Chapter 2

Related Work

In High-performance computing, utilizing multi-core systems within cluster environments presents challenges that demand careful consideration. As we witness the exponential growth of data-intensive applications and the advent of sophisticated computational tasks, ensuring optimal performance in multi-core clusters becomes a critical concern. The multi-core clustering scenario has been changing in recent years, focusing on processing power by applying Artificial Intelligence (AI) techniques, Machine Learning (ML) techniques, and Deep Learning (DL) techniques. A broader classification of multi-core systems for HPC environment has shown in Figure 2.1. This chapter presents a detailed literature to support and improve our work.

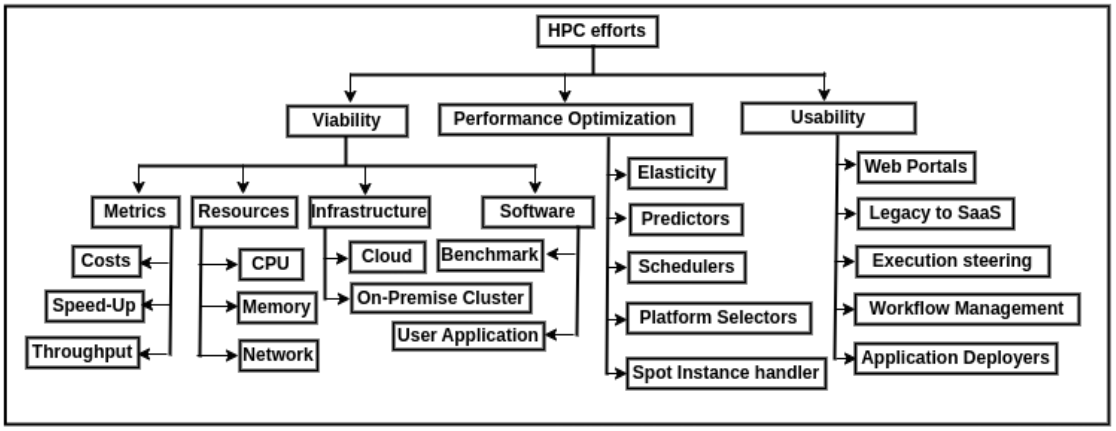

Figure 2.1: Taxonomy of HPC based on multi-core clusters

From Figure 2.1 it is clear that the High-performance computing (HPC) research and efforts can be categorized into three primary domains: viability, performance optimization, and usability. Each of these domains is further divided into more specific subcategories, reflecting the multifaceted nature of HPC in multi-core clusters.

The first domain, viability, encompasses four key subcategories: metrics, resources, infrastructure, and software. Metrics are crucial for evaluating the efficiency and effectiveness of HPC systems and are further divided into costs, speedup, and throughput. These metrics help in assessing the financial and computational efficiency of the system. Resources include the fundamental components of HPC systems, namely CPU, memory, and network capabilities, each playing a vital role in the overall performance. Infrastructure is categorized into cloud and on-premise clusters, offering different deployment and scalability options for HPC environments. Lastly, software is split into benchmark-based and user application-based categories, providing a framework for performance evaluation and application-specific optimizations.

The second domain, performance optimization, is essential for enhancing the efficiency and effectiveness of HPC systems. This domain includes five subcategories: elasticity, predictors, schedulers, platform selectors, and spot instance handlers. Elasticity refers to the system's ability to adapt to varying workloads by scaling resources up or down. Predictors are tools or algorithms used to forecast system behavior and resource needs. Schedulers manage the allocation of tasks to various resources to optimize performance. Platform selectors determine the most suitable computing platform for specific tasks, while spot instance handlers deal with the use of spare computational resources available at lower costs, particularly in cloud environments.

The third domain, usability, focuses on making HPC systems more accessible and user-friendly. It is divided into five subcategories: web portals, legacy to systems to Software as a Service (SaaS) transitions, execution steering, workflow management, and application deployers. Web portals provide user-friendly interfaces for accessing HPC resources. Transitioning legacy systems to Software as a Service (SaaS) models facilitates easier maintenance and scalability. Execution steering allows users to dynamically manage and adjust running applications. Workflow management involves the orchestration of various computational tasks, ensuring smooth and efficient operations. Lastly, application deployers are tools that simplify the deployment of applications across HPC systems.

This classification framework provides a comprehensive overview of the various facets of HPC research and development, highlighting the diverse approaches and methodologies employed to enhance the viability, performance, and usability of multi-core clusters.

2.1 Introduction

High-performance computing (HPC) has evolved beyond merely addressing the specialized needs of researchers in science and engineering. It now plays a critical role in meeting the broader demands of society. The transition from traditional HPC systems to cloud-based solutions represents a significant shift in the landscape of computational resources.

A notable transformation is the substantial investment by cloud vendors in creating global networks of large-scale HPC systems. This shift is driven by several factors, including the ease of internet access, the ability to provision high-performance architectures on demand, and the reduction of operating costs. Consequently, there has been a considerable migration from executing parallel applications on dedicated HPC servers to utilizing cloud environments.

To accommodate this transition, numerous hardware and software technologies have been developed over the years. These innovations ensure the sustainable execution of HPC applications in cloud environments. Modern supercomputers are now equipped with a variety of hardware accelerators, each with unique computational capabilities. These include graphical processing units (GPUs), field-programmable gate arrays (FPGAs) [17], processing in memory (PIM) technologies [18], and, more recently, quantum processors [19].

The continual advancement in HPC architectures necessitates the development of parallel programming libraries and patterns designed to maximize the performance of these diverse systems. This chapter will provide a comprehensive discussion on several critical aspects:

1. **Demands for Multi-Core Cluster Systems:** We will explore the diverse requirements for multicore cluster systems across various fields of science and society. This includes not only the scientific and engineering domains but also other sectors that benefit from high-performance computing capabilities.

The need for multicore cluster systems spans multiple fields and sectors. In scientific research, these systems are essential for conducting complex simulations, analyzing vast datasets, and modeling phenomena that require significant computational power. In engineering, multicore clusters facilitate the design and testing of new materials, structures, and systems. Beyond these fields, sectors such as finance, healthcare, and environmental science increasingly rely on high-performance computing to solve critical problems and improve decision-making processes.

