Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.DOI

A Comprehensive Study Of Load Balancing Approaches in Real-time Multi-Core Systems

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 a number of difficulties and constraints connected with implementing tasks in a real-time context. This research study primarily focuses on load balancing, one of the major challenges of 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. Several algorithms have been implemented in theory to improve performance and efficiency by distributing the system workload across multiple nodes. The goal of this paper is to present a critical analysis of existing Load-Balancing techniques and discuss various parameters such as throughput, performance, migration time, response time, overhead, resource usage, scalability, fault tolerance, power savings, and other factors that determine the effectiveness of Load-Balancing in real-time systems.

INDEX TERMS Scheduling Algorithms; Real-time Systems; Priority-driven Algorithms; EDF (Earliest Deadline First); RM (Rate-Monotonic); LBPSA (Load Balanced Partitioning and Scheduling Algorithm); Homogenous Multi-core; Heterogenous 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 and space, etc. In the real-time system, the correctness of the system's behavior not only depends on the computer's logic but also on the physical instant at which the results are produced. This implies that a real-time computer system must respond to stimuli from the controlled item (or the operator) within time periods (deadlines) 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 real-time applications/servers that ensure maximum resource utilization while completing tasks on/before deadlines.

It is crucial to research and comprehend the available ways since load balancing is crucial for resource utilization, task scheduling, and deadline fulfillment in real-time systems. This research paper discusses the various Load-Balancing algorithms and system parameters on which this algorithm depends.

A. SCOPE

Most of these surveys have given insight into the performance and throughput issues and their solutions in different areas of load-balancing into the most recent and cutting-edge load-balancing methods applied to multi-core real-time systems. This survey has highlighted relevant research articles, conferences, and business developments which 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 fairness, that is used to gauge the efficiency of load balancing.

The paper also talks about the difficulties and unresolved problems with load balancing for real-time multi-core sys-

TABLE 1: A relative comparison of the pre-existing surveys on load-balancing algorithms with the proposed survey

Paper	Year	Objective	Algorithm used	Pros	Cons
Paper [9]	2011	Perform distribution and make adjustments during runtime	EDF	Uses both centralized and decentralized approach	Message passing should be explored more
Paper [3]	2011	Increase Responsiveness	H-EDF and FF	Uses slack time to schedule Aperiodic tasks	Only Aperiodic tasks
Paper [12]	2021	Distribute tasks effectively	DM	Uses a feedback control algorithm	Cluster Calculation
Paper [13]	2019	Heuristic algorithm based on characteristics	LLF	Mixed-integer linear programming(MILP) problem	Only Heterogeneous tasks
Paper[2]	2012	Maximizes core utilization bound	EDF	More efficient than partitioned scheduling	More complex to implement than partitioned scheduling
Paper [18]	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
Paper [19]	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
Paper [20]	2013	Propose a new partitioned fixed-priority real-time scheduling algorithm for multicore platforms	BDTD/TS (B-tree Dependent Task Dispatching/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 communication overhead is too high

tems, describing new trends in the field and future research directions.

B. MOTIVATION

- A thorough analysis and evaluation of the various approaches are lacking in the existing research on load-balancing strategies in real-time multi-core systems. The goal of the paper is to give a general overview of the classification, scheduling mechanisms, and load-balancing mechanisms used in RT systems.
- The literature that is now available primarily examined several load-balancing-based scheduling algorithms, their architecture, and HLD while providing a quality assessment for their future development.
- In order 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 and effectiveness. The paper also groups LB-based algorithms according to various sub-classifications and provides 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 numerous scheduling techniques and problems in Load Balancing Approaches in Real-time Multi-Core Systems through various

studies. We highlight several security-related open problems and difficulties with respect to the implementation and evaluation of algorithms. The contributions of this paper are listed below.

