IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 February 2024, accepted 3 April 2024, date of publication 12 April 2024, date of current version 19 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3388291

SURVEY

A Comprehensive Study of Load Balancing Approaches in Real-Time Multi-Core Systems for Mixed Real-Time Tasks

SHRUTI JADON, PRADYUMAN KAVEDI KANNAN, URMIL KALARIA, K. R. VARSHA, KARTHIK GUPTA, AND PRASAD B. HONNAVALLI, (Member, IEEE)
Department of Computer Science Engineering, PES University, Bengaluru 560085, India
Corresponding author: Shruti Jadon (shrutijadon@pes.edu)

ABSTRACT Real-time systems are becoming pervasive with the growing global connectivity and rising consumer demands. The need for real-time processing has become a crucial part of many business applications worldwide. A key factor that determines the time taken for an application to give out the result hinges on its ability to prioritize, manage, and execute real-time workloads. However, there are several difficulties and constraints connected with implementing tasks in a real-time context. This research study primarily focuses on load balancing for mixed real-time tasks on a multi-core system, one of the major challenges for executing real-time workloads. The purpose of load balancing is to distribute the load evenly among the processor(s) and maximize their utility while minimizing overall execution time. The goal of this paper is to present a critical analysis of existing load balancing techniques for both periodic and aperiodic tasks. The paper explores several factors including throughput, performance, migration time, response time, overhead, resource utilization, scalability, fault tolerance, power efficiency, and other variables that play a crucial role in assessing the efficacy of load balancing in real-time systems. The proposed has contributed in four folds. Firstly, the state-of-the-art of various load balancing algorithms are discussed followed by the architecture involved in real-time multi-core systems. Later, different load balancing based scheduling algorithms were compared on the basis of different schemas and metrics for algorithm evaluation is also provided. Finally, the paper also identifies areas that warrant further exploration or investigation, suggesting potential avenues for future research, and highlighting emerging trends or developments that may shape the field.

INDEX TERMS Scheduling algorithms, real-time systems, priority-driven algorithms, EDF (earliest deadline first), RM (rate-monotonic), LBPSA (load balanced partitioning and scheduling algorithm), homogeneous multi-core, heterogeneous multi-core, static algorithms, dynamic algorithms.

I. INTRODUCTION

Real-time systems are used in many areas of computer science. They play crucial roles in fields of networked multimedia, embedded automotive electronics, defense, space, etc. In a real-time system, the accuracy of the system's performance relies not only on the logical functioning of the computer but also on the specific moment when the results are generated. This implies that a real-time computer system must respond to stimuli from the controlled item (or the operator) within time period (deadline) imposed by its surroundings. However, there have been various challenges in the true realization of a real-time environment. Among those challenges, load balancing is an issue of prime concern. Load balancing in layman's terms means a way to distribute the load/workload across the nodes (essentially a group of computers/servers or a single computer/server). This is done to improve system output, resource utilization, and device performance. There are many load balancing algorithms devised for general as well as custom

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

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

real-time applications/servers that ensure maximum resource utilization while completing tasks on/before deadlines.

Real-time systems rely heavily on effective load balancing to avoid resource waste, ensure timely task completion, and deliver on strict deadlines. Researching this area is critical for optimal performance. This research study covers the various load balancing approaches and the system parameters that impact them.

A. SCOPE

Surveys [55], [56], [57] on multi-core and multi-processor systems have given insight into the performance, throughput issues, and their solutions in different areas of load balancing. This survey has highlighted relevant research articles, conferences, and business developments that examine the ways in which load balancing strategies should be modified for real-time multi-core systems, which need prompt task scheduling and little overhead. The parameters, such as throughput, response time, and least mean square error (LMSE) are used to gauge the efficiency of load balancing.

The paper discusses the difficulties and unresolved problems with load balancing for real-time multi-core systems, describing new trends in the field and future research directions.

B. MOTIVATION

- The existing research on load balancing strategies in real-time multi-core systems lacks a comprehensive analysis and evaluation of algorithms designed for handling a mix of real-time tasks, including periodic, aperiodic, and sporadic tasks. The goal of the paper is to give a general overview of the classification, scheduling mechanisms, and load balancing mechanisms used in real-time systems.
- The literature available primarily examines load balancing-based scheduling algorithms, their architecture, High-Level Design (HLD), and a single type of task set mostly periodic, while providing a quality assessment for their future development, however, scheduling of mixed real-time i.e. aperiodic along with sporadic tasks are missing.
- To help readers choose the best load balancing strategy for a given set of system requirements, this paper evaluates and categorizes different approaches based on their reliability, effectiveness, and performance metrics. The paper also groups load balancing-based algorithms according to various sub-classifications based on tasks and deadlines to provide a detailed overview and ideas of the algorithms chosen in various papers. It also aims to highlight the open issues and challenges considered in the various implementation techniques.
- The survey article seeks to close the research gap by offering a thorough analysis of numerous load balancing systems, their guiding concepts, algorithms, and effects on system performance.

C. CONTRIBUTIONS

This paper provides a thorough overview of various scheduling techniques and problems in load balancing approaches in real-time multi-core systems through various studies. Highlighting several security-related open problems and difficulties concerning the implementation and evaluation of algorithms, the contributions of this paper are listed below.

- The study discusses the development, background, and significance of various load balancing-based scheduling algorithms and their associated scheduling methods along with their advantages and disadvantages.
- In this study, load balancing-based scheduling algorithms are examined concerning the system architecture and its constituent parts, and a high-level design (HLD) for load balancing is provided. Through performance analysis and evaluation, the study highlights how these approaches are beneficial in getting the best system performance.
- The paper also analyses the taxonomy of LB-based algorithms and groups the load balancing-based scheduling algorithms according to various properties. The different algorithms taken into account include homogeneous and heterogeneous, static and dynamic, ML- and non-ML-based, and energy- and non-energy-efficient.
- Finally, it also highlights future areas for study into partitioning and implementation methods using numerous algorithms for load balancing strategies in real-time systems.

D. ORGANIZATION

The subsequent sections of this research paper is structured as follows. Section I discusses an introduction to the survey followed by motivation and contributions. The section explores the background study and related work done. In Section II, a background study on RT-systems classification, scheduling mechanisms, and load balancing mechanisms is discussed along with multi-processor and multi-core architecture. Section III classifies the load balancing-based scheduling algorithms on various properties and discusses the taxonomy of LB-based algorithms. The various algorithms are grouped according to the subsequent categories i) homogeneous and heterogeneous, ii) Static and dynamic, iii) ML-based algorithms (machine learning-based) iv) Energy-efficient Algorithms iv)Task migration v)Task Granularity. Section IV deals with the architecture and high-level designs used in the referred literature metric to collect the result and its calculation strategy used. Section V explains the working of load balancing based scheduling algorithms. Section VI explains the metrics used to evaluate algorithms. Section VII discusses the key challenges. Section VIII delves into the future prospects outlined for further exploration, while Section IX encapsulates the concluding remarks of the paper. Figure 1 shows the organization of the paper.

TABLE 1. A comparison of the existing load balancing algorithms.

Paper	Year	Objective	Algorithm used	Pros	Cons
[1]	2011	Perform distribution and make adjustments during runtime	EDF	Uses both centralized and decentralized approach	Message passing should be explored more
[2]	2011	Increase responsiveness	H-EDF and FF	Uses slack time to schedule aperiodic tasks	Only aperiodic tasks
[5]	2012	Maximizes core utilization bound	EDF	More efficient than partitioned scheduling	More complex to implement than partitioned scheduling
[20]	2013	Propose a new partitioned fixed-priority real-time scheduling algorithm for multi-core platforms	BDTD/TS (B-tree dependent task dispatch-ing/Task splitting) algorithm	Efficient and can be used to schedule a large number of tasks	Algorithm may not be able to schedule all tasks if the communica-tion overhead is too high
[4]	2019	Heuristic algorithm based on characteristics	LLF	Mixed-integer linear program-ming(MILP) problem	Only heterogeneous tasks
[7]	2019	Propose a new periodic task scheduling algorithm for homogeneous multi-core parallel processing systems	EDF and Multi-core alternate sort(MAS)	Outperforms the EDF algorithm in terms of the number of tasks that can be scheduled and the makespan	MAS algorithm is more complex than the EDF algorithm
[6]	2020	Propose a new method for analyzing the real-time performance of applications	Priority-based scheduling scheme	Flexible and efficient which can be used to analyze a variety of applications	Proposed method is not always able to guarantee that all applications will meet their deadlines
[3]	2021	Distribute tasks effectively	DM	Uses a feedback control algorithm	Cluster calculation

II. BACKGROUND STUDY AND RELATED WORK

The following section delves into the comprehensive background study and explores the existing body of related work, providing a contextual foundation for the current research endeavor.

A. REAL-TIME SYSTEMS

A real-time system [9] is a computer system that is programmed to respond to external events or stimuli within a specific time frame called a deadline. It is characterized by its ability to process and react to events in real- time. Real-time systems are crucial in applications where timing is critical and where failure to meet timing requirements can lead to severe consequences, example Embedded systems like anti-lock braking systems (ABS) and engine control, control systems like industrial automation systems, traffic light control, and robotic assembly lines, and information systems like financial trading and stock market systems.

B. CLASSIFICATION

The classification of real-time systems according to deadlines and task characteristics serves as a pivotal framework, categorizing these systems into distinct types based on the time constraints of tasks, enabling a clear understanding of their operational requirements and performance criteria.

1) DEADLINE

In real-time systems, tasks or processes have strict timing constraints that must be met to ensure correct operation. These timing constraints are usually divided into three categories:

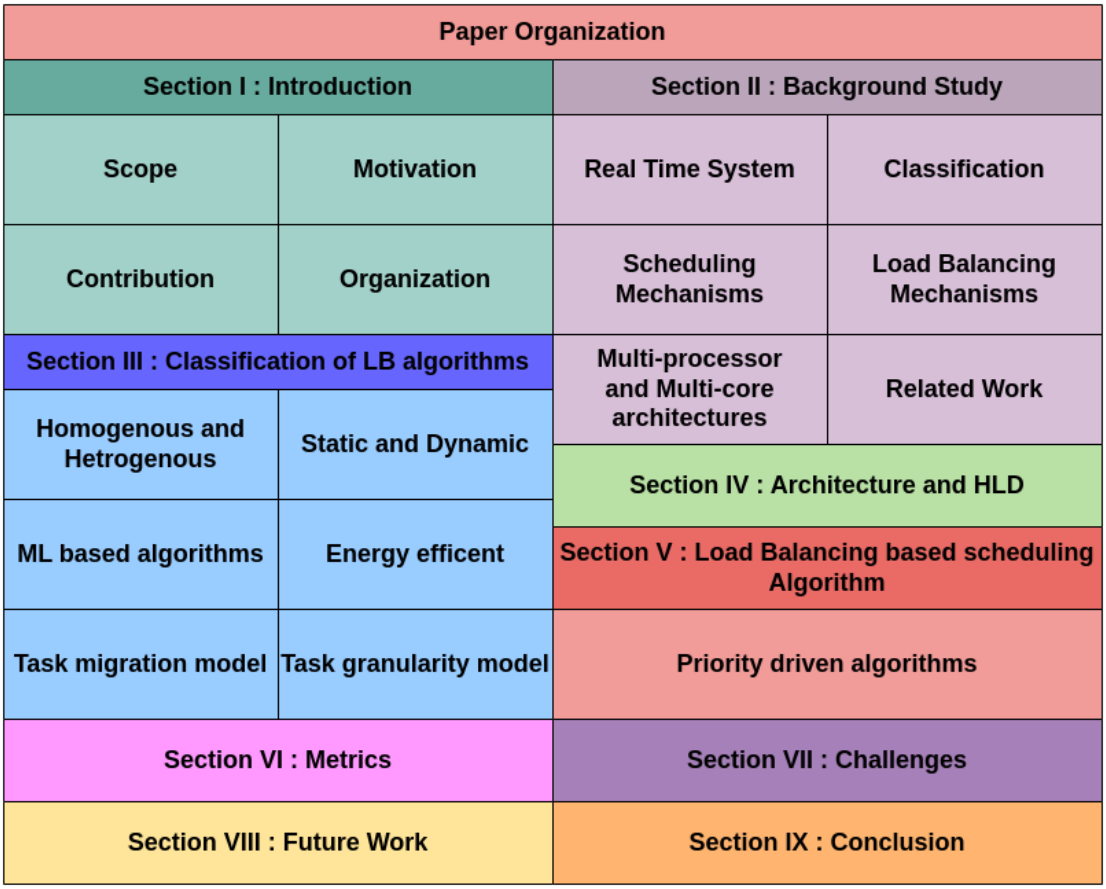


FIGURE 1. Paper organization.