As societal reliance on data-intensive applications grows, so does the demand for robust multicore cluster systems. These systems must be capable of handling the increasing complexity and volume of data generated across various domains. Addressing these demands requires a deep understanding of the specific needs of each field and the ability to tailor computational resources accordingly.

2. **Challenges in Multi-Core Cluster Optimization:** Researchers face significant challenges in optimizing multicore cluster systems. These challenges encompass both hardware and software aspects, including performance enhancement, energy efficiency, and system resilience. We will delve into the specific issues that need to be addressed to improve the execution of applications on these systems.

   Optimizing multicore clusters involves overcoming several challenges related to hardware and software integration. Key hardware challenges include managing the thermal output of densely packed processors, ensuring efficient data transfer between CPUs and accelerators, and developing scalable memory architectures. On the software side, challenges include creating algorithms that can effectively distribute workloads across heterogeneous components, minimizing latency, and enhancing fault tolerance to ensure continuous operation.

   Energy consumption is another critical challenge. As the computational power of multicore clusters increases, so does their energy demand. Developing energy-efficient solutions without compromising performance is a significant research focus. Techniques such as dynamic voltage and frequency scaling (DVFS), energy-aware scheduling, and the use of low-power processors are being explored to address this issue.

   Resilience and reliability are also paramount. Multicore clusters must be capable of handling hardware failures, software bugs, and other disruptions without significant

performance degradation. This requires sophisticated error detection and correction mechanisms, redundant system designs, and robust recovery protocols.

3. **Trends in Heterogeneous Architectures:** With the increasing availability of heterogeneous architectures in multicore clusters, it is crucial to understand how to leverage the unique strengths of each device. We will discuss current trends and best practices for effectively utilizing GPUs, FPGAs, PIM technologies, and quantum processors to achieve optimal performance.

   The rise of heterogeneous architectures within multicore clusters presents both opportunities and challenges. Heterogeneous systems combine different types of processors, each optimized for specific tasks. For instance, GPUs excel at parallel processing tasks, making them ideal for graphics rendering and scientific simulations. FPGAs offer reconfigurable hardware that can be tailored for specific applications, providing high performance with lower energy consumption. PIM technologies integrate processing capabilities directly into memory chips, reducing data transfer times and energy use. Quantum processors, although still in their infancy, promise to solve certain types of problems exponentially faster than classical processors.

   Effectively utilizing these diverse components requires specialized programming models and optimization techniques. Researchers are developing parallel programming libraries and frameworks that can abstract the complexity of heterogeneous systems, allowing developers to focus on application logic rather than hardware specifics. Additionally, new scheduling algorithms are being designed to intelligently distribute workloads across different types of processors, maximizing overall system performance.

Based on these three key points, the challenges are classified into two major classes: scalability and heterogeneity. Scalability is defined as adding cores to a system that doesn't guarantee a proportional increase in performance. Coordinating the execution of tasks across multiple cores while maintaining efficiency poses a formidable obstacle. Moreover, achieving efficient communication and data sharing among cores is crucial, as improper synchronization can lead to bottlenecks and suboptimal performance.

In comparison, heterogeneity is defined in both hardware and workloads. Multi-core clusters often comprise diverse processors with varying architectures, speeds, and memory

capacities. Effectively harnessing this diversity to achieve maximum performance becomes complex. Additionally, the dynamic nature of workloads across a cluster introduces variability, necessitating adaptive strategies for optimal resource utilization.

Three key aspects demand particular attention to enhance the performance of multi-core systems within clusters: memory congestion, resource allocation, and load balancing.

Memory Congestion: Memory congestion arises when the demand for memory exceeds the available resources, leading to performance degradation and increased latency. In multi-core systems, efficient memory management is paramount for preventing bottlenecks and ensuring smooth data access. Addressing memory congestion involves implementing strategies that optimize memory usage, reduce contention, and enhance data retrieval times.

Resource Allocation: Effective resource allocation is critical for maximizing the utilization of computational resources within a cluster. Allocating the right amount of CPU, memory, and other resources to each task or process prevents resource starvation and ensures fair resource distribution. Dynamic resource allocation mechanisms adapt to changing workloads, optimizing performance across the cluster.

Load Balancing: Load balancing involves distributing computational tasks evenly among the available cores to prevent under-utilization or overload of specific resources. Load imbalances can lead to uneven processing times and reduced overall system efficiency. Implementing robust load-balancing mechanisms ensures that each core contributes optimally to the cluster's collective computing power.

This thesis focuses on these issues in multi-core clustering-based performance improvement and presents several challenges. Several researchers have also addressed many challenges [20, 21, 22]. Memory congestion, load balancing and performance enhancement are significant challenges in multi-core systems. In the literature, many research examples on these issues have been presented. This thesis presents the model, strategies and techniques to meet these challenges. These techniques include multi-core cluster computing-based memory congestion reduction and dependency prediction with balanced-load scheduling. By using these techniques, availability, reliability, perform ability, throughput, and resource utilization are to be improved. It can also minimize the make-span and load on the multi-core systems.

2.2 Memory Congestion in Multi-Core Systems

2.2.1 Overview

Over the past few decades, the number of cores in processors has significantly increased. These cores can be arranged in various configurations, not necessarily sequential. Some processors feature cores embedded in a parallel fashion, sharing a single cache memory within their group. This architecture can lead to memory sharing issues for several reasons, with memory congestion being the most critical, especially for high-performance computing (HPC). As system scale increases, memory congestion becomes more pronounced. System scaling can be categorized into two types: *strong scaling* and *weak scaling*. Strong scaling involves parallel communication arrangements between processors, while weak scaling faces challenges in parallel processing. The primary objective of this thesis is to mitigate memory congestion in weak scaling systems to enhance overall performance.

2.2.2 Memory congestion Problem

Memory congestion relates to the distribution of resources among all connected nodes in a network, ensuring that transactions occur at acceptable performance levels. Congestion arises when demand exceeds or approaches the network resource capacity, which includes link bandwidth, memory size, and processing capacity at intermediary nodes. Effective resource allocation is essential even under low load conditions, but the challenge intensifies as the load increases. High loads on a single node can lead to fairness issues and increased overhead, ultimately reducing system performance.

2.2.3 Challenges in Memory Management

Efficient memory management is essential in the pursuit of optimal performance in multi-core systems within cluster environments. Recent research publications have probed into this critical aspect, uncovering a spectrum of challenges that demand nuanced solutions for effective memory management. We have explored the challenges associated with memory congestion.