- The study discusses the development, background study, and significance of various Load Balancing-based scheduling algorithms and their associated scheduling methods along with partitioning. Both its advantages and disadvantages have been discussed.
- In this study, load-balancing-based scheduling algorithms are examined in relation to 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 homogenous and heterogenous, static and dynamic, ML- and non-ML-based, and energy- and non-energy-efficient (Section IV).
- Finally, it also highlights future areas for study into partitioning and implementation methods using numerous algorithms for load-balancing strategies in RT systems.

D. ORGANIZATION

The remaining part of this work is structured as follows. Section II contains a background study on RT-Systems Classification, Scheduling Mechanisms, and Load Balancing Mechanisms. Section II also provides the background of Multi-Processor and Multi-Core Architecture. Section III contains the architecture and High-Level designs used in the referred literature metric to collect the result and its calculation strategy used. Section IV classifies the Load Balancing-based scheduling algorithms on various properties and discusses the Taxonomy of LB-based Algorithms. The various Algorithms considered are i) Homogenous and Heterogenous, ii) Static and Dynamic, iii) ML-based algorithms (Machine Learning based) iv) Energy efficient Algorithms. Section V discusses the limitations and flaws in the pre-described approach taken by the previous work. Section VI talks about the Future Scope. Section VII discusses the Conclusion.

II. BACKGROUND STUDY AND RELATED WORK

A. BACKGROUND STUDY ON RT SYSTEMS

A real-time system is a computer system that is programmed to respond to external events or stimuli within a specific time frame. It is characterized by its ability to process and react to events in real time.

B. CLASSIFICATION

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 two categories:

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 include 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.

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 3 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, also known as sporadic 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) and Earliest Deadline First (EDF) are two common scheduling techniques.
- 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.
- 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.

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. Load Partitioning is one such essential mechanism.

1) Partitioning

Partitioning 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). 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.
- 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.
- 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.

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 dwell deep into various scheduling techniques and algorithms in later sections of this paper.

E. MULTI-PROCESSOR AND MULTI-CORE ARCHITECTURE

A multi-processor architecture 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. There are two main types of multi-processor architectures: symmetric multiprocessing (SMP) and non-uniform memory access (NUMA).

- 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 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. Multi-core architectures are becoming increasingly common, and they are now found in most personal computers and servers. They offer a number of 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 mul-

tiple independent processors, while a multi-core system has multiple cores on a single processor. This difference has a number of 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.

F. RELATED WORK

There has been a lot of study done on the creation of scheduling algorithms for real-time systems. The researchers aim to find the optimal algorithm for such systems. This section provides the nitty-gritty details of some of these works.

In paper [1] authors discussed the Hybrid scheduling algorithms for the soft real-time tasks on a Homogeneous multi-core system. Here a Real-time application is partitioned into independent parallelizable tasks. A Directed Acyclic Graph (DAG) represents a precedence relationship among those tasks. They used MDFA (Most Demand First Algorithm) and IFA (Idlest First Algorithm) as Load-Balancing Allocation Algorithms. They proposed a two-level scheduling scheme where the first level consists of sporadic servers. Each core consists of multiple sporadic servers, each with an assigned CPU budget c, period p, a ready queue of tasks, and a scheduler that uses EDF (Earliest Deadline First) or RM (Rate Monotonic) or Time-sharing algorithms based on the task. At the bottom level, there is an OS scheduler preferably a RM scheduler that maintains all the top-level sporadic servers. The Job of the OS scheduler is to refresh each sporadic server after its period p, reassign the budget, and schedule the task with the RM algorithm. Tasks of the ready servers are executed until the CPU budget of the sporadic server is exhausted or high-priority tasks arrive.