1) Hard real-time: Missing a deadline or failing to reply within the given time limit is considered a system failure in hard real-time systems. Examples of hard real-time systems are flight control systems, airbag deployment systems in automobiles, and medical life support systems. Failure to respond in time can have severe consequences, including loss of life or property.
2) Soft Real-Time: Soft real-time systems also have timing constraints, but missing a deadline is not considered a catastrophic failure. The system can continue to operate, but the quality or efficiency of the system may degrade. Examples of soft real-time systems include multimedia streaming, online gaming, and real-time data analysis.
3) Firm Real-Time: Firm real-time systems are systems that operate with deadlines that are considered firm but not necessarily catastrophic if missed. In these systems, missing some deadlines might lead to a degraded quality of service or performance, but it does not cause severe system failure or danger. Examples of firm real-time systems are streaming services, telecommunication networks, and entertainment systems.

2) TASKS

Tasks in real-time systems are often classified based on their timing requirements and period. The time interval between successive arrivals or activations of a task is referred to as its period. Tasks can be categorized into three different types based on their period:

1) Periodic Tasks: Periodic tasks are recurring tasks that have a fixed and predictable period. These tasks repeat their execution at regular intervals. The arrival time of each instance of the task can be calculated based on the starting time of the first instance and the period. Periodic tasks are often used in real-time systems where certain operations need to be performed periodically. Examples include sensor data sampling at fixed intervals or regular control actions in industrial automation.
2) Aperiodic Tasks: Aperiodic tasks do not have a fixed or predictable period. They are triggered or activated by external events or stimuli, and their arrival times are not determined by a regular schedule. Aperiodic tasks typically have deadlines associated with them, specifying the maximum time within which they should be completed after activation. Examples of aperiodic tasks include handling user requests in a web server,

responding to interrupt events, or processing input from a user interface.

3) Sporadic Tasks: Sporadic tasks are a type of aperiodic task that have minimum inter-arrival times or minimum separation times between consecutive instances. While they don't have a fixed period, they have a constraint that ensures a certain minimum time between two successive activations. Sporadic tasks can be useful in scenarios where tasks need to be triggered by events with a minimum time separation. For example, in a multimedia system, a video frame rendering task may have a sporadic requirement to ensure that frames are not displayed too close together, causing flickering or visual artifacts.

C. SCHEDULING MECHANISMS

To meet the timing constraints of real-time systems, several techniques and mechanisms are employed, some of these techniques are:

- Task Scheduling: Real-time systems often employ scheduling algorithms to ensure that tasks are executed within their respective deadlines. Rate Monotonic Scheduling (RMS) [10], Earliest Deadline First (EDF) [11], and Deadline monotonic scheduling (DMS) [12] are three common scheduling techniques. Real-time scheduling algorithms like EDF and RMS are used in critical embedded applications to ensure reliable task execution. This paper discusses the above-mentioned algorithms in detail in the following sections.
- Priority-Based Execution: Tasks in real-time systems are assigned priorities based on their timing requirements. Higher-priority tasks are executed before lower-priority tasks, ensuring that critical tasks are processed in a timely manner.
- Predictable Execution: Real-time systems aim to provide deterministic behavior, where the execution time of tasks is known and bounded. This predictability ensures that timing requirements can be met reliably. Work done by Sayed et al. [15] and Schuh et al. [13] discusses some techniques for predictable execution.
- Interrupt Handling: Real-time systems use interrupts to handle external events. Interrupt handlers are designed to respond quickly to these events, minimizing the latency between event occurrence and system response.
- Priority Assignment: EDF (Earliest Deadline First) prioritizes tasks based on nearest deadlines dynamically, giving highest priority to those with earliest deadlines. RMS (Rate Monotonic Scheduling) statically assigns priorities according to task durations, favoring shorter tasks consistently. Similarly, DMS (Deadline Monotonic Scheduling) statically assigns priorities based on deadlines, with a focus on timely task completion rather than periods. Each method caters to different system requirements and constraints.
- Adaptability: EDF (Earliest Deadline First) dynamically adjusts priorities to accommodate changing task characteristics and deadlines during runtime, offering high adaptability. RMS (Rate Monotonic Scheduling) and DMS (Deadline Monotonic Scheduling) both statically assign priorities, limiting adaptability; however, DMS (Deadline Monotonic Scheduling) prioritizes meeting deadlines over task periods, distinguishing it from RMS (Rate Monotonic Scheduling).

D. LOAD BALANCING MECHANISMS

In real-time systems, load balancing mechanisms are essential for ensuring that the system's computational tasks are efficiently distributed across processing units to meet timing requirements and prevent overloading. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones. This increases the availability of your application. Load partitioning is one such essential mechanism through which load balancing can be achieved.

1) PARTITIONING

Partitioning [14], [15], [16], [17], [18] is a load balancing technique that involves dividing the system's workload into smaller, manageable partitions, each assigned to different processing units (e.g., cores, processors, or nodes). Work done by Gracioli et al. [19] discusses the implementation of partitioning-based scheduling algorithms on hard real-time tasks. The goal is to achieve a balanced distribution of computational tasks to ensure that no processing unit is overwhelmed while others remain underutilized. There are several ways partitioning can be implemented in real-time systems:

- Static Partitioning: In this approach, the workload is divided into fixed partitions during system initialization, and each partition is statically assigned to a specific processing unit. This method offers simplicity and determinism but might not adapt well to varying workloads. Examples of static partitioning algorithms are First Fit (FF), Best Fit (BF), and Worst Fit (WF). Table 2 shows the partitioning task set consisting of tasks along with their size. Each processor considered here are same and has a maximum capacity of 550 units.

TABLE 2. Static partitioning task set.

Task Id	Task Size
T_1	417
T_2	426
T_3	212
T_4	112

  - First Fit: It assigns the incoming task to the first available partition in the core that is large enough to accommodate it. It is simple and easy to implement. It has faster allocation since it scans memory sequentially. However, larger tasks might not find suitable space in the first available partition,

causing inefficient core utilization. Figure 2 shows the allocation strategy for the first fit algorithm.

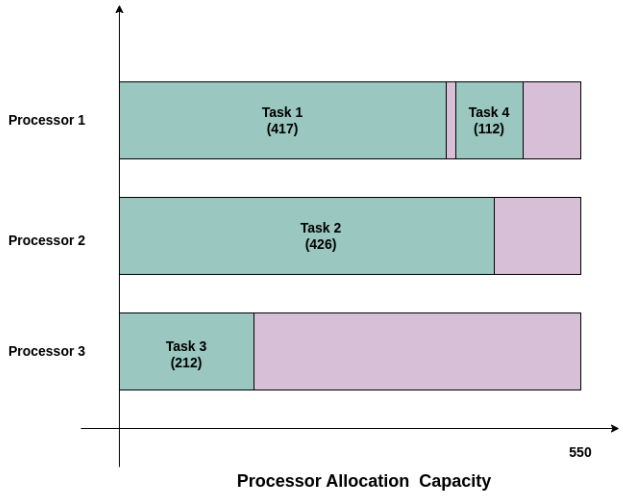


FIGURE 2. Working of first fit algorithm.

– Best Fit: It selects the smallest partition that is large enough to accommodate the incoming task. It scans the entire core to find the smallest utilization that can hold the task. It minimizes wastage by selecting the smallest adequate partition, reducing fragmentation. However, it is slower than First Fit as it has to traverse through the entire core to find the best-suited partition. Figure 3 shows the allocation strategy for the best-fit algorithm.

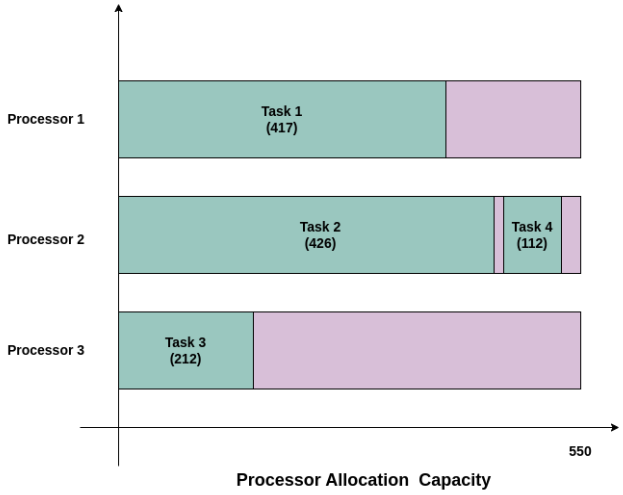


FIGURE 3. Working of best fit algorithm.

– Worst Fit: It allocates the incoming task to the largest available partition in the core that can accommodate it. It reduces the possibility of large leftover gaps, minimizing external fragmentation. However, it tends to leave behind smaller unused utilization, which can lead to more fragmentation over time. Figure 4 shows the allocation strategy for the worst-fit algorithm.

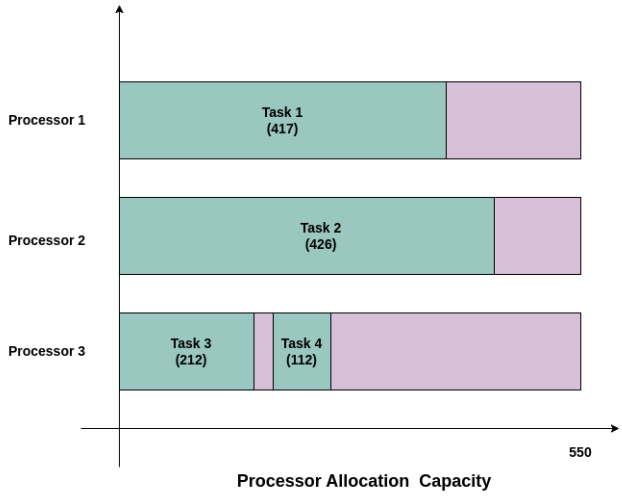


FIGURE 4. Working of worst fit algorithm.

- Dynamic Partitioning: Unlike static partitioning, dynamic partitioning allows the system to adapt and reassign partitions based on the current workload. The partitioning decisions can be made periodically or triggered by certain events or thresholds. This flexibility helps in handling dynamic workloads efficiently. Work done by Saranya and Hansdah [20] discusses some partitioning-based scheduling algorithms for Real-time multi-core systems.
- Load Estimation: To implement effective partitioning, the system needs to estimate the computational load of each task or partition accurately. Load estimation techniques can include monitoring CPU utilization, execution times, memory usage, or other relevant metrics.
- Load Balancing algorithms: Various load balancing algorithms can be used to determine how tasks are allocated to partitions. Examples include Round Robin, Weighted round robin, Least loaded, and least slack time algorithms. The algorithm used is determined by the unique needs and characteristics of the real-time system.

2) PARTITIONING STRATEGIES

Load balancing methodologies [15], [21], [22] in real-time systems often grapple with uneven workload distribution. Thus, partitioning becomes an important part of any load balancing algorithm. Table 3 shows a dynamic partitioning task set for real-time periodic tasks. Partitioning strategies are broadly divided into 3 categories: Consider the task set T as shown in Table 3. From the table, it can be seen that the tasks' utilization has been calculated using execution/period. Let us assume that there are 3 cores and tasks are added in the cores using the best-fit allocation method. The order of tasks is in increasing order of their deadlines will be T_4, T_5, T_1, T_6, T_2, and T_3.

TABLE 3. Dynamic partitioning task set.

Task Id	Exec Time	Period	Deadline	Utilization
T_1	2	6	6	0.33
T_2	1	12	12	0.083
T_3	2	12	12	0.17
T_4	2	4	4	0.5
T_5	3	4	4	0.75
T_6	2	7	7	0.29

The process starts with tasks T_4, T_5, T_1 and they will be allocated on cores C_1, C_2 and C_3 respectively. Next, when task T6 arrives it will be allocated on the core C_3 as it is the least utilized core. Similarly, tasks T_2 and T_3 are assigned on the core C_1. Finally, all the tasks in T are assigned on cores successfully as shown in Figure 5.

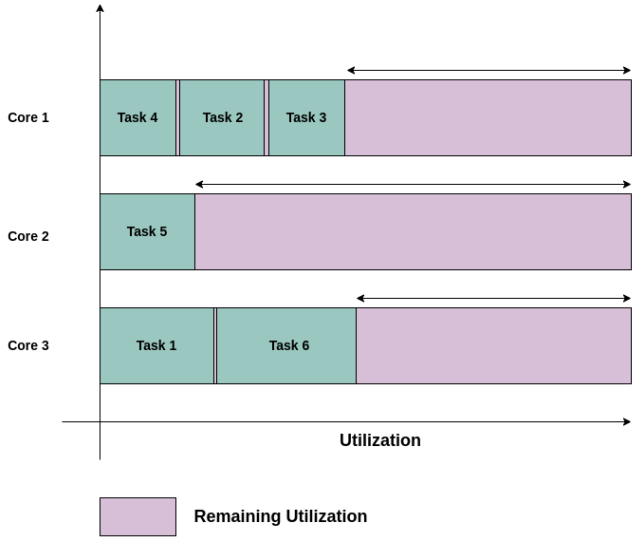


FIGURE 5. First allocation.