Memory congestion in multi-core systems arises from the complexity of concurrent processes vying for limited memory resources. Recent studies, such as the work by Wang and Jose [23], have brought to light the multifaceted challenges in memory management, emphasizing the need for tailored strategies in the face of evolving computational demands.

Based on Access Patterns: Understanding and managing diverse memory access patterns is complex task. Different applications exhibit distinct behaviour to highlight the need for adaptive memory allocation schemes that can dynamically adjust to the varying requirements of concurrent tasks.

Cache Coherency Overheads: Cache coherence, a fundamental aspect of multi-core systems, introduces overheads that can restrict memory access. A recent work by Ching-Hwa Cheng [24] explores the complexities of maintaining cache coherence in memory congestion. The study underscores the trade-offs involved in ensuring coherence while minimizing performance.

Scalability Concerns: Memory congestion and resource allocation become essential as clusters scale up. The work of Wu et al. [25] highlighted these challenges over memory management techniques to large multi-core clusters. Achieving efficient memory management across a group of cores requires innovative solutions to prevent bottlenecks and ensure uniform performance.

Dynamic Workload Variability: The dynamic nature of workloads in cluster environments introduces variability in memory usage. Zhang et al. [26] discuss the challenges of workload fluctuations and propose adaptive memory management strategies that dynamically respond to changing computational demands by using MPI.

2.2.4 Memory Congestion Reduction Techniques

Based on the challenges outlined, it is evident that memory congestion becomes a significant issue as cluster sizes or memory interconnections increase. Each node in a multi-core system is connected to local DRAM and a memory controller, allowing data access via memory controllers and interconnected links. These controllers often use high-speed connections like QP Interconnect [27]. However, despite these high-speed interconnections, accessing

data from other nodes incurs higher latency compared to accessing data from the local node.

High-Performance Computing and large-scale multi-processing systems benefit from such cores. In these environments, each processor can belong to one or more processor groups, known as nodes, which share memory, memory controllers, and interconnected links. Logically, all memory controllers are shared among all processor cores across all groups.

Memory latency is influenced by the data's location. To explicitly express data movement, this work uses MPI for its proposed algorithms. Mulya et al. [28, 29] discuss process placement strategies for different kernels like CG, FT, LU, and MG [30, 31]. Placement methods are crucial for multi-core systems and can be managed using MPI process mapping techniques [30]. Each mapping is identified by an MPI-rank, and MPI process managers allocate each process to a core based on this mapping, facilitating local data access and reducing execution penalties.

Using MPI placement strategies improves data locality by matching communication patterns to the underlying hardware. Performance enhancements occur when reducing the number of communication patterns accessed at memory controllers, especially when the number of processor cores per node is small. Fabien et al. [32] presented a study optimizing memory congestion at memory controllers, showing significant performance improvements. However, their approach introduces longer delays at the memory controller, particularly in modern multi-core systems where physical distance traversal increases latency. As core counts per node increase, memory congestion at memory controllers also rises, degrading overall performance. Hence, maintaining locality is crucial to minimize traffic on interconnected links.

Reconciling locality and congestion remains a significant challenge in modern multi-core systems [33]. This section compares different available techniques, as detailed in Table 2.1. The table categorizes algorithms based on their attributes and optimization targets, highlighting that dynamic CPTs and strict CPTs are preferred for their ability to adapt to workload behavior and enforce system-level policies, respectively. Table 2.1 illustrates the diverse range of optimizations achieved through CP and summarizes the related work on memory congestion reduction techniques. This table categorizes the cited papers according to their contributions in terms of Algorithm categories, Cache Partitioning, Optimization, Scheduling, Machine learning and Real-Time datasets.

Table 2.1: Characteristics of the category of the algorithm in the literature for memory congestion reduction technique

Algorithm	**Cache Partitioning**	**Optimization**	**Machine Learning**	**Real-Time**
Static Cache Partitioning [34, 35]	√			
Both Static and Dynamic Cache Partitioning [36, 37, 38, 39]	√			√
Pseudo-Partitioning [40, 41]	√			
Both Pseudo and Strict Partitioning [42]	√			√
Way-Based Partitioning [40, 43]	√			√
Set/Color-Based Partitioning [37, 44]	√			√
Block-Level Real-Coded GA [45, 46]		√		
Regression and Curve Fitting Optimization [36, 47, 48]		√		
Machine Learning [49, 50, 51]			√	
Dynamic Programming [52, 53]		√		
Feedback Control Theory [54]		√		
Gradient-Descent Algorithm [55, 56, 57]		√		
Partial Tag [58, 59]	√			√

2.2.5 Cache Partitioning Algorithms

Based on Table 2.1, the integration of cache partitioning techniques, both static and dynamic, offers a comprehensive solution to managing cache resources effectively, reducing memory congestion, and improving system performance in multicore environments. The below subsections further elaborates on the comparative analysis of these techniques by highlighting their strengths and areas of application.

2.2.5.1 Static Cache Partitioning

Stone et al. [34] explores static cache partitioning strategies aimed at reducing memory congestion and improving overall performance in multicore systems. By partitioning the cache statically, each core is allocated a fixed portion of the cache, ensuring that critical data remains in the cache and reducing cache misses.

Yu et al. [35] investigate methods for statically partitioning cache to optimize performance for specific applications. Their approach involves analyzing application behavior to determine optimal cache partition sizes.

2.2.5.2 Both Static and Dynamic Cache Partitioning

Kandemir et al. [36] examines the hybrid approach of using both static and dynamic cache partitioning to adapt to varying workload demands. This approach provides the flexibility of dynamic partitioning while maintaining the predictability of static partitioning.

Ye et al. [37] discusses techniques for dynamically adjusting cache partitions based on runtime analysis, while still retaining a static base partition to ensure baseline performance.

Cook et al. [38] propose a method that combines static and dynamic partitioning to balance performance and resource utilization in multicore systems.

Jiang et al. [39] explores the implementation of hybrid cache partitioning strategies that leverage both static allocation for critical tasks and dynamic adjustments based on workload changes.

2.2.5.3 Pseudo-Partitioning

Kaseridis et al. [40] introduces the concept of pseudo-partitioning, where the cache is not strictly partitioned but managed in a way that mimics partitioning, providing flexibility and efficiency.