On a Network-on-Chip (NoC) based multi-core architecture, Paper [2] provides a semi-partitioned strategy for reliably scheduling periodic heavy real-time workloads utilizing cache migration. To reduce the cost associated with online cache migration, as many processes as feasible are statically partitioned onto cores. To enhance task set schedulability, the remaining jobs are permitted to move in a preset way among a pre-selected group of cores. The Earliest Deadline First (EDF) strategy is used to schedule work on each core since it maximizes the bound on core utilization. Tasks on a particular core are permitted to statically choose and lock a portion of their memory lines in the core's private cache in order to increase the predictability of multi-task execution on a single core. For a job that is migrating, locked lines that are part of the task are moved and then relocked on the target core. Even though a task may be moved in the middle of a particular job's execution since we employ locked cache migration, migration overhead is predictable. Using a time-division multiplexed (TDM) arbitration method, reliable sharing of the on-chip communication infrastructure is ensured.

On heterogeneous real-time systems, paper [3] focuses 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. [4], [5], and [6] focus on slack management and distribution techniques. The proposed Slack Distribution Techniques have been considered with EDF and Rate-Based scheduling schemes.

Paper [7] provided 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.

In paper [8] a unique scheduling approach for real-time multicore computers was suggested to balance computation demands and save electricity. For their power and deadline-aware multicore 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.

The paper [9] describes a dynamic method for job scheduling and load balancing in message-passing systems that is application-independent. It is a DAG-based Dynamic Load Balancing technique for Real-time applications (DAG-DLBR) that is meant to function dynamically to deal with any changes in load during runtime.

Paper [12], The algorithm 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.

Paper [13], The paper 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.

Paper[18], The authors shed fresh light on the performance of Linux's real-time extensions, Xenomai and RT-Preempt, on a homogenous multicore CPU. They identify distinct multicore installations and analyse their trade-offs, as determined via experimental scheduling delay evaluation. Then, to find the optimal multicore deployment, they suggest a statistical approach based on a version of the chi-square test. The periodicity of the best multicore deployment is used to validate its practicality.

Paper[19], 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.

The suggested approach is an extremely useful tool for scheduling periodic activities on homogeneous multi-core parallel processing systems. The algorithm can schedule all jobs and fulfill their deadlines, and it surpasses the EDF method in terms of task scheduling capacity and makespan.

Paper[20] This paper 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 can be classified in different ways such

- Homogenous and Heterogenous Algorithms
- Static and Dynamic Algorithms
- ML-based Algorithms
- Energy Efficient Algorithms

A. HOMOGENOUS AND HETEROGENOUS

Homogeneous algorithms refer to computational tasks that can be evenly distributed among the available cores without causing significant imbalance. These algorithms exhibit uniform time complexity across all inputs and are designed to take advantage of the parallel processing capabilities of multi-core architectures efficiently.

Homogeneous algorithms refer to computational tasks that can be evenly distributed among the available cores without causing significant imbalance. These algorithms exhibit uniform time complexity across all inputs and are designed to take advantage of the parallel processing capabilities of multi-core architectures efficiently. Table 2, provides a complete classification of papers based on Homogeneous and Heterogeneous multi-core systems.

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.

Dynamic load balancing is a technique where the assignment of tasks to cores is made during runtime. As the program 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. Table 3, provides a complete classification of papers based on Static and Dynamic Load Balancing Algorithms.

C. ML AND NON-ML BASED ALGORITHMS

In contrast, ML algorithms are a subset of artificial intelligence that allows computers to learn from data without being explicitly programmed. 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.

Paper [10] uses particle swarm optimization which uses ML.