- Partitioned Partitioning Strategy: In this strategy, the workload or tasks are divided into separate partitions or subsets. Each partition operates independently and can be assigned to a system's specific processing units, nodes, or cores. Once assigned it stays in the same core/processor forever.
  Using the same example in this approach all the instances of a task will be executed on the same core. Therefore all the instances of tasks T_4, T_2, and T_3 will executed on core C_1, and the same works for the other cores. Finally, in Figure 6 it can be seen that the assignment of tasks to cores is fixed.
- Semi-Partitioned Partitioning Strategy: In a semi-partitioned strategy, while certain tasks or resources are divided into separate partitions, there are also shared resources or areas where components can interact or access shared data. This strategy is commonly used with tasks that are not completely independent.

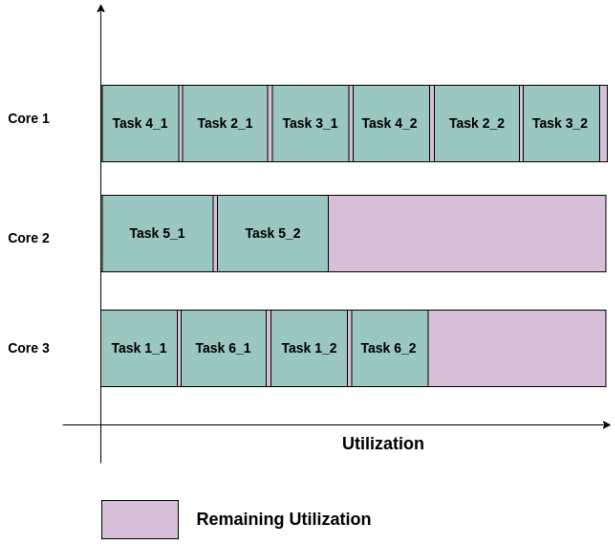


FIGURE 6. Working of partitioned algorithm.

Consider the same example with only one modification. Task T_7 has been added in task set T at arrival time t=6 with utilization of 0.50. The current utilization of cores C_1, C_2, and C_3 after the assignment of tasks T_1 to T_5 is 0.753, 0.75, and 0.62 respectively. Now, when T_7 arrives it will be assigned to C_3 as it is the least utilized core. The utilization of task T_7 is 0.50 but the remaining utilization of C_3 is 0.38 and task T_7 cannot fully be executed in C_3. Therefore, as a solution in a semi-partitioned approach, task T_7 is split in a way that 0.38 can be utilized on C_3 and 0.12 will be utilized on C_2 as it is the next least utilized core. The same is depicted in Figure 7.

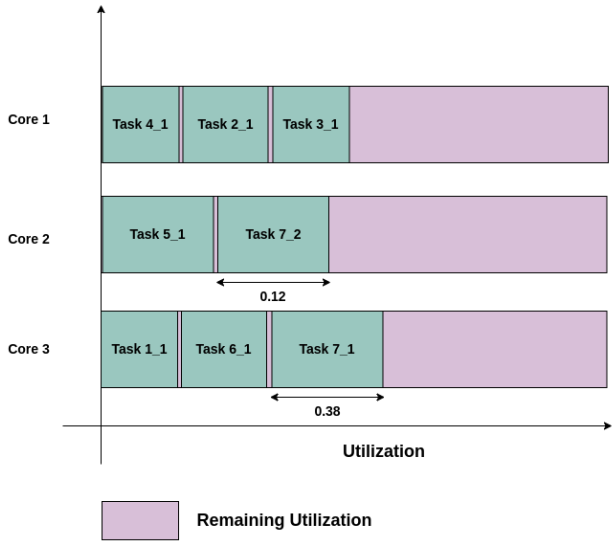


FIGURE 7. Working of semi-partitioned algorithm.

Consider a task set T as shown in Table 3, it can be seen that the task utilization is being calculated using

execution/period. Let us assume that there are 3 cores and tasks are added in the cores using the best-fit allocation method. The order of tasks in increasing order of their deadlines will be T_4, T_5, T_6, T_2, T_3.
The process starts with task T_4,T_5,T_1 and they will be allocated on cores C_1,C_2 and C_3 respectively. Next, when task T_6 arrives, it will be allocated on core C_3 as it is the least allocated on core. Similarly, tasks T_2 and T_3 are assigned on core C_1. Finally, all the tasks in T are assigned on cores successfully as shown in Figure 7.

- Global Partitioning Strategy: A global partition refers to a shared resource or a centralized control mechanism that oversees the load balancing process across partitions in a system. Example DAG-based partitioning. This approach is mostly dynamic i.e. partitioning is performed at runtime rather than compile time and is mostly used to partition aperiodic tasks.
  For the given example, in the global scheme the tasks are considered in a dynamic way. The tasks assigned on each core are not fixed. Using the same order T_4,T_5,T_1,T_6,T_2,T_3, the first instance T_4,T_5 and T_1 will be assigned to cores C_1, C_2 and C_3 respectively. The utilization of the 3 cores after this allocation will be 0.50, 0.75, 0.33. Now, when the first instance of T_6 comes it will be assigned to C_3. Similarly, T_2 and T_3 are assigned on C_1. The final core utilization can be observed in Figure 8.

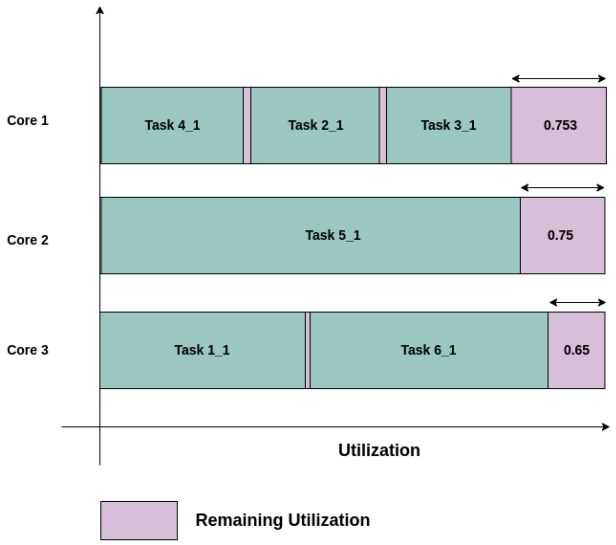


FIGURE 8. First allocation of global-partitioned.

For the next iteration of Task 4_2, it will be assigned to the core having the least utilization which is C_3 in this case. Similarly, in the second instance, the tasks will be assigned to different cores as shown in Figure 9.

Of all these techniques, scheduling is at the core of any load balancing system. Scheduling plays a vital role in load balancing (LB) because it determines how tasks are assigned to processing cores in a multi-core system. Effective scheduling ensures the computational workload is distributed optimally, considering factors such as task characteristics, resource availability, and system performance requirements. We will delve deep into various scheduling techniques and algorithms in later sections of this paper.

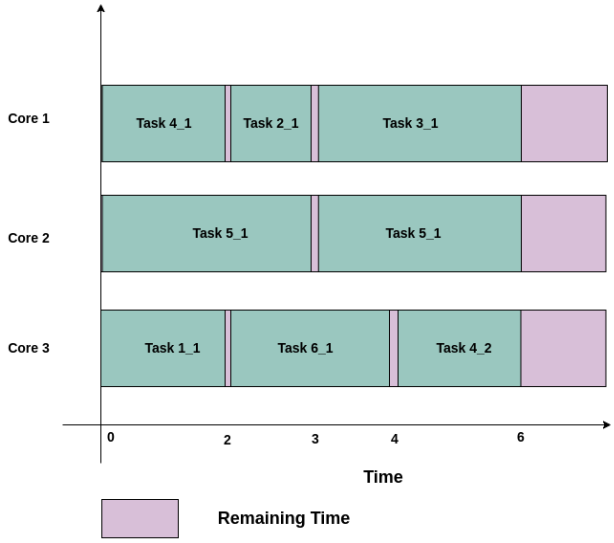


FIGURE 9. Working of global-partitioned algorithm.

E. MULTI-PROCESSOR AND MULTI-CORE ARCHITECTURE

A multi-processor architecture [23], [24], [25], [26], [27], [28], [29] is a system that has multiple processors. Each processor has its own independent instruction set and can execute instructions simultaneously. This allows the system to handle more tasks at the same time, which can improve performance. Figures 10 and 11 show the architecture of multi-processor systems and multi-core systems respectively. There are two main types of multi-processor architectures: symmetric multiprocessing (SMP) [30] and non-uniform memory access (NUMA) [31].

- SMP is a system where all processors have equal access to memory and other resources. This is the simplest type of multi-processor architecture, but it can be difficult to scale to a large number of processors.
- NUMA is a system where each processor has its own local memory, and access to remote memory is slower. This can improve performance for applications that are memory-intensive, but it can also make it more difficult to develop and debug software

A multi-core architecture [32], [33], [34], [35], [36] is a system that has multiple cores on a single processor. Each core is a complete processor, with its own instruction set and execution unit. This allows the system to handle more tasks at the same time, which can improve performance. Figure 11 shows the architecture of multi-core systems. Multi-core architectures are becoming increasingly common,

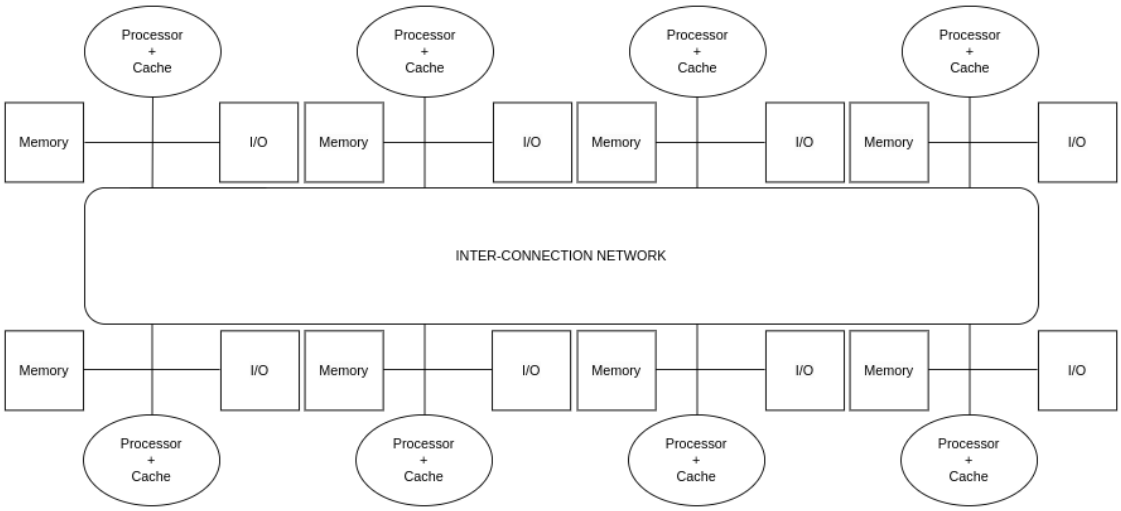


FIGURE 10. Multi-processor architecture.

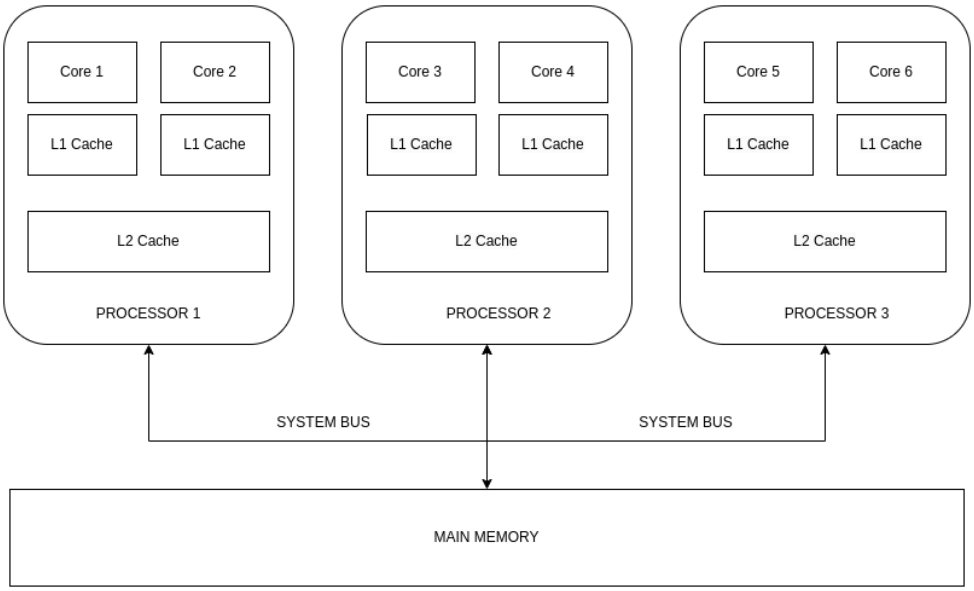


FIGURE 11. Multi-core architecture.

and they are now found in most personal computers and servers. They offer many advantages over traditional single-core processors, including:

- Increased performance: Multi-core processors can execute multiple instructions at the same time, which can significantly improve performance for tasks that can be parallelized.
- Reduced power consumption: Because multi-core processors may share resources like the instruction cache and memory controller, they can be more efficient than single-core CPUs.
- Improved scalability: Multi-core processors can be easily scaled to handle more tasks, by adding more cores.

