013; 51(16): 4732–4754.

59. Hussain H, Qureshi MB, Shoaib M, et al. Load balancing through task shifting and task splitting strategies in multi-core environment. In: *Proceedings of IEEE 8th international conference on digital information management*, Islamabad, Pakistan, 10–12 September 2013, pp.385–390. New York: IEEE.

60. Li J, Qiu M, Ming Z, et al. Online optimization for scheduling preemptable tasks on IaaS cloud systems. *J Parallel Distrib Comput* 2012; 72(5): 666–677.

61. Kołodziej J, Khan SU, Wang L, et al. Energy efficient genetic-based schedulers in computational grids. *Concurr Comput: Pract Exp* 2012; 27(4): 809–829.

62. Min-Allah N, Hussain H, Khan SU, et al. Power efficient rate monotonic scheduling for multi-core systems. *J Parallel Distrib Comput* 2012; 72(1): 48–57.

63. Liu S, Quan G and Ren S. On-line scheduling of real-time services with profit and penalty. In: *Proceedings of the 2011 ACM symposium on applied computing*, Taichung, Taiwan, 21–24 March 2011, pp.1476–1481. New York: ACM.

64. Kumar K, Feng J, Nimmagadda Y, et al. Resource allocation for real-time tasks using cloud computing. In: *Proceedings of 20th international conference on computer communications and networks (ICCCN)*, Maui, HI, 31 July–4 August 2011, pp.1–7. New York: IEEE.

65. Kim KH, Beloglazov A and Buyya R. Power-aware provisioning of virtual machines for real-time cloud services. *Concurr Comput: Pract Exp* 2011; 23(13): 1491–1505.

66. Awad A, El-Hefnawy N and Abdel-kader H. Enhanced particle swarm optimization for task scheduling in cloud computing environments. *Proc Comput Sci* 2015; 65: 920–929.

67. Kalaiselvi S and Kanimozhi Selvi CS. Hybrid cloud resource provisioning (HCRP) algorithm for optimal resource allocation using MKFCM and bat algorithm. *Wireless Pers Commun* 2020; 111: 1171–1185.

68. Jyoti A and Shrimali M. Dynamic provisioning of resources based on load balancing and service broker policy in cloud computing. *Clust Comput* 2020; 23: 377–395.
69. agileanswer.blogspot.com [Internet]. Cloud computing taxonomy; http://agileanswer.blogspot.com/2010/12/cloud-computing-taxonomy.html (accessed 28 April 2020).

70. Wu X, Deng M, Zhang R, et al. A task scheduling algorithm based on QoS-driven in cloud computing. *Procedia Comp Sci* 2013; 17: 1162–1169.

71. Tsai C, Huang W, Chiang M, et al. A hyper-heuristic scheduling algorithm for cloud. *IEEE T Cloud Comput* 2014; 2(2): 236–250.

72. Abrishami S, Naghibzadeh M and Epema DHJ. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Gener Comp Sy" 2013; 29(1): 158–169.

73. Malawski M, Juve G, Deelman E, et al. Algorithms for cost- and deadline constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Gener Comp Sy" 2015; 48: 1–18.

74. Sait SM, Bala A and El-Maleh AH. Cuckoo search based resource optimization of datacenters. *Appl Intell* 2015; 44(3): 489–506.

75. Abdullahi M and Ngadi MA. Hybrid symbiotic organisms search optimization algorithm for scheduling of tasks on cloud computing environment. *PLOS ONE* 2016; 11(6): 6–26.

76. Satish K and Reddy AR. Resource allocation in grid computing environment using genetic-auction based algorithm. *Int J Grid High Perf Comput* 2018; 10(1): 1–15.

77. Wang J, Liu A, Yan T, et al. A resource allocation model based on double-sided combinational auctions for transparent computing. *Peer Peer Newt Appl* 2018; 11(4): 679–696.

78. Frederic NZ and Yang Y. Effective task scheduling and dynamic resource optimization based on heuristic algorithms in cloud computing environment. *KSInt Trans Internet Inf Syst* 2017; 11(12): 5780–5802.

79. Singh AK, Dziurzanski P, Mendis HR, et al. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-manycore systems. *ACM Comput Surv* 2017; 50–52: 1–40.