TABLE 2: Classification of papers based on Homogenous and Heterogenous

Paper	Year	Homogenous / Heterogenous	Approach	Drawbacks
[1]	2010	Homogenous	Hybrid scheduling method using two-level scheduling scheme.	Future allow on heterogenous cores
[2]	2012	Homogenous	Under Locked Cache Migration, Semi-Partitioned Hard-Real-Time Scheduling	Only on homogenous machines
[3]	2011	Heterogenous	Soft aperiodic tasks and hard periodic tasks	Aperiodic tasks have soft deadlines
[9]	2021	Heterogenous	Uses particle swarm optimization	Complexity, Extra overhead
[13]	2019	Dynamic	Examines the existing scheduling techniques in a heterogeneous multicore system and devise a method to adapt the homogeneous system model to a heterogeneous scheduling architecture.	
[15]	2015	Homogenous	Discusses the criteria and techniques for evaluating schedulability tests for real-time scheduling algorithms and just periodic jobs.	Only Periodic Tasks
[16]	2012	Homogenous	Scheduling algorithm that takes into account the load balance of real-time periodic tasks	Only periodic
[17]	2018	Homogenous	Optimizing scheduling overhead in such processors	
[18]	2020	Homogenous	Insights into the real-time performance of real-time extensions of Linux, namely Xenomai and RT-Preempt	Only homogenous Processors
[20]	2013	Homogenous	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.

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, Stealing algorithm, Threshold, 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. 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.

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 maximiz-

TABLE 3: Classification of papers based on Static and Dynamic

Paper	Year	Static / Dynamic	Approach	Algorithm/Model
[1]	2010	Static	Hybrid scheduling method using two-level scheduling scheme.	Rate Monotonic/Sporadic server
[2]	2012	Static	Semi-Partitioned Hard-Real-Time Scheduling Under Locked Cache Migration	Cache locking and locked cache migration
[7]	2010	Dynamic	The dynamic load balancing scheduling problem is represented by this model as a quintuple <E, T, L, S, C>.	Model
[9]	2010	Dynamic	Dynamic Load Balancing Algorithm Based on DAG for Real-Time Applications	(DAG-DLBR)
[10]	2021	Dynamic	Uses particle swarm optimization	Particle swarm
[12]	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.	Strategy
[13]	2019	Dynamic	Examines the existing scheduling techniques in a heterogeneous multicore system and devise a method to adapt the homogeneous system model to a heterogeneous scheduling architecture.	Model
[14]	2016	Static	Discusses the criteria and techniques for evaluating the performance of real-time scheduling algorithms' schedulability tests.	Model
[20]	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)

ing throughput and minimizing task completion times without necessarily optimizing power usage. Table 4, provides a complete classification of papers based on Energy efficiency in multi-core systems.

IV. ARCHITECTURE AND HIGH-LEVEL DESIGN

The ability to pack several million transistors into a chip thanks to current chip fabrication technology enables progressively more complicated designs. The diminishing instruction-level parallelism gains, nearly unchanged memory latency, increased heat generation, inherent complexity of designing a single core with a large number of transistors, and the financial costs associated with this design all serve to discourage the development of more complex uniprocessors. Due of these factors, the practice of creating chips with several processor cores (multi-core) is currently popular. A multi-core processor combines two or more independent cores into a single package composed of a single integrated circuit (IC). The processors and utilize the same interconnect

TABLE 4: Classification of papers based on Energy and Non Energy efficient

Paper	Year	Static / Dynamic	Approach	Efficiency
[8]	2014	Energy efficient	power and deadline-aware multicore scheduling	54.2%
[17]	2018	Energy efficient	Optimizing scheduling overhead in such processors	PAES 0 % NPGA 13.3 % NSGA 23.4 % MOGA 23.4 %

with rest of the system. Each "core" implements it's own optimizations such superscalar execution, pipe-lining, and multithreading. There are 2 types of multi-core architecture that broadly 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.

Paper [1], 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.

Paper [2], 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.

Paper [3], categorizes one processor as a Master Processor and all other processors as slave processors. Each slave processors run a scheduling algorithm and the master 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. Paper [10] 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 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 to meet timing constraints and guarantee system responsiveness. These algorithms are further divided into two categories namely Fixed Priority and Dynamic Scheduling.

Priority Driven algorithms are a very important class of algorithms in scheduling and therefore it is the focus of this section.