The main difference between multi-processor and multi-core architectures is that a multi-processor system has multiple independent processors, while a multi-core system has multiple cores on a single processor. Thus in multi-core architecture, communication costs are minimal compared to multi-processor architectures, where significantly higher communication costs are required. This difference has several implications:

- Performance: Multi-processor systems can typically achieve higher performance than multi-core systems because they have more independent processors. However, multi-processor systems can also be more expensive and complex.

- Power consumption: Multi-core systems typically consume less power than multi-processor systems, because they have fewer independent processors. This is because multi-core processors can share resources such as the instruction cache and memory controller.
- Scalability: Multi-processor systems are more scalable than multi-core systems because they can be easily scaled by adding more processors. However, multi-processor systems can also become more complex and difficult to manage as the number of processors increases.

The evolution of multiprocessor and multi-core architectures has significantly advanced computing capabilities, enabling higher performance, efficiency, and scalability in modern systems. Through the exploration of various design paradigms, such as symmetric multiprocessing (SMP) and heterogeneous architectures, they play a crucial role in addressing the increasing demands of complex computational tasks. As technology continues to progress, further innovations in hardware design, memory hierarchies, and software optimization will continue to shape the landscape of multiprocessor and multi-core architectures, driving the development of more powerful and versatile computing systems that cater to diverse application requirements.

F. RELATED WORK

The studies done concerning scheduling algorithms for real-time systems aim to find the optimal algorithm that caters to the system's needs and is efficient. This section provides the typical details of some of these works.

Tan et al. [37] work delves into Hybrid scheduling algorithms tailored for soft real-time tasks within a Homogeneous multi-core system. In this context, a real-time application undergoes partitioning into independent and parallelizable tasks, depicted by a Directed Acyclic Graph (DAG) illustrating the precedence relationships among these tasks. load balancing allocation Algorithms, specifically MDFA (Most Demand First Algorithm) and IFA (Idlest First Algorithm), are employed by the authors. The proposed scheduling scheme comprises two levels, with the first level incorporating sporadic servers. Each core encompasses multiple sporadic servers, each equipped with an assigned CPU budget (c), a period (p), a ready queue of tasks, and a scheduler utilizing EDF (Earliest Deadline First), RM (Rate Monotonic), or Time-sharing algorithms based on the task's characteristics. At the lower level, an OS scheduler, preferably an RM scheduler, oversees all top-level sporadic servers. The OS scheduler's responsibilities include refreshing each sporadic server after its period (p), reallocating the budget, and employing the RM algorithm to schedule tasks. Execution of tasks from the ready servers continues until the CPU budget of the sporadic server is depleted or high-priority tasks are introduced.

Hu et al. [5] conducted research on a multi-core architecture based on Network-on-Chip (NoC). In their work, the authors introduce a semi-partitioned strategy for effectively scheduling periodic intensive real-time workloads, utilizing cache migration. To minimize the costs associated with online cache migration, the authors statically partition as many processes as possible onto cores. To improve the schedulability of the task set, the remaining jobs are allowed to move among a predetermined group of cores in a predefined manner. The scheduling strategy employed is Earliest Deadline First (EDF) on each core, maximizing core utilization. Tasks on a specific core can selectively lock a portion of their memory lines in the core's private cache, enhancing the predictability of multi-task execution on a single core. In the case of a migrating job, locked lines associated with the task are transferred and then re-locked on the target core. Despite the potential for tasks to be moved during a job's execution, the use of locked cache migration ensures predictable migration overhead. The reliable sharing of the on-chip communication infrastructure is achieved through a time-division multiplexed (TDM) arbitration method.

The work done by Tang et al. [2] on heterogeneous real-time systems, the authors focus on scheduling soft aperiodic jobs alongside periodic ones with strict deadline requirements. By initially scheduling periodic jobs offline and then dynamically arranging aperiodic tasks in the remaining resource slack time, we present a way to enhance aperiodic task responsiveness. Results indicate that slack distribution has a significant impact on how well aperiodic task scheduling performs. The work done by Acharya and Mahapatra [38], Jeon et al. [39], and Kumar [40] focus on slack management and distribution techniques. The proposed slack distribution techniques have been considered with EDF and rate-based scheduling schemes.

Later Geng et al. [41] provides a thorough examination of the key contributing parameters that affect multi-core load balancing. A broad dynamic load balancing paradigm based on CMP (Chip Multi-Processor) was suggested.

The work done by Wu et al. [8] gives a unique scheduling approach for real-time multi-core computers was suggested to balance computation demands and save electricity. For their power and deadline-aware multi-core scheduling (PDAMS), the article examines numerous criteria, a unique component, and a task deadline. The suggested method can cut energy usage by up to 54.2%, according to experimental data.

El Kabbany et al. [1] present an alternative method outlining a dynamic approach to job scheduling and load balancing within message-passing systems, emphasizing its application independence. This approach introduces a Dynamic Load Balancing Technique for Real-Time Applications (DAG-DLBR), utilizing a Directed Acyclic Graph (DAG) as its foundation. The technique is designed to operate dynamically, adapting to any fluctuations in load that may occur during runtime.

Also in the work done by Tan et al. [3], provides an algorithm that considers both the load of individual nodes and the network communication latency to distribute tasks

effectively. The load balancing server monitors the load of all nodes in the cluster and calculates the current load and latency of each node. Based on this information, the server determines which node is best suited to handle the incoming task and assigns the task to that node. The load balancing server uses a feedback control algorithm to adjust the workload distribution based on the current load and latency of each node. This ensures that no node is overloaded while maintaining low latency and optimal resource utilization. The test results demonstrate that load balancing can be executed automatically when there is an imbalance in the system's load so that all of the system's processors can work to maximize throughput and resource utilization. Additionally, load balancing results in very little system lag.

Baital and Chakrabart [4], introduces a task model that captures both the real-time and energy constraints of the tasks. Based on this task model, the authors formulate the scheduling problem as a mixed-integer linear programming (MILP) problem. The objective of the MILP problem is to minimize energy consumption while meeting the real-time constraints of the tasks. To solve the MILP problem efficiently, the authors propose a heuristic algorithm that partitions the tasks into clusters based on their characteristics and schedules the tasks within each cluster using an energy-aware scheduling algorithm. The proposed approach also takes into account the effect of temperature on the energy consumption of the system.

In the approach given by Delgado and Choi [6], sheds fresh light on the performance of Linux's real-time extensions, Xenomai and RT-Preempt, on a homogeneous multi-core CPU. They identify distinct multi-core installations and analyze their trade-offs, as determined via experimental scheduling delay evaluation. Then, to find the optimal multi-core deployment, they suggest a statistical approach based on a version of the chi-square test. The periodicity of the best multi-core deployment is used to validate its practicality.

The work done by Xiao et al. [7], proposes a new periodic job scheduling approach for homogeneous multi-core parallel processing systems. The approach is based on the earliest deadline first (EDF) algorithm and accounts for task communication overhead. The program initially arranges the assignments according to their deadlines. The work is then assigned to each CPU one by one, beginning with the processor with the least load. When a job is allocated to a processor, the algorithm determines if the communication overhead between the task and the tasks already operating on the processor can be tolerated. The job is allocated to the processor if the communication overhead can be tolerated. If not, the work is passed on to the next processor.

The suggested approach is tested by scheduling a collection of periodic jobs on a homogeneous multi-core parallel processing machine. The findings indicate that the algorithm can schedule and complete all jobs on time. In terms of the number of jobs that may be scheduled and the makespan, the method surpasses the EDF algorithm.

Another work done by Wu et al. [8], proposes a new partitioned fixed-priority real-time scheduling algorithm for a homogeneous multi-core platform. In this algorithm, periodic tasks are split into two types of tasks: independent tasks and dependent tasks. Independent tasks are scheduled in each processor by the EDF algorithm. For dependent tasks, are first partitioned into a series of sequential jobs and then scheduled in each processor by the EDF algorithm.

From all of these works, it may be concluded that there is still a need to have an efficient multi-model scheduling scheme to incorporate all types of tasks be it periodic, aperiodic, or sporadic for allocating these tasks to real-time multi-core systems maintaining the deadline constraints.

III. CLASSIFICATION OF LB-BASED ALGORITHMS

Load balancing algorithms [48], [49], [50], [51], [52] are methods used in computer systems and networks to distribute tasks or workloads across multiple resources. They're crucial for ensuring that each resource (like servers, CPUs, or network links) in a system handles an equitable amount of work, preventing some from becoming overloaded while others remain underutilized.

Load Balancing algorithms can be classified in different ways such

- Homogeneous and Heterogeneous algorithms
- Static and Dynamic algorithms
- ML-based algorithms
- Energy Efficient algorithms
- Task Migration models
- Task Granularity models

A. HOMOGENEOUS AND HETEROGENEOUS

Homogeneous load balancing algorithms are designed to distribute computational or network loads evenly across multiple resources that are identical or similar in terms of their processing capacity. Web servers, Grid Computing, and Data centers are some examples that use homogeneous load balancing algorithms.

Heterogeneous load balancing algorithms are designed to distribute computational or network loads across resources that have different capacities or capabilities. Unlike homogeneous load balancing, where resources are assumed to be identical or very similar, heterogeneous load balancing algorithms take into account variations in processing power, memory, or other attributes among the available resources. Examples are Particle Swarm Optimization, Ant Colony Optimization, Genetic Algorithm, and Neural Networks. This type of algorithm are used in High-Performance Computing (HPC) clusters, Edge Computing environments, and Scientific Computing.

Table 4, categorizes research papers based on whether they focus on homogeneous or heterogeneous systems in the context of real-time load balancing. Each entry provides information on the publication year, the type of system targeted, the approach taken, and any drawbacks associated with the respective approach. This classification

TABLE 4. Classification of papers based on homogeneous and heterogeneous.

Paper	Year	Homogeneous / Heterogeneous	Approach	Drawbacks
[37]	2010	Homogeneous	Hybrid scheduling method using two-level scheduling scheme.	Future allow on heterogeneous cores
[2]	2011	Heterogeneous	Soft aperiodic tasks and hard periodic tasks	Aperiodic tasks have soft deadlines
[46]	2012	Homogeneous	Scheduling algorithm that takes into account the load balance of real-time periodic tasks	Only periodic
[5]	2013	Homogeneous	Under Locked Cache Migration, Semi-Partitioned Hard-Real-Time Scheduling	Only on homogeneous machines
[8]	2013	Homogeneous	On a homogeneous multi-core platform, a partitioned fixed-priority real-time scheduling based on dependent tasks-split is presented.	Doesn't incorporate other dependent factors like resource sharing and cache overhead.
[45]	2015	Homogeneous	Discusses the criteria and techniques for evaluating schedulability tests for real-time scheduling algorithms and just periodic jobs.	Only Periodic Tasks
[47]	2018	Homogeneous	Optimizing scheduling overhead in such processors	Performance is measured in terms of temperature.
[4]	2019	Heterogeneous	Examine the existing scheduling techniques in a heterogeneous multi-core system and devise a method to adapt the homogeneous system model to a heterogeneous scheduling architecture.	Issues related to resource sharing and performance parameters are not discussed.
[6]	2020	Homogeneous	Insights into the real-time performance of real-time extensions of Linux, namely Xenomai and RT-Preempt	Only homogeneous Processors
[43]	2020	Heterogeneous	Uses particle swarm optimization	Complexity, Extra overhead

aids in understanding the scope and limitations of each paper with respect to whether they address homogeneous or heterogeneous systems and highlights potential areas for improvement or further research.

B. STATIC AND DYNAMIC ALGORITHMS

Static load balancing is a technique where the assignment of tasks to cores is determined before the execution of the program and remains fixed throughout its execution. In other words, the workload distribution is decided during the program's initialization or at compile-time. The tasks are divided among the cores in a way that is expected to achieve good load balance based on prior knowledge of the application's characteristics or historical data. These are mainly used in batch processing systems, embedded systems, and telecommunication networks.

Dynamic load balancing is a technique where the assignment of tasks to cores is made during runtime. As the program

TABLE 5. Classification of papers based on static and dynamic.

Paper	Year	Static / Dynamic	Approach	Algorithm/Model
[1]	2011	Dynamic	Dynamic Load Balancing Algorithm Based on DAG for Real-Time Applications	(DAG-DLBR)
[37]	2010	Static	Hybrid scheduling method using two-level scheduling scheme.	Rate Monotonic/Sporadic server
[41]	2010	Dynamic	The dynamic load balancing scheduling problem is represented by this model as a quintuple <E, T, L, S, C>.	Model
[5]	2013	Static	Semi-Partitioned Hard-Real-Time Scheduling Under Locked Cache Migration	Cache locking and locked cache migration
[8]	2013	Static	On a homogeneous multi-core platform, a partitioned fixed-priority real-time scheduling based on dependent tasks-split is presented.	BDTD/TS (B-tree Dependent Task Dispatching/Task Splitting)
[44]	2016	Static	Discusses the criteria and techniques for evaluating the performance of real-time scheduling algorithms' schedulability tests.	Model
[4]	2019	Dynamic	Examines the existing scheduling techniques in a heterogeneous multi-core system and devise a method to adapt the homogeneous system model to a heterogeneous scheduling architecture.	Model
[3]	2021	Dynamic	A centralized global dynamic multi-core load balancing approach with low latency, high performance, and high real-time for the microkernel operating system.	Kernel optimised algorithm
[43]	2020	Dynamic	Uses particle swarm optimization	Particle swarm

runs, the system continuously monitors the workload on each core and redistributes tasks to achieve a more balanced distribution. The decision-making process is based on real-time measurements of the current system state and the characteristics of the tasks. They are mainly used in cloud computing, online transaction processing (OLTP) systems, and content delivery networks.