80. Rouhifar M and Ravannemehr R. A survey on scheduling approaches for hard real-time systems. *Int J Comput Appl* 2015; 131(17): 41–48.

81. Stavrinides GL and Karatzas HD. Scheduling data-intensive workloads in large scale distributed systems: trends and challenges. In: Kolodziej J, Pop F and Dobre C (eds) *Modeling and simulation in HPC and cloud systems*, vol. 36. Cham: Springer, 2018, pp.19–43.

82. Zhang Q, Gui L, Hou F, et al. Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud RAN. *IEEE Internet Things* 2020; 7(4): 3282–3299.

83. Ahmed E, Ahmed A, Yaqoob I, et al. Bringing computation closer toward the user network: is edge computing the solution? *IEEE Commun Mag* 2017; 55(11): 138–144.

84. Sun H, Yu H, Fan G, et al. Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture. *Peer Peer Newt Appl* 2020; 13: 548–563.

85. Mahmud R, Koch FL and Buyya R. Cloud-fog interoperability in IoT-enabled healthcare solutions. In: *Proceedings of the 19th international conference on distributed computing and networking. ICDCN ’18*, Varanasi, India, 4–7 January 2018, pp.1–10. New York: ACM.

86. Zarrin J, Aguiar RL and Barraea JP. Decentralized resource discovery and management for future manycore systems. arXiv:1710.03649, 2017, pp.1–46.

87. Yang B, Guan L, Säntti T, et al. Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip. *Microprocess Micros" 2013; 37(4): 460–471.

88. Anagnostopoulos I, Tsoutsouras V, Bartzas A, et al. Distributed run-time resource management for malleable applications on many-core platforms. In: *Proceedings of the 50th annual design automation conference*, Austin, TX, 29 May–7 June 2013, pp.1–6. New York: IEEE.

89. Dong F and Akl SG. Scheduling algorithms for grid computing: state of the art and open problems. Technical Report No. 2006-504, 2006, pp.1–57, http://research.cs.queensu.ca/home/akl/techreports/GridComputing.pdf

90. Sabin G, Lang M and Sadayappan P. Moldable parallel job scheduling using job efficiency: an iterative approach. In: Frachtenberg E and Schwiegelshohn U (eds) *Job scheduling strategies for parallel processing*. JSSPP 2006. Lecture notes in computer science, vol. 4376. Berlin: Springer, 2007, pp.94–114.

91. Jiang Y, Shen X, Chen J, et al. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In: *Proceedings of the 17th international conference on parallel architectures and compilation techniques*, Toronto, ON, Canada, 25–29 October 2008, pp.220–229. New York: ACM.

92. Kim DW, Kim KH, Jang W, et al. Unrelated parallel machine scheduling with setup times using simulated annealing. *Robot Comp: Int Manuf* 2002; 18(3): 223–231.

93. Boillat JE and Kropf PG. A fast-distributed mapping algorithm. In: *Proceedings of the joint international conference on vector and parallel processing (CONPAR 90/ VAPP IV)*, Zurich, 10–13 September 1990, pp.405–416. Berlin: Springer.

94. Marcon C, Moreno E, Calazans N, et al. Evaluation of algorithms for low energy mapping onto NoCs. In: *Proceedings of IEEE international symposium on circuits and systems*, New Orleans, LA, 27–30 May 2007, pp.389–392. New York: IEEE.

95. Hoffmann H, Maggio M, Santambrogio MD, et al. SEEC: a framework for self-aware management of multi-core resources. Technical Report, Massachusetts Institute of Technology, Cambridge, MA, 2011, pp.1–14.

96. Shojaei H, Gharaeeian AH, Basten T, et al. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In: *Proceedings of 46th IEEE/ACM annual design automation conference*, San Francisco, CA, 26–31 July 2009, pp.917–922. New York: IEEE.

97. Schor L, Bacivarov I, Rai D, et al. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In: *Proceedings of the 2012 ACM international conference on compilers, architectures and synthesis for embedded systems*, Tampere, 7–12 October 2012, pp.71–80. New York: ACM.

98. Laplante PA. *Real-time systems design and analysis*. 3rd ed. Hoboken, NJ: Wiley, 2004.