Xie et al. [41] discovered pseudo-partitioning techniques that allow for adaptive cache management, improving performance without the overhead of strict partitioning.

2.2.5.4 Both Pseudo and Strict Partitioning

Rodrigues et al. [42] explores the combination of pseudo and strict partitioning techniques to optimize cache usage, leveraging the strengths of both approaches.

2.2.5.5 Way-Based Partitioning

Kaseridis et al. [40] present a detailed analysis of way-level partitioning techniques and their impact on multicore system performance.

Zhou et al. [43] introduces way-level partitioning, where cache ways (sets of cache lines) are allocated to different cores, providing finer granularity control over cache allocation.

2.2.5.6 Set/Color-Based Partitioning

Ye et al. [37] and Heechul et al. [44] provides an in-depth analysis of set/color-level partitioning techniques and their application in modern multicore systems. Their work focuses on set/color-level partitioning on shared cache, where cache sets are divided based on color coding to manage cache allocation more effectively. Their work also investigates the impact of cache performance and memory congestion in multicore system environment.

2.2.5.7 Block-Level Real-Coded Genetic Algorithm (GA)

Kasture et al. [45] introduces block-level real-coded GA for optimizing cache allocation, demonstrating its effectiveness in reducing memory congestion.

Sanchez et al. [46] research applies real-coded GA to block-level cache partitioning, achieving significant performance improvements. Their work further refines the use of real-coded GA in cache partitioning, highlighting its adaptability and efficiency.

2.2.6 Optimization Techniques

Based on Table 2.1, the Optimization technique offers a comprehensive solution to managing the cache resources effectively, reducing memory congestion, and improving system performance in multicore environments. The below subsections further elaborates on the comparative analysis of these techniques by highlighting their strengths.

2.2.6.1 Regression and Curve Fitting Optimization

Kandemir et al. [36] explores regression and curve fitting techniques for optimizing cache performance, using statistical models to predict and improve cache behavior.

Dongyuan et al. [47] applies regression analysis to optimize cache partitioning, demonstrating its effectiveness in enhancing system performance.

Muralidhara et al. [48] discusses various curve fitting techniques and their application in optimizing cache allocation in multicore systems.

2.2.7 Dynamic Programming

Majo et al. [53] applies dynamic programming to the problem of cache partitioning, showing significant performance gains. They have used dynamic programming techniques for cache optimization, providing a framework for making optimal cache allocation decisions.

2.2.8 Feedback Control Theory

Pellegrini et al. [54] discusses the application of feedback control theory to cache management, using control systems to dynamically adjust cache partitions based on workload changes.

2.2.9 Gradient-Descent Algorithm

Lo et al. [55] and Emmanuel et al. [58] explores the use of gradient-descent algorithms for optimizing cache allocation, demonstrating their ability to efficiently find optimal solutions.

Hasenplaugh et al. [56], Mekkat et al. [57] provides a comprehensive overview of gradient-descent algorithms and their application in HPC optimization problems.

2.2.10 Machine Learning Techniques

Based on Table 2.1, the Machine Learning technique offers a comprehensive solution to improve system performance in multicore environments. The below subsections further elaborates on the comparative analysis of these techniques by highlighting their strengths.

2.2.10.1 Machine Learning

Liu et al. [49] investigates the application of machine learning techniques to predict and optimize cache allocation, using historical data to improve performance.

Bitirgen et al. [50] discusses various machine learning algorithms and their effectiveness in optimizing cache performance in multicore systems.

Taha et al. [51] explores the potential of machine learning for HPC optimization, laying the groundwork for future research in this area.

2.2.11 Real-Time Capabilities

Emmanuel et al. [58] introduces partial tag techniques for real-time cache management, ensuring timely execution and responsiveness in HPC systems.

Stijn et al. [59] focuses on the use of partial tag methods to achieve real-time performance in multicore processors, highlighting their effectiveness in reducing latency.

2.3 Resource Allocation in Multi-Core Cluster Environments

2.3.1 Overview

In high-performance computing, multi-core clusters have emerged as essential entities, harnessing parallel processing capabilities to meet the demands of modern computational workloads. Efficiently allocating resources within multi-core collections is pivotal for optimizing performance and ensuring seamless execution of diverse tasks. This thesis explores resource allocation techniques in multi-core cluster environments, investigating challenges and advancements that outline the current state of research. Multi-core clusters, comprising interconnected processors, present a potent paradigm for parallel computing. The allocation of resources in these environments involves complex decision-making processes to balance workloads, minimize latency, and improve the parallel processing potential. The difficulties of resource allocation strengthen as cluster sizes and computational scale and innovative solutions become necessary to address concurrency, load balancing, and the heterogeneity inherent in modern clusters.

Therefore, the thesis sets the stage for an in-depth exploration of resource allocation problems in multi-core clusters, highlighting innovative solutions to improve the efficiency and scalability of these critical computing environments.

2.3.2 Challenges in Resource Allocation

Resource allocation is a fundamental aspect of multi-core computing systems. It plays a critical role in ensuring optimal performance and efficient utilization of available resources.

Resource allocation contains the reasonable distribution of computing resources such as CPU time, memory, and network bandwidth to various tasks and processes within a computing environment. The goal is to ensure a balanced computing environment that maximizes throughput, minimizes response times, and ensures a balanced utilization of resources. during this task, many challenges have been addressed by several researchers. they are listed below:

1. ***Concurrency and Load Imbalance:*** As the parallelization improves, challenges related to concurrency also occur. Allocating resources in such a way that balances the load among all the cores is a challenging task, especially when a dynamic workload is applied to the multi-core clusters.

2. ***Workload Conflict:*** Interconnected memory and cores for sharing resources pose a significant challenge. Conflict in resource requests and fulfilling the demand, can lead to suboptimal performance and increased response times.

3. ***Dynamic Workloads:*** The dynamic workloads add an additional layer of complexity which is again another challenge. Resource allocation mechanisms must be agile enough to respond to changes in demand while avoiding under-utilization or overload scenarios.

4. ***Scalability:*** Scalability is one more challenge in multi-core cluster computing environments since the demand for complex computation has increased day by day. So, the resource allocation mechanisms must exhibit that scalability. Ensuring efficient allocation in large-scale clusters or distributed environments is a persistent challenge.