Table 5 provides a comprehensive overview of the research landscape in load balancing, classifying nine papers published between 2010 and 2021 based on their methodological approach as either static or dynamic. This categorization highlights the diverse strategies employed to address load balancing challenges in real-time environments.

Static load balancing methods, as reflected in papers such as [5], [6], [8], [29], [37], [45], and [47], establish task allocation schemes that remain fixed throughout system operation. These approaches often rely on task characteristics and system parameters to determine optimal task assignments.

However, their static nature may limit their adaptability to dynamic system conditions and unpredictable workloads.

Dynamic load balancing methods, exemplified by papers [1], [3], [4], [41], address the limitations of static approaches by proactively adjusting task allocations in response to real-time changes. These methods typically employ monitoring and feedback mechanisms to capture system dynamics and adapt task scheduling accordingly. While dynamic methods offer greater flexibility, they may introduce additional computational overhead due to the need for continuous evaluation and adaptation.

The papers used to show a model that uses dynamic load balancing like the work done by Geng et al. [41] and Baital and Chakrabart [4]. The other papers use dynamic algorithms [1] which uses a Directed acyclic graph for load balancing it represents tasks as nodes and then makes them like a graph with directed edges. They consider factors like task dependencies, computational requirements, and communication overhead to minimize idle time and maximize resource utilization. The work done by Biswas and Kuila [43] uses particle swarm optimization which is a dynamic ML approach.

C. ML AND NON-ML BASED ALGORITHMS

Machine learning algorithms form a subset of artificial intelligence, enabling computers to learn from data without requiring explicit programming. These algorithms employ statistical approaches to identify patterns, make predictions, and derive insights from data. Based on their learning technique, ML algorithms are classified into several forms, including reinforcement learning, neural networks, decision trees, random forest, and genetic algorithms. These algorithms use ML to predict patterns of tasks from previous data and predict what the next tasks will be like and when they will arrive. With this information, it adjusts the system to take and manage load for efficient allocation. This type of algorithm is mainly used in distributed stream processing systems, predictive maintenance in data centers, and autoscaling in microservices architectures. Some well-known machine-learning techniques that are used in load balancing are:

- Reinforcement Learning (RL):
  - Q-learning, Deep Q-learning: RL algorithms can learn optimal scheduling policies by interacting with the system environment, and making decisions based on rewards and penalties.
  - Actor-Critic Models: Combining policy-based and value-based methods for more stable learning.
- Neural Networks:
  - Deep Neural Networks (DNN): DNNs can be trained to predict future workload patterns and dynamically adjust scheduling policies based on these predictions.
  - Convolutional Neural Networks (CNN): Used for workload forecasting and pattern recognition.
- Genetic Algorithms (GA):
  - Genetic algorithms are used to evolve scheduling policies by representing them as chromosomes and iteratively evolving and selecting the best policies based on fitness criteria.
- Fuzzy Logic:
  - Fuzzy logic-based controllers are used for adaptive scheduling, allowing for more nuanced decision-making based on vague or imprecise input data.
- Bayesian Networks:
  - Bayesian models can be utilized for probabilistic reasoning about future workloads and optimal scheduling decisions.
- Swarm Intelligence:
  - Algorithms inspired by swarm behavior, such as ant colony optimization or particle swarm optimization, can be employed for load balancing and scheduling.

These algorithms can adapt to changing workloads, optimize resource utilization, and improve the overall efficiency of multi-core systems by dynamically scheduling tasks across cores while considering real-time constraints and system characteristics.

The research conducted by Biswas and Kuila [43] employs particle swarm optimization, a machine learning approach that iteratively explores candidate solutions to optimize a given problem based on their quality. This method involves a population of candidate solutions referred to as particles, which are systematically navigated within the search space.

Non-ML algorithms are traditional algorithms that use predefined rules, logical steps, or mathematical computations to perform a specific task. They rely on explicit programming and do not require data-driven training or learning from examples. Some examples of non-ML algorithms include Round Robin, Slack stealing algorithm, etc.

D. ENERGY EFFICIENT AND NON ENERGY EFFICIENT

Energy-efficient load balancing in multi-core systems refers to the management of workload distribution across cores in a way that minimizes energy consumption while maintaining performance. This type of algorithm is mostly used in wireless sensor networks (WSNs), mobile edge computing (MEC), and smart grids. The work done by Reddy et al. [63] describes how multi-core systems greatly boost performance while consuming less energy. The goal is to achieve a balance between computational tasks and the energy consumed by each core. This can involve techniques such as dynamic voltage and frequency scaling (DVFS) to adjust the operating frequency and voltage of cores based on the workload to save power. DVFS works on the basis of dynamically varying a processor's voltage and clock frequency in response to workload. It is possible to save power and lessen heat

emissions by lowering the voltage and frequency when the workload is light. On the other hand, higher voltage and frequency can be used to boost performance as the workload grows. There are two main approaches for energy-efficient scheduling viz. DVFS and Power capping.

1) DYNAMIC VOLTAGE AND FREQUENCY SCALING (DVFS)

DVFS(Dynamic Voltage and Frequency Scaling) [53] is a technique that uses frequency/voltage scaling to cap the power requirements of the application. The work done by Madala and Chellamani [64] examines energy efficiency achieved in embedded controllers through DVFS. The implementation of DVFS can be done at both Software and Hardware levels. Hardware-level implementation promises a very 20-45% improvement in responsiveness and about 1/30 in latency. However, there are drawbacks to using the DVFS. The software-level implementation had the information of the current and desired performance state, and this information was used directly to change frequency and voltage, however with hardware-level implementation this information will no longer be available and changes need to be made to overcome this problem. Thus in practical implementation power capping is more widely used than DVFS. Figure 12 shows the workload patterns before and after applying the DVFS algorithm. With DVFS time required to complete the tasks increased but the deadlines were not missed thus it is power efficient.

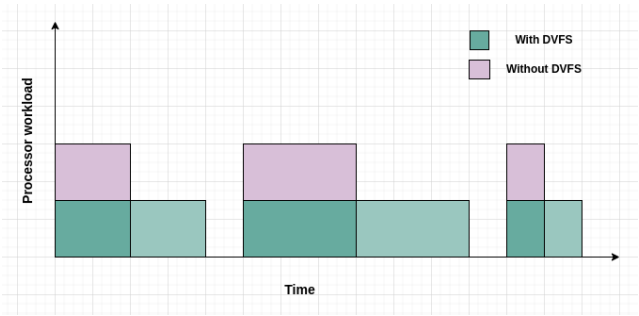


FIGURE 12. Workload pattern in DVFS.

2) POWER CAPPING

Power Capping [54] is an adaptive control theory to meet application/task performance goals by tuning hardware power caps. It operates on applications without prior knowledge of their specific performance/power trade-offs. The system employs a Kalman filter to adjust control for non-linearity in the power cap/performance relationship. Additionally, adaptive gain limits are introduced to avoid over-allocating power when applications or tasks cannot achieve further speedup. In other words, if a workload's performance doesn't improve with expanded power limits, the power capping mechanism refrains from allocating additional power. Figure 13 shows the control design for the Power capping technique using CoPPer.

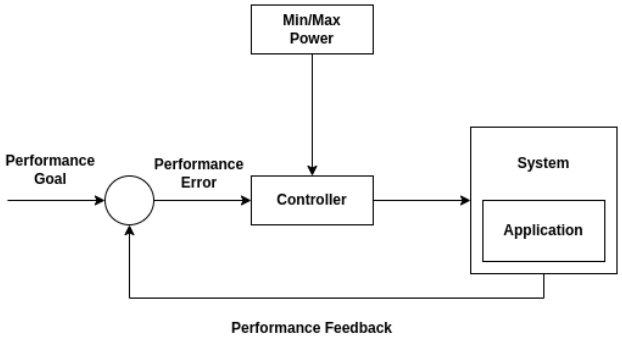


FIGURE 13. Feedback control design in CoPPer.

Non-energy efficient load balancing in multi-core systems refers to load balancing approaches that prioritize performance and computational efficiency without considering energy consumption. These methods might focus on maximizing throughput and minimizing task completion times without necessarily optimizing power usage. Table 6, provides a complete classification of papers based on Energy efficiency in multi-core systems.

E. TASK MIGRATION MODELS

In pre-emptive migration, tasks are allowed to be moved from one processor to another before their completion. This model is proactive and aims to optimize the distribution of computational load in real-time. It is particularly beneficial in dynamic environments where the workload on processors may change dynamically. Pre-emptive migration enables the system to adapt to varying workloads, preventing the occurrence of performance bottlenecks and ensuring a more balanced utilization of resources.

Non-pre-emptive migration involves moving tasks only after they have been completed or when they are in a safe state to be moved. This approach prioritizes the completion of tasks without interruption. Non-pre-emptive migration is suitable for scenarios where the overhead associated with task migration during execution is considered undesirable or when tasks have specific dependencies that must be maintained. This model ensures the stability of ongoing computations but may be less agile in responding to dynamic changes in workload.

F. TASK GRANULARITY

Fine-grained load balancing involves breaking down computational tasks into smaller units or sub-tasks and distributing them across processors. This approach allows for more frequent load adjustments and is well-suited for workloads with varying task sizes and execution times. Fine-grained load balancing minimizes idle times on individual processors, promoting efficient resource utilization. It is effective in scenarios where the computational workload is composed of numerous small, independent tasks.

Coarse-grained load balancing, in contrast, entails distributing larger tasks or entire jobs to processors. This

TABLE 6. Classification of papers based on energy and non energy efficient.

Paper	Year	Energy efficient / Non energy efficient	Approach	Efficiency
[61]	2010	Non Energy Efficient	Genetic algorithm based scheduling	Inefficient distribution of load more heat will be generated by the overloaded servers
[42]	2014	Energy efficient	Power and deadline-aware multi-core scheduling	54.2%
[59]	2016	Energy efficient	Green computing solution	Energy-Aware Task Scheduler using Genetic Algorithm
[47]	2018	Energy efficient	Optimizing scheduling overhead in such processors	PAES 0 % NPGA 13.3 % NSGA 23.4 % MOGA 23.4 %
[60]	2018	Energy Efficient	Energy-aware scheduling algorithm for time-constrained workflow tasks	modified DVFS.
[62]	2019	Non-energy efficient	Weighted primitive scheduling	do not effectively take energy consideration into the picture.
[58]	2023	Energy efficient	Energy-aware scheduling methods used in a modern HPC environment	Dynamic voltage and frequency scaling (DVFS), power capping

approach is suitable when tasks are substantial in size and have relatively predictable execution times. Coarse-grained load balancing helps reduce the overhead associated with frequent task migration, as the tasks being moved are more significant. This model is often employed in scenarios where the cost of migrating tasks is higher compared to the potential benefits of fine-grained load balancing.

IV. ARCHITECTURE AND HIGH-LEVEL DESIGN

The capability to integrate millions of transistors onto a single chip facilitates the creation of increasingly intricate designs. Challenges such as diminishing gains in instruction-level parallelism, relatively constant memory latency, heightened heat generation, the intricate nature of designing a single core with a substantial number of transistors, and the associated financial costs have led to discouragement of the development of more complex uni-processors. Consequently, the current trend favors the creation of chips with multiple processor cores, known as multi-core processors. In this architecture, two or more independent cores are consolidated into a unified package, forming a single integrated circuit (IC). These processors share the same interconnect with the rest of the system, and each core incorporates its own optimizations, including superscalar execution, pipelining, and multi-threading. Broadly, two types of multi-core architecture classify all real-time systems.

- Homogeneous multi-core: This type of architecture contains multiple cores that are identical in terms of their instruction set architecture (ISA) and functionality. Each core within the processor package is capable of independently executing tasks and operating as a separate processing unit.
- Heterogeneous multi-core: This type of architecture contains multiple cores that are not identical in terms of their architecture, instruction set, or performance characteristics. Unlike homogeneous multi-core processors where all cores are identical, heterogeneous multi-core processors have different types of cores within the same processor package.

Most of the related works classify their algorithms as suitable for one of these architectures.

Work done by Tan et al. [37], uses a homogeneous multi-core architecture, where a two-level scheduling scheme is discussed. The dispatched work is scheduled using sporadic servers at the top level in accordance with a scheduling policy. A rate-monotonic server at the bottom level is used to maintain and schedule jobs for top-level erratic servers.