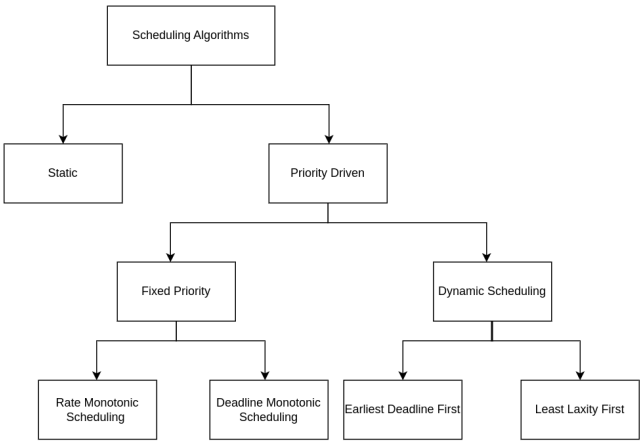

FIGURE 1: Classification of algorithms

TABLE 5: 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 multicore 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		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
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

A. PRIORITY DRIVEN ALGORITHMS

As represented in Fig. 1, Priority Driven algorithms are further divided into 2 categories namely Fixed Priority and Dynamic Scheduling. This division is based on the priority assignment, which may be fixed or variable at runtime.

- Fixed Priority: Fixed Priority algorithms are commonly used in real-time systems to schedule tasks with different priorities. The most common type of algorithm in this class is the Rate Monotonic algorithm (RMS). RMS refers tasks according to their period. The tasks with the shortest period is given top priority. RMS assumes that tasks are periodic and have fixed execution times. If the system's overall use is below or equal to its capacity, it ensures scheduleability. This algorithm's drawback is that it cannot deliver an ideal outcome under low load.
- 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 2, which shows sample tasks that are used to illustrate the working of EDF and RM algorithms and how they are scheduled in the system. The Earliest Deadline First Scheduling algorithm and Rate Monotonic algorithm are used to schedule this job set.

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

TABLE 6: Real-Time Task Set

In the Real-Time Task set Table 2, C_i represents Cost, D_i represent Deadline, P_i represents Period of task. We have considered 3 tasks T_1, T_2, and T_3 with all three properties. We will schedule this using both RM and EDF algorithms to see the difference among real-time task scheduling approaches. 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. We have considered a time frame of 16 time units.

Figure 2, 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 miss than fixed priority algorithms like the RM algorithm.

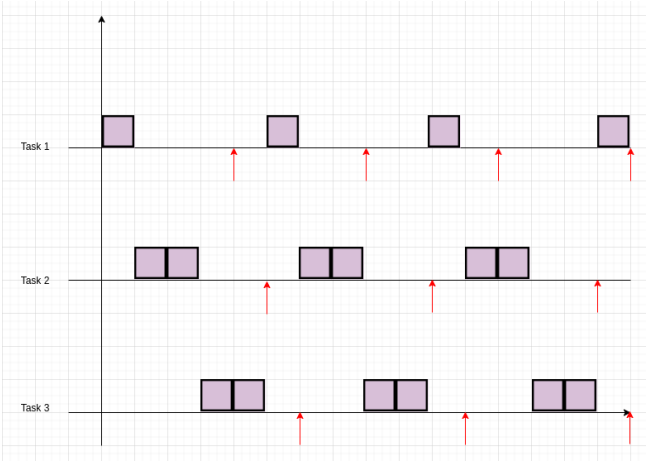

FIGURE 2: Working of EDF algorithm

Figure 3 shows the working of the RM algorithm where each is scheduled using the fixed priority. The priority of tasks are in order of $P_1 > P_2 > P_3$. The algorithm assumes fully preemptive scheduling of tasks. When a task with a higher priority arrives the executing task halts and gives way to a higher-priority task. Due to this, there are more deadline misses than the EDF algorithm, which is shown in Fig. 3.