5. ***Heterogeneity:*** A Heterogeneous computing environment, where nodes or processors may differ in capabilities, introduces challenges in allocating resources effectively. Strategies to address diverse hardware configurations are more crucial.

Researchers are actively engaged in devising innovative solutions to overcome these challenges. Machine learning, optimization algorithms, and adaptive resource management are areas of exploration to enhance resource allocation strategies.

2.3.3 Advanced Resource Allocation Strategies

Various approaches have been explored for resource allocation strategies, encompassing domains such as multi-core processors, GPUs, and CPUs. In architecture-based systems like multicore cluster computing, resource allocation issues are prevalent [60]. The maintenance of multicore clusters incurs substantial costs and requires large equipment and significant space. In these scenarios, data transfer between units or data access can pose considerable challenges. Researchers have made significant advancements in data parallelization and resource allocation, highlighting the efficacy of techniques like Grid Weka's data mining work [61]. This work demonstrates the power of grid computing by addressing three critical aspects: mining on CPU datasets, performance prediction, and resource allocation with workload characterization.

A promising solution to these challenges is the implementation of cache partitioning techniques. Cache partitioning can optimize resource allocation by effectively managing the cache space shared among multiple cores or processing units. By partitioning the cache, it ensures that each core or process has dedicated cache resources, reducing contention and improving data locality. This technique not only enhances performance but also reduces the latency associated with data movement and access in cluster computing environments.

In the context of multicore systems, cache partitioning can significantly alleviate the overhead of data transfer and improve the efficiency of resource utilization. When applied to multi-core systems, it can streamline the execution of parallel tasks by minimizing cache conflicts and maximizing the cache hit rate. Consequently, cache partitioning stands out as a pivotal method for addressing resource allocation challenges in complex computing systems.

Table 2.2 summarizes in-depth review of the research conducted globally on resource allocation strategies, with a focus on cache partitioning techniques. These reviews will cover advancements in multi-core systems, GPUs, and CPUs, highlighting the improvements

in performance prediction, workload characterization, and data parallelization achieved through effective cache management.

Table 2.2: Related Work on Parallel and Multicore Data Mining Systems

Work	**Data Mining Task**	**Parallel Architecture**	**GPU Utilization**	**Multicore System Performance**
Perez et al. [62]	Preprocessing, Post-processing	Grid-based		
Farg et al. [61]	k-means, Apriori Frequent Pattern Mining	GPU-CPU Co-processing	√	
Xiaohong et al. [8]				√
Tiago et al. [5]	Matrix Multiplication			√
Benjamin et al. [63]	Performance Prediction	Simulation-based	√	√
George [64]	DNN-based Model			√
Karan et al. [65]	Performance Improvement	Machine Learning	√	√
Ang li et al. [66]	Performance Improvement	Machine Learning	√	√
Zheng et al. [16]	Performance Estimation	Dynamic Coupling	√	√

2.3.4 Mining on CPU Datasets

Based on Table 2.2 this section categorizes the cited papers according to their contributions in terms of data mining tasks, parallel architectures, GPU utilization, and multicore system performance which are as follows:

Perez et al. [62] have introduced a Parallel Architecture called WekaG. The WekaG is working in two major areas:

Data Mining Task: This work introduces WekaG, an application designed for performing data mining tasks on a grid. The tasks are divided into preprocessing and post-processing steps, following a parallel architecture of data mining.

Parallel Architecture: WekaG leverages a grid-based architecture to distribute data mining

tasks across multiple nodes, enhancing computational efficiency and scalability. A similar task has bee introduced by Fang et al. [61] which is based on GPUMiner.

Data Mining Task: GPUMiner is a parallel data-mining system that implements k-means and Apriori Frequent Pattern Mining algorithms. It focuses on efficient data processing and mining.

Parallel Architecture: GPUMiner utilizes a GPU-CPU co-processing model, where both processors collaborate to handle different aspects of the data mining tasks.

GPU Utilization: The system includes a GPU-based view module and a GPU-CPU buffer manager, demonstrating significant utilization of GPU resources to accelerate data mining processes.

Xiaohong Qiu et al. and Ren et al. [8, 67] have worked on multicore system performance. There research investigates the performance of various twin CPU multicore systems, comprising 4 to 8 cores. The study reports significant speedup and performance improvements, particularly in large space production environments. The findings highlight the effectiveness of multicore clusters in executing data-intensive tasks efficiently.

Tiago et al. [5] works specifically examines the performance of matrix multiplication algorithms on multicore systems. The study presents compelling results, showcasing the advantages of multicore systems for computational tasks, even though a more effective method for distributing data-mining equipment on a single machine was not identified.

2.3.5 Performance Prediction

Simulation-based Performance Prediction

Benjamin et al. [63], Sameh et al. [68]: These researchers have demonstrated methods for predicting CPU performance using simulation-based techniques. Their work criticizes the micro-architecture features of CPUs to make accurate performance predictions.

DNN-based Model

George et al. [64]: This work employs a Deep Neural Network (DNN)-based model, a

cutting-edge algorithm in the realm of machine learning. The model utilizes features such as buffer size for read-write operations and pipeline stages for performance prediction, excluding specific micro-architectural features.

Workload Prediction Using Mechanistic Models

Allan et al. [69] studies about the development of mechanistic models to predict CPU workload. The models use empirical data to enhance the performance prediction of multicore systems.

Empirical Models and Machine Learning

Sameh et al. [68], Karan et al. [65], and Ang et al. [66]: These researchers apply empirical models alongside machine learning techniques to boost the performance of multicore systems. Newsha et al. [70], and Xinnian et al. [71]: Their work integrates machine learning models to further improve system performance.

Dynamic Coupling-based Performance Estimation

Zheng et al. [16]: This study uses a dynamic coupling-based performance counter method for estimating performance, leveraging micro-architectural features for accurate predictions.

Machine Similarity and Data Transposition

Liu et al. [72]: This research employs data transposition techniques to determine machine similarity, contributing to the selection and performance optimization of CPU components.

2.3.6 Resource Allocation and Workload Characterization

Phansalkar et al. have made significant contributions to workload characterization in their studies [20, 73]. However, their research does not address the performance and scalability of CPUs, leading to inconsistencies in their results. In contrast, our study aims to include performance prediction. Shelepov et al. [74, 75] utilized static workload features such as cache size, spatial locality, and frequency to predict performance. They also projected speedups across various processors.