Another work done by Hu et al. [5], uses a heterogeneous multi-core, where they are scheduling periodic tasks offline or statically and aperiodic tasks online or dynamically (on the fly). They utilized the concept of slack time to guarantee deadlines are met for periodic tasks, however, aperiodic tasks be handled as soft RT in nature.

Tang et al. [2], categorizes one processor as a primary processor and all other processors as secondary processors. Each secondary processor runs a scheduling algorithm and the primary processor runs a load-dispatching algorithm. The advantage of this kind of design is it can work on both homogeneous as well as heterogeneous architectures.

Likewise with the growing popularity of Artificial Intelligence (AI) and Machine Learning (ML) techniques, researchers are also focusing on developing a technique that uses ML to predict the task execution and accordingly schedule the task to a core. Work done by Biswas and Kuia [43] uses a similar approach by using the scheduling principles of the particle swarm algorithm and annealing algorithm to arrive at an optimal scheduling state.

V. LOAD BALANCING BASED SCHEDULING ALGORITHMS

Load Balancing based Scheduling algorithms can be classified into 2 different categories of algorithms namely:

1) Static algorithms: Algorithms, such as Round Robin (RR) and First Come First Served (FCFS), which equitably distribute processing time across the jobs, fall within the static category.
2) Priority Driven algorithms: Priority-driven algorithms are algorithms to determine the order in which tasks or processes are executed based on their priority levels. These algorithms ensure that tasks with higher priority are executed before those with lower priority

TABLE 7. Comparison of scheduling algorithms.

Metrics	Scheduling Algorithms							
	Rate Monotonic (RM)	**Earliest Deadline First (EDF)**	**Least Laxity First (LLF)**	**Deadline Monotonic (DM)**	**H-EDF**	**power and deadline-aware multi-core scheduling (PDAMS)**	**LBPSA**	**P-EDF + TBS**
Priority Assignment	Static	Dynamic	Dynamic	Static	Hybrid	Dynamic	Static	Dynamic
CPU Utilization	Less	Full Utilization	Full Utilization	Less than EDF but more than RM	Full Utilization	Very Less	Less	High
Number of context switching	Very Less	Less	High	Very Less	High	Less	Average	High
Optimality	No	Yes	Yes	No	Yes	Yes	Yes	Yes
Deadline misses	High	Average	Average	High	Less	Less	Less	Less
Response Time	Less	High	Average	Less	Very High	Average	Less	Less
Effectiveness	Simple to implement and has a known utilization bound	Optimal Easy to implement	Takes execution time into consideration	Maximizes the likelihood of meeting task deadlines	Hierarchical structure with adaptability and flexibility in design	Power efficiency	Easy to implement	Less response time on aperiodic task
Limitations	Doesn't provide perfect results in under-loaded situations	Doesn't work efficiently in overload situations	In Laxiety tie more context switch occurs	Cannot handle Aperiodic tasks	Complex design and high overhead in terms of context switching	Additional overhead in terms of power monitoring	Only for Periodic Tasks	Hard to implement
Scheduling Criteria	Period	Deadline	Laxity	Releative Deadline	Hybrid of Period and Deadline	Power consumption and deadlines	Utilization	Peiodic tasks by Partition - EDF and Aperiodic by TBS
Preemptive/Non-Preemptive	Pre-emptive	Both	Both	Preemptive	Both	Both	Preemptive	Preemptive

to meet timing constraints and guarantee system responsiveness. These algorithms are further divided into two categories namely fixed priority and dynamic scheduling.

In this section, only priority driven algorithms are discussed. Also, from table 7 it can be observed that various scheduling algorithms are being compared under different parameters. Priority driven algorithms are a very important class of algorithms in scheduling and therefore it is the focus of this section.

A. PRIORITY DRIVEN ALGORITHMS

As represented in Figure 14, Priority Driven algorithms are further divided into two categories namely Fixed Priority and Dynamic Scheduling. This division is based on the priority assignment, which may be fixed or variable at runtime. Figure 7 shows a comparative study of various Priority Driven Scheduling algorithms.

- Fixed Priority: Fixed Priority algorithms are commonly used in real-time systems to schedule tasks with different priorities. The most common type of algorithms in this class are Rate Monotonic Scheduling (RMS) and Deadline Monotonic Scheduling (DMS). RMS refers to tasks according to their period. The tasks with the shortest period are given top priority in RMS. RMS assumes that tasks are periodic and have fixed execution times. If the system's overall use is below or equal to

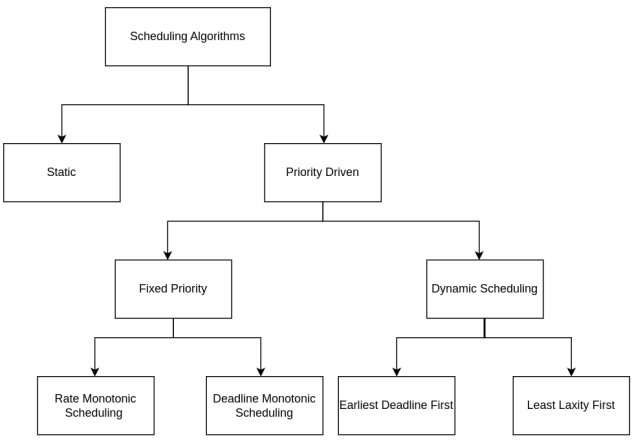


FIGURE 14. Classification of algorithms.

its capacity, it ensures schedulability. This algorithm's drawback is that it cannot deliver an ideal outcome under low load. The Deadline Monotonic Scheduling (DMS) algorithm which also comes under this section, is a real-time scheduling approach where tasks receive priorities based on their deadlines, with shorter deadlines receiving higher priority. This algorithm ensures that tasks with the earliest absolute deadlines are executed first. If a higher-priority task arrives, it preempts the currently executing task.

- Dynamic scheduling: Dynamic scheduling is a technique in which task priorities or scheduling decisions are made at runtime based on the system's current state and conditions. Unlike fixed priority scheduling, where task priorities are predetermined and static, dynamic scheduling allows for more flexible and adaptive task scheduling. The Earliest Deadline First (EDF) algorithm is a dynamic scheduling algorithm where tasks are scheduled based on their relative deadlines. The task with the earliest absolute deadline has the highest priority.

Consider Table 8, which shows sample tasks that are used to illustrate the working of EDF, RM, and DMS algorithms and how they are scheduled in the system. The Earliest Deadline First Scheduling algorithm, Rate Monotonic algorithm, and Deadline Monotonic Scheduling algorithm are used to schedule this job set.

TABLE 8. Real-time task set.

Task	C_i	D_i	P_i
T_1	1	4	4
T_2	2	5	5
T_3	2	6	6

In the real-time task set as given in Table 8, C_i represents Cost or execution time, D_i represents Deadline, and P_i represents the period of the task. After considering three tasks T_1, T_2, and T_3 with all three properties and scheduling this using both RM and EDF algorithms to see the difference among real-time task scheduling approaches, the results showed that all these tasks are periodic in nature and will repeat after P_i time from arrival time. It is assumed that all the tasks arrive at T_0.

Figure 15, shows the working of the EDF algorithm where each task is scheduled dynamically and not at arrival time or priority of the tasks. EDF ensures fewer deadlines missed than fixed priority algorithms like the RM algorithm.

As a dynamic scheduling system, the Earliest Deadline First (EDF) algorithm arranges jobs according to their deadlines rather than their arrival time or priority. Every work is given priority by the algorithm according to its deadline; the task with the earliest deadline is given the most priority. The algorithm chooses the work with the highest priority to be executed and adjusts the priority of tasks based on their deadlines. In order to reduce the likelihood of missing deadlines and support meeting real-time needs, EDF makes sure that jobs with the earliest deadlines are completed first. Given a group of periodic tasks with given deadlines and execution times, the EDF algorithm is the best choice as long as the tasks' aggregate utilization stays below unity. Compared to fixed priority algorithms like the RM algorithm, the technique guarantees fewer missed deadlines.

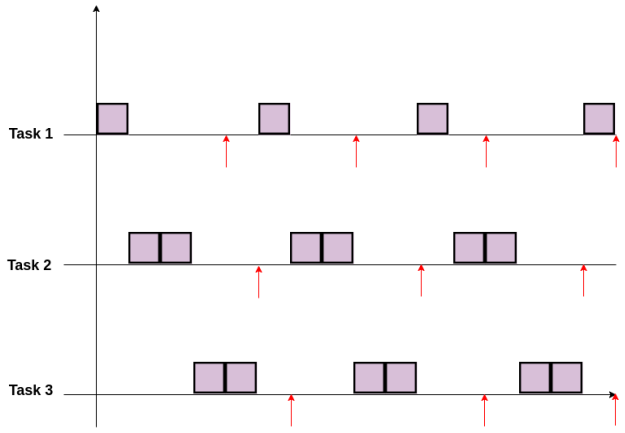


FIGURE 15. Working of EDF algorithm.

The Rate-Monotonic (RM) method, which uses fixed priority scheduling, is shown in Figure 16. This approach assumes completely non-preemptive scheduling, where a task with a higher priority is interrupted from its current execution to allow the higher-priority work to proceed. Tasks are assigned priorities in the following order: $P_1 > P_2 > P_3$. As demonstrated in Figure 15, this proactive behavior may result in more deadline misses than the Earliest Deadline First (EDF) algorithm. In contrast, the EDF algorithm prioritizes activities based on their deadlines and chooses the work with the earliest deadline for execution in order to dynamically schedule tasks based on their deadlines and reduce the likelihood of missing a deadline.

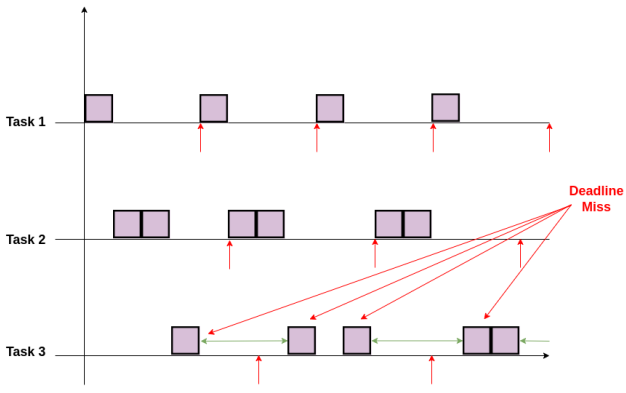


FIGURE 16. Working of RM algorithm.

The Deadline Monotonic Scheduling (DMS) method is a fixed priority-based algorithm where tasks are prioritized

according to their relative deadlines. In this preemptive scheduling approach, tasks with shorter deadlines are assigned higher priorities. If a higher priority task becomes available, the currently running task is preempted, and the higher priority task takes control of the CPU. The priority of a task is inversely related to its deadline, meaning that a task with a shorter deadline receives a higher priority. Figure 17 shows the working of this algorithm.

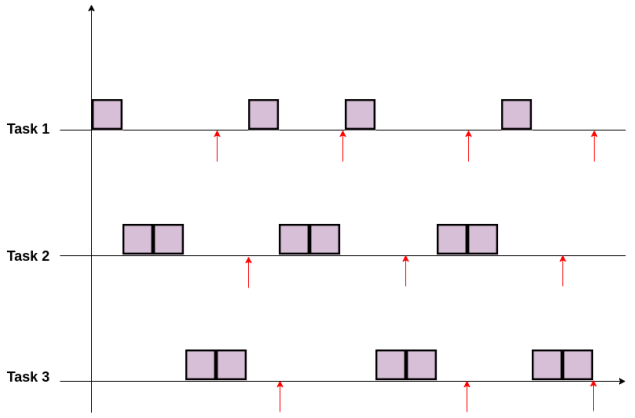


FIGURE 17. Working of DMS algorithm.

Most of the algorithms that are developed are either extensions of these two algorithms or a hybrid of the two. Heuristic and evolutionary algorithms fall under this group. They take more time to execute yet produce results that are close to ideal. The majority of algorithms created for different uses are either hybrids or extensions of the Rate-Monotonic (RM) and Earliest Deadline First (EDF) algorithms. This includes heuristic and evolutionary algorithms. These algorithms are made to find approximations for difficult problems (NP-Hard problems, for example) that are difficult to solve in polynomial time.

Heuristic algorithms are a family of problem-solving strategies that combine practical knowledge and intelligent guesses to arrive at satisfactory answers to a given problem. They frequently use a methodical approach, honing the solution as new data becomes available. A class of heuristic algorithms known as evolutionary algorithms draws inspiration from biological evolution and finds optimal solutions through mechanisms like recombination, mutation, reproduction, and selection.

One such algorithm is LBPSA (Load-Balanced Partitioning and Scheduling Algorithm) mentioned in the work done by Jain and Jain [45], which uses average utilization with upper and lower thresholds for load balancing. All tasks are allocated to processors using EDF-FF or EDF-BF algorithms taking the algorithm with the least tasks missed. After allocating tasks to processors they compute average processor utilization and then shift tasks from processors that have utilization above the upper threshold to processors with utilization below the threshold tasks should be small so that threshold conditions are not reversed.