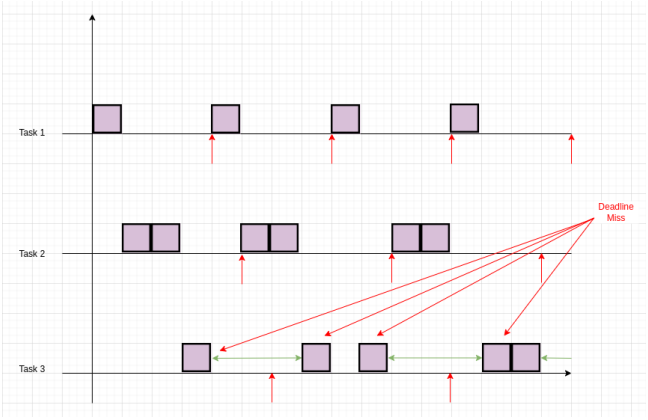

FIGURE 3: Working of RM algorithm

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.

One such algorithm is LBPSA (Load Balanced Partitioning and Scheduling Algorithm) mentioned in the paper [15], 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 which have utilization above upper threshold to processors with utilization below 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 of using the same. Numerous metrics are taken into consideration where 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. Paper [14] classifies this into 2 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.

$$Utilization\ Bound = \sum (WCET)_i/T_i \qquad (1)$$

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): Paper [15] has 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.

$$SSTR = n'/n \qquad (2)$$

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): Paper [16] uses 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: Paper [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 a Load balancing-based scheduling algorithm.s

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 come 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 workload.
- 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 multicore 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 suboptimal 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. This could include the development of new algorithms, the evaluation of existing algorithms, and the study of the impact of load balancing on the performance of real-time systems. Investigation of the use of machine learning techniques for real-time scheduling. Machine learning could be used to improve the performance of real-time scheduling algorithms by dynamically adapting to the changing workload of a sys-

tem. Study of the impact of heterogeneous multicore architectures on real-time scheduling. Heterogeneous multicore architectures pose new challenges for real-time scheduling, such as the need to balance load across different cores and the need to account for the different processing speeds of different cores in addition to the overhead of task migration. Development of new scheduling algorithms for emerging real-time applications. Emerging real-time applications, such as those for autonomous vehicles and medical devices, pose new challenges for real-time scheduling. New algorithms are needed to meet the strict timing requirements of these applications and to accommodate all types of tasks be it periodic or aperiodic or sporadic.

IX. CONCLUSION

At the end of the survey 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. Overall, this thorough investigation advances knowledge of load-balancing-based scheduling methods in the context of real-time multi-core systems.

ACKNOWLEDGMENT

We would like to express our gratitude to Dr. Shruti Jadon, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this Research. We take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from her. We are grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro-Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, Dr. B.K. Keshavan, Dean of Faculty, PES University for providing us with various opportunities and enlightenment during every step of the way. Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

REFERENCES

[1] Tan, Pengliu & Shu, Jian & Wu, Zhenhua, Hybrid Real-Time Scheduling Approach on Multi-Core Architectures JSW. 5. 958-965. 10.4304/jsw.5.9.958-965.

[2] Jinyu Hu, Sanghyun Chung, and Jongkeun Kim, "Semi-Partitioned Task Scheduling for Predictable Execution on NoC-Based Multicores with Cache Migration," IEEE Transactions on Computers, vol. 62, no. 11, pp. 2450-2463, Nov. 2013.

[3] Tang, Hsiang-Kuo & Ramanathan, Parmesh & Compton, Katherine. (2011). Combining Hard Periodic and Soft Aperiodic Real-Time Task Scheduling on Heterogeneous Compute Resources. Proceedings of the International Conference on Parallel Processing. 753 - 762. 10.1109/ICPP.2011.69.

[4] Acharya, Subrata & Mahapatra, Rabi. (2008). A Dynamic Slack Management Technique for Real-Time Distributed Embedded Systems. Computers, IEEE Transactions on. 57. 215-230. 10.1109/TC.2007.70789.

[5] Jeon, Wonbo & Kim, Wonsop & Lee, Heoncheol & Lee, Cheol-Hoon. (2019). Online Slack-Stealing Scheduling with Modified laEDF in Real-Time Systems. Electronics. 8. 1286. 10.3390/electronics8111286.