Additionally, several researchers have explored cloud scheduling to analyze workload characterization [76]. Hoste et al. conducted a feature-based study that focused on features independent of micro-architecture for workload classification. They considered locality and Instruction Level Parallelism (ILP) metrics for performance prediction. The locality features they examined include LRU stack models, independent reference models, density functions (both temporal and spatial), set models, and memory reuse distance models.

Table 2.3 classifies the works based on their architecture, evaluation technique and resource allocation. All the references cited in the table are briefly described below:

Wang et al. [77] introduces PadWS, a novel load balancing strategy designed to optimize work stealing in multi-core systems. Traditional work-stealing approaches often suffer from performance bottlenecks due to synchronization overhead and contention caused by concurrent access to task queues. PadWS addresses these issues through a block-based design, which significantly reduces metadata contention. In this approach, the queue owner and task thieves operate on separate blocks, thereby minimizing conflicts. Moreover, PadWS employs an asynchronous delegation strategy, eliminating the need for idle processors to spin-wait after sending a request. This method enhances scheduling efficiency and overall system performance.

Jadon et al. [78] provides a comprehensive analysis of load balancing techniques for managing mixed real-time tasks on multi-core systems. Real-time systems must prioritize, manage, and execute tasks within stringent time constraints to ensure timely results. Their work examines various load balancing algorithms that aim to evenly distribute the workload among processors, thereby maximizing their utility and minimizing execution time. Key performance metrics evaluated include throughput, response time, migration time, and overhead. The study also highlights the challenges of implementing these techniques in real-time contexts and suggests potential avenues for further research to improve load balancing efficacy.

W. Li et al. [79] focuses on optimizing Sparse Matrix-Vector Multiplication (SpMV) on asymmetric multi-core processors (AMPs). These processors feature heterogeneous cores with varying performance characteristics, such as Intel's P- and E-cores or AMD's cores with and without 3D V-Cache. The proposed parallel algorithm, HASpMV, assigns workloads based on the performance characteristics of different cores to achieve better

Table 2.3: Comparison of resource allocation and workload distribution across various hardware accelerator architectures

Reference	Performance Metric	Evaluated Application	Core Technique
Wang et al. [77]	Reducing metadata contention	Task scheduling	Partial and asynchronous delegated work-stealing
—	avoiding spin-wait	—	—
Jadon et al. [78]	Throughput	Mixed real-time tasks	Load balancing for real-time workloads
—	performance (up to 4.18x)	—	—
—	response time (reduced)	—	—
W. Li et al. [79]	Speedup (up to 9.46x)	Sparse matrix-vector multiplication (SpMV)	Heterogeneity-aware SpMV
Manumachu et al. [80]	Execution time	Matrix multiplication	Parallel data partitioning
—	energy consumption	—	—
—	memory costs	FFT	—
Omar et al. [81]	Deterministic performance	Safety-critical applications	Isolated core clusters
—	security	—	—
—	resiliency	—	—
Shaw et al. [82]	Max(Speedup): 8×	Fast Fourier Transform	DRAM access optimization
—	Max(Speedup): 6.5×	Fast Fourier Transform	DRAM access optimization
Richard et al. [83]	Max(Speedup): 8.68×	Data parallel applications	Multi-port cache parallel access
Mejri et al. [84]	Processing rate: 35 frames/s	AlexNet	On-chip/off-chip memory minimization
—	Processing rate: 0.7 frames/s	IAT(VGG-16)	Accelerated address translation
Villavieja et al. [85]	Performance gain: 1.36×	Capsule Networks	Reduced on-chip memory size and off-chip access

cache locality and load balancing. Experimental results demonstrate significant performance improvements, with HASpMV achieving speedups of up to 9.46x compared to existing libraries and algorithms. This work highlights the importance of heterogeneity-aware optimization in modern multi-core systems.

Manumachu et al. [80] introduces parallel data partitioning algorithms designed to optimize workload distribution in self-adaptable data-parallel applications on extreme-scale multi-core platforms. The algorithms address challenges such as resource contention and non-uniform memory access, which are inherent in modern multi-core architectures. By employing dynamic programming techniques, the proposed algorithms achieve globally optimal workload distributions with low time complexity and minimal memory overhead. The study includes practical evaluations on applications like matrix multiplication and Fast Fourier Transform (FFT), demonstrating that the algorithms offer substantial speedups and low runtime costs, making them suitable for deployment in self-adaptable systems.

Omar et al. [81] proposes a novel framework for multi-core systems aimed at executing safety-critical applications that require deterministic, resilient, and secure environments. The framework constructs isolated clusters of cores for each concurrent application, ensuring deterministic performance by preventing interference from other applications. This approach also enhances security by mitigating side-channel attacks that exploit shared hardware resources. Additionally, the framework supports dynamic resizing of core clusters to maintain load-balanced execution. However, this flexibility introduces trade-offs between performance, resilience, and security, which the framework manages effectively to meet the stringent requirements of safety-critical applications.

Shaw et al. [82] focus on optimizing the Fast Fourier Transform (FFT) by improving DRAM access patterns in multi-core systems. Their optimization techniques lead to a maximum speedup of 8× and 6.5× for FFT tasks. By enhancing memory access efficiency, the proposed method reduces latency and increases throughput, effectively balancing the load across the system's cores.

Richard et al. [83] present a load balancing strategy for data parallel applications that leverages multi-port cache parallel access. This technique achieves a maximum speedup of 8.68×, demonstrating its effectiveness in optimizing resource utilization and minimizing

access conflicts in multi-core environments. The approach enhances data throughput and reduces contention, leading to better overall system performance.

Mejri et al. [84] investigate load balancing in the context of deep learning applications, specifically focusing on AlexNet and VGG-16 architectures. Their work achieves a processing rate of 35 frames per second for AlexNet by minimizing on-chip and off-chip memory usage. For VGG-16, the approach includes accelerated address translation, resulting in a processing rate of 0.7 frames per second. These techniques optimize memory management, crucial for handling the intensive computational demands of deep learning tasks on multi-core systems.

Villavieja et al. [85] explore load balancing for Capsule Networks by reducing on-chip memory size and off-chip access. Their approach yields a performance gain of 1.36×, highlighting the importance of efficient memory utilization in enhancing the performance of neural network models on multi-core platforms. The reduced memory footprint and improved access patterns contribute to better load distribution and faster processing times.