A comparison of a few heuristic algorithms using several criteria, including priority, CPU usage, number of context switches, optimality, likelihood of missing deadlines, response time, predictability, efficacy, and restrictions, is shown in Table 1.

As shown in Table 1, various algorithms are taken into account and categorized based on different measures listed using the same. Numerous metrics are taken into consideration and several algorithms show differing measures based on their performance. Each algorithm has its highlights and challenges, and the selection is based on the particular needs and features of the multi-core system.

VI. METRICS TO EVALUATE ALGORITHMS

This section explains the standards and procedures that can be used to gauge how well real-time scheduling algorithms perform in schedulability tests. There are several techniques to compare the effectiveness of schedulability tests for real-time scheduling algorithms. Work done by Niz and Rajkumar [14] classifies this into two categories namely theoretical methods and empirical methods. Some important metrics derived from various references are:

Utilization Bound: The utilization bound refers to a theoretical limit on the total utilization or resource usage of tasks within the system. The utilization bound is an important parameter for assessing the schedulability of real-time systems. According to the utilization-based schedulability analysis, if the total utilization of tasks exceeds the utilization bound, the system is considered to be over-utilized and may not be able to meet all task deadlines. On the other hand, if the total utilization is below the utilization bound, the system is theoretically schedulable, and task deadlines can be met given the specified scheduling algorithm. The utilization of a task is calculated as the ratio of its worst-case execution time (WCET) to its period or deadline.

$$Utilization\ Bound = \sum (WCET)_i / T_i \tag{1}$$

The utilization bound is typically expressed as a percentage or a decimal value. For example, if the utilization bound is 0.8 or 80%, it means that the total utilization of tasks in the system should not exceed 80% of the available processing capacity.

Successful Scheduled Task Ratio (SSTR): Work done by Sayed et al. [15] used SSTR as a parameter of evaluation of scheduling algorithm in a Real-Time system. SSTR is a ratio of tasks successfully scheduled to the total number of tasks provided.

$$SSTR = n'/n \tag{2}$$

In the above formula, n' denotes the number of tasks successfully scheduled and n denotes the total number of tasks.

Loads Mean Square Error (LMSE): Work done by Zhang et al. [46] used Loads Mean Square Error (LMSE) as a performance metric to evaluate the accuracy of a system's

load compared to a reference or desired load.

$$LMSE = \sqrt{\frac{1}{n}\sum(L_i - \overline{L})^2} \tag{3}$$

In the above equation $\overline{L} = \frac{L}{m}$ where L is the total load and m is the number of processors. A load of each task set is calculated as $L_i = \sum \frac{C_i}{T_i}$ and Total Load $L = \sum L_i$ over task set.

Energy Consumption: Work done by Chen et al. [17] utilizes energy consumption as a performance indicator to assess the amount of energy needed by each core to process the job set. The energy consumption of a core can be modeled as

$$E_{ti} = \alpha C V_i^2 . f_i . (f_i - s_i) \tag{4}$$

where switching probability is denoted by α, load capacitance by C_i, frequency of core by f_i and voltage by V_i

These are some of the widely used metrics used to evaluate the performance of load balancing-based scheduling algorithms.

VII. CHALLENGES

Optimization of various systems' performance and resource use depends heavily on load balancing methods, ranging from parallel computing and distributed networks to real-time applications. However, developing and implementing effective load balancing algorithms comes with several challenges that researchers and developers must address. These challenges are not specific to any particular paper or algorithm but are common concerns in the domain of load balancing. Some of the key challenges include:

- Heterogeneity of Workloads: Systems often encounter a mix of different types of tasks or workloads, each with unique resource requirements and characteristics. Designing load balancing algorithms that can efficiently handle heterogeneous workloads.
- Scalability: As the size of systems and the number of processing units increase, ensuring the scalability of load balancing algorithms becomes crucial. Scalability challenges arise when the algorithm must efficiently distribute tasks in large-scale systems while minimizing communication overhead and maintaining low latency.
- Dynamic Workloads: Many systems experience dynamic workloads where the task arrival rates and resource demands change over time. Adapting load balancing strategies to handle such dynamically changing conditions is a complex task, as the algorithm needs to be responsive and adaptable in real-time.
- Communication Overhead: In distributed systems or multi-core platforms, communication overhead between processing units can become a bottleneck. Load balancing algorithms must account for this overhead and make intelligent decisions to minimize communication costs while ensuring efficient task allocation.
- Load Prediction and Estimation: Effectively balancing the workload requires accurate prediction and estimation of task execution times and resource requirements. However, predicting future loads with precision is challenging, and inaccuracies can lead to sub-optimal load balancing decisions.
- Synchronization and Coordination: In distributed systems, load balancing algorithms often require synchronization and coordination among processing units. Devising efficient synchronization mechanisms while minimizing contention and overhead is a significant challenge.
- Trade-offs in Objectives: Load balancing often involves multiple objectives, such as maximizing resource utilization, minimizing response time, or meeting real-time deadlines. These objectives may conflict with each other, and achieving a balance among them poses a challenge.
- Algorithm Complexity and Overhead: Some load balancing algorithms can be computationally intensive and introduce additional overhead. Striking a balance between algorithm complexity and the benefits it provides is crucial to ensure practical and efficient implementations.
- Fault Tolerance and Reliability: Load balancing algorithms should be robust and resilient to handle system failures, node crashes, or network partitions. Ensuring fault tolerance and reliability in dynamic load balancing scenarios is a non-trivial challenge.
- Interoperability and Portability: Load balancing solutions must be designed with consideration for interoperability across different platforms and systems. Creating algorithms that are portable and can adapt to various environments is a challenge, particularly in heterogeneous or evolving infrastructures.

Addressing these challenges requires a combination of theoretical analysis, algorithmic innovations, empirical evaluations, and real-world testing. Researchers and developers continually strive to overcome these obstacles to design load balancing algorithms that can optimize system performance, resource utilization, and responsiveness across a wide range of applications and computing environments.

VIII. FUTURE WORKS

Further study of load balancing algorithms for real-time systems could include:

- Development of new algorithms and evaluation of existing ones for real-time scheduling, exploring the impact of load balancing on system performance.
- Investigation of machine learning techniques for real-time scheduling, aiming to enhance scheduling algorithms by adapting to changing workloads dynamically.
- Study of heterogeneous multi-core architectures' impact on real-time scheduling, addressing challenges like load balancing across different cores and accounting for varying processing speeds, task migration overhead, etc.

- Development of new scheduling algorithms tailored for emerging real-time applications (e.g., autonomous vehicles, medical devices), meeting their stringent timing requirements and accommodating various task types (periodic, aperiodic, sporadic).

IX. CONCLUSION

In this paper, various scheduling algorithms for load balancing are categorized and evaluated according to their efficiencies. The paper explores a variety of load balancing algorithms, including Rate Monotonic (RM), Earliest Deadline First (EDF), and Least Laxity First (LLF), among others, through a thorough assessment of the body of existing literature. By taking into account elements such as scalability, overhead, fairness, and reaction time, it offers insights into the strengths and weaknesses of each strategy. Moreover, the study investigates the impact of load balancing on real-time properties of multi-core systems, such as meeting deadlines and ensuring predictable performance. It addresses the trade-offs between load balancing effectiveness and the associated overhead introduced by different techniques.

Based on the findings and analysis, the study draws the conclusion that load balancing in real-time multi-core systems is a challenging operation requiring careful consideration of system features and performance requirements based on the findings and analysis. The selection of a load balancing technique should be based on the individual system context and objectives because no single solution can successfully address all cases. The research emphasizes the necessity of additional investigation and development of load balancing strategies designed for real-time multi-core systems. It highlights how crucial it is to take into account both static and dynamic approaches and to weigh the effects of load balancing on real-time attributes in order to get the best system performance. However, future scope suggested in this paper will help the researchers to extend the information provided in this paper in the areas of heterogenous and multidomain systems. Overall, this thorough investigation aims to advance knowledge of load balancing-based scheduling methods in the context of real-time multi-core systems.

REFERENCES

[1] G. F. El Kabbany, N. M. Wanas, N. H. Hegazi, and S. I. Shaheen, "A dynamic load balancing framework for real-time applications in message passing systems," *Int. J. Parallel Program.*, vol. 39, no. 2, pp. 143–182, Apr. 2011, doi: 10.1007/s10766-010-0134-5.

[2] H.-K. Tang, P. Ramanathan, and K. Compton, "Combining hard periodic and soft aperiodic real-time task scheduling on heterogeneous compute resources," in *Proc. Int. Conf. Parallel Process.*, Sep. 2011, pp. 753–762, doi: 10.1109/ICPP.2011.69.

[3] Q. Tan, K. Xiao, W. He, P. Lei, and L. Chen, "A global dynamic load balancing mechanism with low latency for micokernel operating system," in *Proc. 7th Int. Symp. Syst. Softw. Rel. (ISSSR)*, Sep. 2021, pp. 178–187, doi: 10.1109/isssr53171.2021.00026.

[4] K. Baital and A. Chakrabart, "Dynamic scheduling of real-time tasks in heterogeneous multi-core systems," *IEEE Embedded Syst. Lett.*, vol. 11, no. 1, pp. 29–32, Jun. 2019, doi: 10.1109/LES.2018.2846666.

[5] J. Hu, S. Chung, and J. Kim, "Semi-partitioned task scheduling for predictable execution on NoC-based multi-cores with cache migration," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2450–2463, Nov. 2013.

[6] R. Delgado and B. W. Choi, "New insights into the real-time performance of a multicore processor," *IEEE Access*, vol. 8, pp. 186199–186211, 2020.

[7] S. Xiao, D. Li, and S. Wang, "Periodic task scheduling algorithm for homogeneous multi-core parallel processing system," *Int. J. Parallel Distrib. Syst. Netw.*, vol. 14, no. 3, pp. 116–125, Sep. 2021.

[8] G. Wu, Y. Li, J. Ren, and C. Lin, "Partitioned fixed-priority real-time scheduling based on dependent task-split on multi-core platform," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 12, pp. 3427–3439, Dec. 2013.

[9] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Norwell, MA, USA: Kluwer, 1997, Ch. 10–11.

[10] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. Real-Time Syst. Symp.*, Santa Monica, CA, USA, 1989, pp. 166–171, doi: 10.1109/real.1989.63567.

[11] M. Short, "Improved task management techniques for enforcing EDF scheduling on recurring tasks," in *Proc. 16th IEEE Real-Time Embedded Technol. Appl. Symp.*, Stockholm, Sweden, Apr. 2010, pp. 56–65, doi: 10.1109/RTAS.2010.22.

[12] T. P. Baker, A. Bui, and S. Tixeuil, Eds. *Principles of Distributed Systems*. (Lecture Notes in Computer Science), vol. 5401. Berlin, Germany: Springer, 2008, doi: 10.1007/978-3-540-92221-6_8.

[13] M. Schuh, C. Maiza, J. Goossens, P. Raymond, and B. D. de Dinechin, "A study of predictable execution models implementation for industrial data-flow applications on a multi-core platform with shared banked memory," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Houston, TX, USA, Dec. 2020, pp. 283–295.

[14] D. D. Niz and R. Rajkumar, "Partitioning bin-packing algorithms for distributed real-time systems," *Int. J. Embedded Syst.*, vol. 2, nos. 3–4, p. 196, 2006, doi: 10.1504/ijes.2006.014855.

[15] M. A. El Sayed, E. S. M. Saad, R. F. Aly, and S. M. Habashy, "Energy-efficient task partitioning for real-time scheduling on multi-core platforms," *Computers*, vol. 10, no. 1, p. 10, Jan. 2021, doi: 10.3390/computers10010010.

[16] Y Li, A. Cheng, and A. Mok, "Regularity-based partitioning of uniform resources in real-time systems," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2012, pp. 368–377, doi: 10.1109/RTCSA.2012.40.

[17] B. Chen, H. Chen, D. Ning, M. Zhu, C. Ai, X. Qiu, and W. Dai, "A two-tier partition algorithm for the optimization of the large-scale simulation of information diffusion in social networks," *Symmetry*, vol. 12, no. 5, p. 843, May 2020, doi: 10.3390/sym12050843.

[18] L. Abeni, A. Biondi, and E. Bini, "Partitioning real-time workloads on multi-core virtual machines," *J. Syst. Archit.*, vol. 131, Oct. 2022, Art. no. 102733, doi: 10.1016/j.sysarc.2022.102733.

[19] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time OS," *Real-Time Syst.*, vol. 49, no. 6, pp. 669–714, Nov. 2013, doi: 10.1007/s11241-013-9183-3.

[20] N. Saranya and R. C. Hansdah, "Dynamic partitioning based scheduling of real-time tasks in multicore processors," in *Proc. IEEE 18th Int. Symp. Real-Time Distrib. Comput.*, Auckland, New Zealand, Apr. 2015, pp. 190–197, doi: 10.1109/ISORC.2015.23.

[21] D. Casini, A. Biondi, and G. Buttazzo, "Task splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF," *IEEE Trans. Comput.*, vol. 70, no. 12, pp. 2168–2181, Dec. 2021, doi: 10.1109/TC.2020.3038286.