[6] ShivamKumar1 Least Slack Time (LST) scheduling Algorithm in real-time systems https://www.geeksforgeeks.org/least-slack-time-lst-scheduling-algorithm-in-real-time-systems/ 2020.

[7] Geng, Xiaozhong & Xu, Gaochao & Zhang, Yuan. (2010). Dynamic Load Balancing Scheduling Model Based on Multi-core Processor. Proceedings - 5th International Conference on Frontier of Computer Science and Technology, FCST 2010. 398-403. 10.1109/FCST.2010.54.

[8] Cho, Keng-Mao & Chiu, Yi-Shiuan & Yang, Chu-Sing. (2014). A High Performance Load Balance Strategy for Real-Time Multicore Systems. TheScientificWorldJournal. 2014. 101529. 10.1155/2014/101529.

[9] El-Kabbany, Ghada & Wanas, Nayer & Hegazi, Nadia & Shaheen, Samir. (2011). A Dynamic Load Balancing Framework for Real-time Applications in Message Passing Systems. International Journal of Parallel Programming. 39. 143-182. 10.1007/s10766-010-0134-5.

[10] Biswas, Tarun & Kuila, Pratyay. (2020). Particle Swarm Optimization based Multi-criteria Scheduling for Multi-Core Systems. 115-120. 10.1109/ICE348803.2020.9122860.

[11] Khera, Ishan & Kakkar, Ajay. (2012). Comparative Study of Scheduling Algorithms for Real Time Environment. International Journal of Computer Applications. 44. 5-8. 10.5120/6233-7797.

[12] Q. Tan & K. Xiao & W. He & P. Lei & L. Chen. (2021). A Global Dynamic Load Balancing Mechanism with Low Latency for Microkernel Operating System. pp. 178-187. 10.1109/ISSSR53171.2021.00026.

[13] K. Baital & A. Chakrabart. (2019). Dynamic Scheduling of Real-Time Tasks in Heterogeneous Multicore Systems. vol. 11, no. 1, pp. 29-32. 10.1109/LES.2018.2846666.

[14] Davis, Robert I.. On the Evaluation of Schedulability Tests for Real-Time Scheduling Algorithms. (2016).

[15] Jain, Divya & Jain, Sushil. (2015). Load balancing real-time periodic task scheduling algorithm for multiprocessor enviornment. 1-5. 10.1109/ICCPCT.2015.7159407.

[16] K. Zhang, B. Qi, Q. Jiang and L. Tang, "Real-time periodic task scheduling considering load-balance in multiprocessor environment," 2012 3rd IEEE International Conference on Network Infrastructure and Digital Content, Beijing, China, 2012 pp. 247-250, doi: 10.1109/ICNIDC.2012.6418753.

[17] R. Lavanya, S. Sivarani, C. P. M. Lordwin, T. Jeyalakshmi and M. Muthulakshmi, "Evaluating the Performance of Various MOEA's to Optimize Scheduling Overhead in Homogeneous Multicore Architecture," 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), Coimbatore, India, 2018 pp. 1-9, doi: 10.1109/ICCTCT.2018.8550921.

[18] R. Delgado and Byoung-Wook Choi, "New Insights Into the Real-Time Performance of a Multicore Processor," IEEE Access, vol. 8, pp. 186199-186211, Oct. 2020.

[19] Siyu Xiao, Dongguang Li, and Shiyao Wang, Periodic Task Scheduling Algorithm for Homogeneous Multi-core Parallel Processing System, International Journal of Parallel and Distributed Systems and Networks, vol. 14, no. 3, pp. 116-125, Sep. 2021
[20] Guowei Wu, Ying Li, Jiankang Ren, and Chi Lin, "Partitioned Fixed-priority Real-time Scheduling Based on Dependent Task-Split on Multi-core Platform," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 12, pp. 3427-3439, Dec. 2013.