These works collectively advance the field of load balancing in multi-core systems, offering various strategies to optimize performance, resource utilization, and efficiency across different types of applications.

2.4 Load Balancing in Multi-Core Cluster Environments

2.4.1 Overview

Efficient load balancing is critical in multi-core clustering environments. Several researchers have proposed different methods, most finding solutions through heuristics and meta-heuristic techniques. This section delves into load-balanced transaction scheduling in a multi-core cluster environment, leveraging a detailed comparison between discovered methods in the literature. Our exploration focuses on achieving optimal execution characteristics for a specified set of transactions within their stipulated deadlines.

A detailed classification of Load Balancing techniques has shown in Figure 2.2 which is based on multicore clustering environment. The Load Balancing techniques are divided

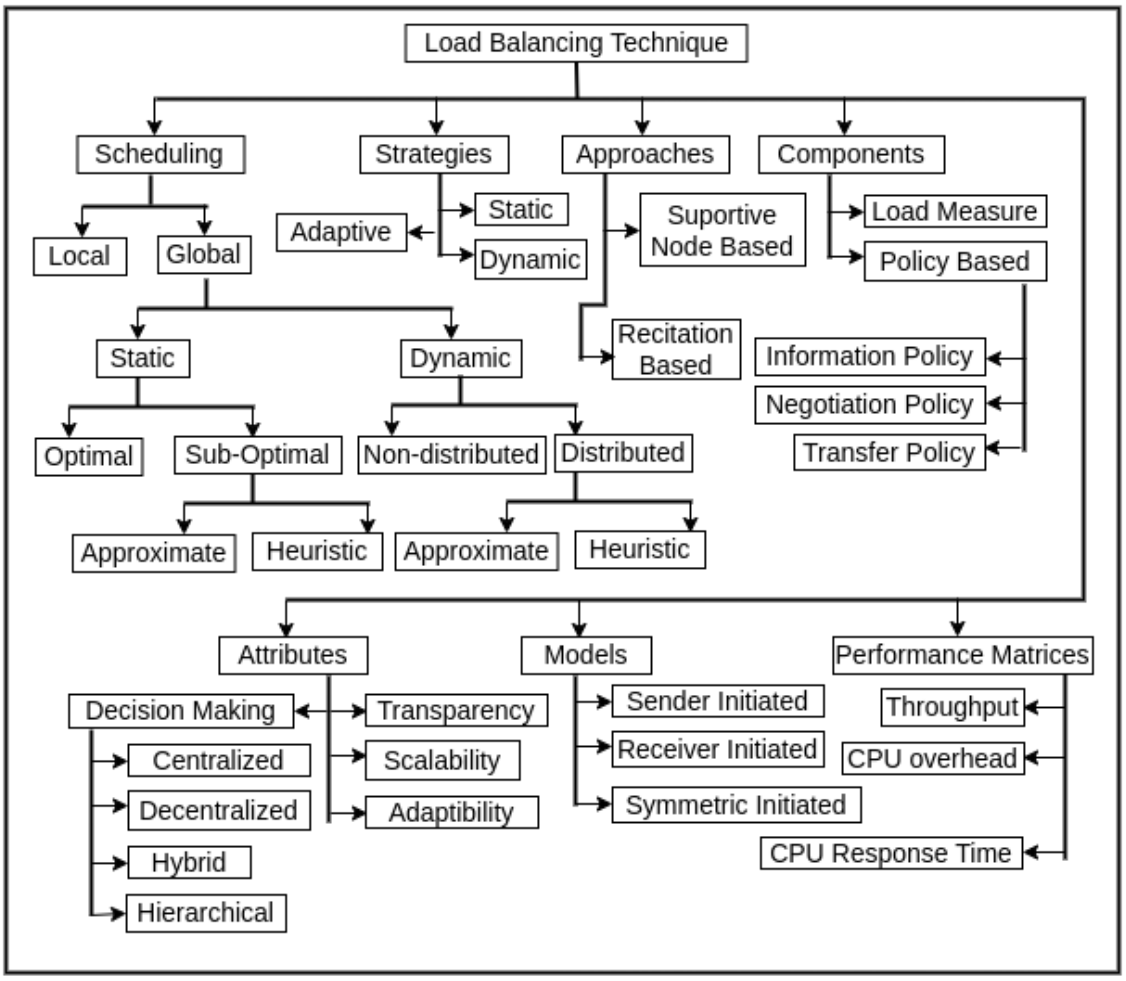

Figure 2.2: Classification of load balancing techniques based on HPC environment

into its primary categories, based on load distribution time, system architecture, load balancing strategy, and load information availability. Each category further subdivides into specific techniques, illustrating the comprehensive taxonomy of load balancing methods for multi-core clusters. From the Figure 2.2, it is clear that the load balancing techniques in multi-core clusters can be categorized into seven primary classes, each with its own set of subcategories. These classifications help in understanding and evaluating various load balancing methods comprehensively. All classes are briefly explained below:

- **Scheduling:** This class is divided into two main subclasses: local and global. The global scheduling is further split into static and dynamic. The static scheduling is divided into optimal and suboptimal methods, where suboptimal is further classified into approximate and heuristic techniques. On the other hand, dynamic scheduling is

divided into non-distributed and distributed classes, with the distributed class further divided into approximate and heuristic methods.

- **Strategies:** This category includes three classes: adaptive, static, and dynamic strategies. These strategies define how the system responds to varying loads and resource availability.
- **Approaches:** Approaches are divided into two subclasses: supportive and recitation-based. These approaches describe different methodologies for implementing load balancing.
- **Components:** This class is divided into five subclasses: load measure, policy-based, information policy, negotiation policy, and transfer policy. These components are crucial for determining how load balancing decisions are made and executed.
- **Attributes:** Attributes are classified into four categories: transparency, scalability, adaptability, and decision-making. The decision-making attribute is further divided into four subclasses: centralized, decentralized, hybrid, and hierarchical, each representing different decision-making structures.
- **Models:** Models are classified into three subclasses: sender-initiated, receiver-initiated, and symmetric-initiated. These models describe the initiation point of the load balancing process, whether it starts from the sender, receiver, or both.
- **Performance Metrics:** Performance metrics are divided into three subclasses: throughput, CPU overhead, and CPU response time. These metrics are used to evaluate the effectiveness of load balancing techniques.