[22] J. Kang and D. G. Waddington, "Load balancing aware real-time task partitioning in multicore systems," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2012, pp. 404–407, doi: 10.1109/RTCSA.2012.71.

[23] V. Krishnan and J. Torrellas, "A chip-multiprocessor architecture with speculative multithreading," *IEEE Trans. Comput.*, vol. 48, no. 9, pp. 866–880, Sep. 1999, doi: 10.1109/12.795218.

[24] D. Kotz, "Introduction to multiprocessor I/O architecture," in *Input/Output in Parallel and Distributed Computer Systems*, vol. 362, R. Jain, J. Werth, and J. C. Browne, Eds. Boston, MA, USA: Springer, 1996, doi: 10.1007/978-1-4613-1401-1_4.

[25] P. Rajput and V. Kumari, "Modelling and evaluation of multiprocessor architecture," *Int. J. Comput. Appl.*, vol. 51, no. 21, pp. 1–3, Aug. 2012, doi: 10.5120/8323-1371.

[26] A. Ginosar, "MP/C: A multiprocessor/computer architecture," *IEEE Trans. Comput.*, vol. C-31, no. 5, pp. 455–473, May 1982, doi: 10.1109/tc.1982.1676022.

[27] D. J. Lilja, "A multiprocessor architecture combining fine-grained and coarse-grained parallelism strategies," *Parallel Comput.*, vol. 20, no. 5, pp. 729–751, May 1994, doi: 10.1016/0167-8191(94)90003-5.

[28] P. Rajput and V. Kumari, "Modelling and evaluation of multiprocessor architecture," *Int. J. Comput. Appl.*, vol. 51, no. 21, pp. 1–3, Aug. 2012.
[29] G. Liu, Y. Lu, S. Wang, and Z. Gu, "Partitioned multiprocessor scheduling of mixed-criticality parallel jobs," in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2014, pp. 1–10, doi: 10.1109/RTCSA.2014.6910497.
[30] J. Hlusi. *Symmetric Multi-Processing (SMP) Systems on Top of Contemporary Intel Appliances*. Accessed: 2003. [Online]. Available: oai:trepo.tuni.fi:10024/90482
[31] C. Lameter, "NUMA (non-uniform memory access): An overview," *Queue*, vol. 11, no. 7, pp. 40–51, Jul. 2013, doi: 10.1145/2508834.2513149.
[32] L. Fanucci, "Homogeneous and heterogeneous MPSoC architectures with network-on-chip connectivity for low-power and real-time multimedia signal processing," *VLSI Des.*, vol. 2012, Jun. 2012, Art. no. 450302, doi: 10.1155/2012/450302.
[33] X. Wang, D. Qu, Y. Song, and D. Zhang, "Design and implementation of homogeneous multi-core system," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Guiyang, China, Oct. 2017, pp. 788–791, doi: 10.1109/ASICON.2017.8252594.
[34] A. Kayi, T. El-Ghazawi, and G. B. Newby, "Performance issues in emerging homogeneous multi-core architectures," *Simul. Model. Pract. Theory*, vol. 17, no. 9, pp. 1485–1499, Oct. 2009, doi: 10.1016/j.simpat.2009.06.014.
[35] A. Fasiku, O. Oyinloye, F. S. Oluwole, and O. S. Adewale, "Performance evaluation of multi-core processors," Ekiti State Univ., Ado, Ekiti, Ekiti State, Nigeria, 2014, vol. 4, no. 1.
[36] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proc. 22nd Digit. Avionics Syst. Conf.*, San Diego, CA, USA, Dec. 2003, pp. 81–92, doi: 10.1109/MICRO.2003.1253185.
[37] P. Tan, J. Shu, and Z. Wu, "A hybrid real-time scheduling approach on multi-core architectures," *J. Softw.*, vol. 5, no. 9, pp. 958–965, Sep. 2010, doi: 10.4304/jsw.5.9.958-965.
[38] S. Acharya and R. Mahapatra, "A dynamic slack management technique for real-time distributed embedded systems," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 215–230, Feb. 2008, doi: 10.1109/tc.2007.70789.
[39] W. Jeon, W. Kim, H. Lee, and C.-H. Lee, "Online slack-stealing scheduling with modified laEDF in real-time systems," *Electronics*, vol. 8, no. 11, p. 1286, Nov. 2019, doi: 10.3390/electronics8111286.
[40] S. Kumar. (2020). *Least Slack Time (LST) Scheduling Algorithm Real-Time Systems*. [Online]. Available: https://www.geeksforgeeks.org/least-slack-time-lst-scheduling-algorithm-in-real-time-systems/
[41] X. Geng, G. Xu, and Y. Zhang, "Dynamic load balancing scheduling model based on multi-core processor," in *Proc. 5th Int. Conf. Frontier Comput. Sci. Technol.*, Aug. 2010, pp. 398–403, doi: 10.1109/FCST.2010.54.
[42] K.-M. Cho, C.-W. Tsai, Y.-S. Chiu, and C.-S. Yang, "A high performance load balance strategy for real-time multicore systems," *Scientific World J.*, vol. 2014, pp. 1–14, Jan. 2014, doi: 10.1155/2014/101529.
[43] T. Biswas and P. Kuila, "Particle swarm optimization based multi-criteria scheduling for multi-core systems," Nat. Inst. Technol., Sikkim, India, 2020, pp. 115–120, doi: 10.1109/ICE348803.2020.9122860.
[44] R. I. Davis, "On the evaluation of schedulability tests for real-time scheduling algorithms," Univ. York, INRIA, Paris, France, 2016.
[45] D. Jain and S. C. Jain, "Load balancing real-time periodic task scheduling algorithm for multiprocessor environment," in *Proc. Int. Conf. Circuits, Power Comput. Technol. (ICCPCT)*, Nagercoil, India, 2015, pp. 1–5, doi: 10.1109/ICCPCT.2015.7159407.
[46] K. Zhang, B. Qi, Q. Jiang, and L. Tang, "Real-time periodic task scheduling considering load-balance in multiprocessor environment," in *Proc. 3rd IEEE Int. Conf. Netw. Infrastruct. Digit. Content*, Beijing, China, Sep. 2012, pp. 247–250, doi: 10.1109/ICNIDC.2012.6418753.
[47] R. Lavanya, S. Sivarani, C. Prabhaker. M. Lordwin, T. Jeyalakshmi, and M. Muthulakshmi, "Evaluating the performance of various MOEA's to optimize scheduling overhead in homogeneous multicore architecture," in *Proc. Int. Conf. Current Trends towards Converging Technol. (ICCTCT)*, Coimbatore, India, Mar. 2018, pp. 1–9, doi: 10.1109/ICCTCT.2018.8550921.
[48] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas, "Load balancing across near-homogeneous multi-resource servers," in *Proc. 9th Heterogeneous Comput. Workshop (HCW)*, 2000, pp. 60–71, doi: 10.1109/hcw.2000.843733.
[49] S. A. Khawatreh, "An efficient algorithm for load balancing in multiprocessor systems," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 3, pp. 160–164, 2018, doi: 10.14569/ijacsa.2018.090324.
[50] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 88, pp. 50–71, Jun. 2017, doi: 10.1016/j.jnca.2017.04.007.
[51] K. Samunnisa, G. Kumar, and K. R. Madhavi, "A circumscribed research of load balancing techniques in cloud computing," *Int. J. Innov. Technol. Exploring Eng.*,vol. 8, no. 6S4, pp. 336–339, Apr. 2019.
[52] M. Kumar and B. Bhushan, "A methodological comparison of the most efficient load balancing algorithms in cloud computing," in in *Proc. Int. Conf. Innov. Comput. Commun. (ICICC)*, 2020, pp. 1–6. [Online]. Available: https://ssrn.com/abstract=3598908 or and http://dx.doi.org/10.2139/ssrn.3598908
[53] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Gener. Comput. Syst.*, vol. 37, pp. 141–147, Jul. 2014, doi: 10.1016/j.future.2013.06.009.
[54] C. Imes, H. Zhang, K. Zhao, and H. Hoffmann, "CoPPer: Soft real-time application performance using hardware power capping," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Umea, Sweden, Jun. 2019, pp. 31–41, doi: 10.1109/ICAC.2019.00015.
[55] S. Shukla and R. Singh, "Survey on load balancing techniques," Pranveer Singh Inst. Technol., Kanpur, India, 2019.
[56] S. Aslam and M. A. Shah, "Load balancing algorithms in cloud computing: A survey of modern techniques," in *Proc. Nat. Softw. Eng. Conf. (NSEC)*, Rawalpindi, Pakistan, Dec. 2015, pp. 30–35, doi: 10.1109/NSEC.2015.7396341.
[57] R. Ramya, S. Puspalatha, T. Hemalatha, and M. Bhuvana, "A survey on and performance analysis of load balancing algorithms using meta heuristics approach in public cloud-service Provider's perspective," in *Proc. Int. Conf. Intell. Comput. Commun. for Smart World (I2C2SW)*, Erode, India, Dec. 2018, pp. 380–385, doi: 10.1109/i2c2sw45816.2018.8997138.
[58] B. Kocot, P. Czarnul, and J. Proficz, "Energy-aware scheduling for high-performance computing systems: A survey," *Energies*, vol. 16, no. 2, p. 890, Jan. 2023, doi: 10.3390/en16020890.
[59] I. Kar, R. N. R. Parida, and H. Das, "Energy aware scheduling using genetic algorithm in cloud data centers," in *Proc. Int. Conf. Electr., Electron., Optim. Techn. (ICEEOT)*, Chennai, India, Mar. 2016, pp. 3545–3550, doi: 10.1109/ICEEOT.2016.7755364.
[60] M. Safari and R. Khorsand, "Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment," *Simul. Model. Pract. Theory*, vol. 87, pp. 311–326, Sep. 2018, doi: 10.1016/j.simpat.2018.07.006.
[61] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Proc. 3rd Int. Symp. Parallel Archit., Algorithms Program.*, Dec. 2010, pp. 89–96.
[62] M. Sirshar, H. Rizwan, M. Shabbir, and H. Nawaz, "Scheduling in real time system," Fatima Jinnah Women Univ., Rawalpindi Pakistan, 2019, doi: 10.20944/preprints201912.0072.v1.
[63] B. N. K. Reddy, Y. C. Krishna, P. N. S. Nitish, and S. D. Bharatula, "Optimizing task scheduling in multi-thread real-time systems using augmented particle swarm optimization," in *Proc. 37th Int. Conf. VLSI Design 23rd Int. Conf. Embedded Syst. (VLSID)*, Kolkata, India, Jan. 2024, pp. 666–671, doi: 10.1109/vlsid60093.2024.00118.
[64] V. Madala and G. K. Chellamani, "Energy efficiency of embedded controllers through DVFS," in *Proc. 4th Int. Conf. Adv. Electr., Comput., Commun. Sustain. Technol. (ICAECT)*, Bhilai, India, Jan. 2024, pp. 1–5, doi: 10.1109/icaect60202.2024.10468931.



SHRUTI JADON received the B.Tech. degree from Uttar Pradesh Technical University, the M.Tech. degree from Banasthali Vidyapith, and the Ph.D. degree from NIT Allahabad, in 2019

Currently, she is an Associate Professor with PES University, Bengaluru, India. She has published many papers in various journals and conferences. Her areas of research interests include real-time systems, cryptography, and blockchain.



PRADYUMAN KAVEDI KANNAN is currently pursuing the B.Tech. degree in computer science and engineering with PES University, Bengaluru, India. He is a Software Intern with Arista Networks. His research interests include data structures and algorithms and computer networking.



URMIL KALARIA is currently pursuing the B.Tech. degree in computer science and engineering (CSE) with PES University, Bengaluru, India. He is a full-time Software Development Engineer (SDE) Intern with Samsung Electronics, Bengaluru. His areas of research interests include designing scalable systems, real-time systems, full-stack development, distributed systems, and big data and database technologies.



K. R. VARSHA is currently pursuing the B.Tech. degree in computer science and engineering (CSE) with PES University, Bengaluru, India. She is a Cybersecurity Engineer Intern with the Aptiv Technical Centre, Bengaluru. Her areas of research interests include cybersecurity, cryptography, and supply chain management.



KARTHIK GUPTA is currently pursuing the B.Tech. degree in computer science and engineering with PES University, RR Campus, Bengaluru, India. He is a SDE Intern with Zebra Technologies. His research interests include RTOS and Kernel development.



PRASAD B. HONNAVALLI (Member, IEEE) received the B.E. degree from UVCE, Bangalore University, and the M.B.A. degree from The University of Melbourne, Australia.

Currently, he is a Professor with PES University, Bengaluru, India. He is also the Director of the PESU Center for Information Security, Forensics, and Cyber Resilience (C-ISFCR) and the PESU Center for Internet of Things (C-IoT), with a focus on security. He leads teachings, research executive education, and industry partnerships and consultancy in his areas of focus. He has authored and coauthored several research papers in leading journals and conferences. His research is focused on various aspects of information security, such as network and cloud security, software and web security, mobile application network security, cryptography and blockchain, the IoT, SCADA, and industry 4.0.