By categorizing load balancing techniques into these seven classes and their respective subclasses, this section provides a comprehensive framework for understanding the various methods and strategies employed in multi-core clusters. This detailed classification helps in identifying the strengths and weaknesses of different load balancing approaches, facilitating the development of more efficient and effective techniques.

2.4.2 Load Balancing Techniques

Efficient load balancing is crucial in multi-core cluster computing environments, particularly for Decision making structure, CPU execution, virtual organization, Load redistribution, information freshness and horizon of load balancing with strict deadlines. Balanced task allocation aims to optimize resource availability by distributing tasks to nodes in a way that prevents overload and maximizes system throughput. However, the challenge is compounded by the possibility of resource failure at any time at any node in the cluster, making this a highly complex problem. Numerous studies have explored various load balancing and resource allocation strategies in different computing environments. Implementing these strategies effectively involves identifying the optimal node for each transaction to meet its deadline, thus ensuring balanced task allocation and efficient transaction processing is a complex and challenging task in multicore clusters.

Table 2.4 classifies the works based on their Evaluation Technique, Key Features and Performance Metrics. From the Table 2.4 Sam et al. [86] introduced a novel heuristic algorithm that integrates multiple techniques, such as a customized analytic hierarchy process (AHP), bandwidth based scheduling (BSS), optimization using Bar-Truss Model, and a shared memory based DAC strategy. This comprehensive approach resulted in notable enhancements in turnaround and response times, thereby improving CPU, memory, and bandwidth utilization.

PadWS (Partial and Asynchronous Delegated Work-Stealing), introduced by Wang et al. [77], employs a block-based design to minimize metadata contention between the queue owner and thieves. By allowing asynchronous delegation, PadWS reduces the need for thieves to spin-wait, thereby improving load balancing efficiency and reducing synchronization overhead.

Real-time Load Balancing for Mixed Real-Time Tasks, discussed by Jadon et al. [78], focuses on the challenges of executing real-time workloads on multi-core systems. This technique evaluates several factors, including throughput, performance, migration time, response time, overhead, and resource utilization. It aims to distribute the load evenly among processors to maximize utility and minimize execution time, thereby enhancing task prioritization and execution.

Table 2.4: Comparison of Load Balancing Techniques in Multi-Core Cluster Environments

Reference	Technique	Key Features	Performance Metrics
Wang et al. [77]	PadWS (Partial and Asynchronous Delegated Work-Stealing)	reduced metadata contention, asynchronous delegation	Improved load balancing efficiency, reduced synchronization overhead
Jadon et al. [78]	Load balancing for mixed real-time tasks	Evaluation of throughput, performance, migration time, response time, overhead, resource utilization	Enhanced task prioritization and execution
W. Li et al. [79]	HASpMV (Heterogeneity-Aware SpMV)	Bandwidth optimization, workload distribution between different cores	better cache locality and load balancing
Manumachu et al. [80]	Parallel data partitioning algorithms	DP approach, low runtime and memory costs	Linear speedup (O(p)), globally optimal workload distribution, low execution times
Omar et al. [81]	Isolated core clusters	Deterministic performance, dynamic resizing of clusters, trade-offs between performance, resilience, and security	Improved resilience and security, efficient execution environment for concurrent applications

HASpMV (Heterogeneity-Aware SpMV), explored by W. Li et al. [79], is designed to improve the performance of sparse matrix-vector multiplication on asymmetric multicore processors. By optimizing bandwidth and distributing workloads between different types of cores, HASpMV achieves significant speedups, ranging from 2.61× to 9.46× depending on the processor. This approach enhances cache locality and load balancing.

Parallel Data Partitioning Algorithms, from Manumachu et al. [80], address the complexities of resource contention and non-uniform memory access in extreme-scale multicore platforms. Utilizing a dynamic programming approach, these algorithms achieve linear speedup (O(p)) and globally optimal workload distribution with low runtime and memory costs, making them ideal for self-adaptable data-parallel applications.

Omar et al. [81], propose a framework for constructing isolated clusters of cores for concurrent applications, ensuring deterministic performance and dynamic resizing of clusters. This method balances performance, resilience, and security, providing an efficient execution environment for safety-critical applications by mitigating interference and vulnerabilities due to resource sharing.

Figure 2.2 shows a classification of Load Balancing Techniques. Overall, these techniques collectively advance the state-of-the-art in load balancing for multi-core cluster environments, addressing various challenges and optimizing performance across diverse applications.

2.4.3 Dynamic Load Balancing

High-Performance Computing (HPC) environments are characterized by the need for efficiently utilizing computing resources to handle complex and data-intensive applications. Dynamic load balancing plays a pivotal role in optimizing resource usage and enhancing overall system performance. Dynamic load balancing involves the strategies of real-time adjustment of computational workloads across a cluster of computing nodes to ensure equitable resource utilization. In an HPC setting, where tasks vary in computational intensity and execution time, effective load balancing becomes critical for maximizing throughput and minimizing response times. The Methods of Dynamic Load Balancing involves the following techniques:

Task Migration Techniques: Dynamic load balancing often involves migrating tasks between computing nodes based on the current system state. Algorithms such as task scheduling and task migration heuristics, dynamically redistributed tasks to idle or underutilized nodes, optimizing resource usage, are some of the task migration techniques.

Prediction-Based Approaches: Predictive load balancing methods leverage historical data and predictive models to forecast future resource requirements. Machine learning algorithms can be employed to predict task execution times and allocate resources accordingly.

Work Queue Strategies: Work queues facilitate the efficient distribution of tasks among computing nodes. Load balancing algorithms, like work stealing and work-first, dynamically adjust the distribution of tasks within the queue to maintain a balanced workload are the best examples.

By applying these techniques the performance may definitely improve. A list of performance measure matrices are discussed below in brief.

Throughput Enhancement: Dynamic load balancing minimizes idle time on computing nodes, ensuring continuous utilization and thereby increasing overall system throughput.

Response Time Reduction: By redistributing tasks based on real-time conditions, dynamic load balancing reduces task response times, improving the overall responsiveness of the system.

Resource Utilization Optimization: Efficient load balancing ensures that each computing node operates near its capacity, optimizing resource utilization and avoiding under-utilization or overload scenarios.

Scalability and Flexibility: Dynamic load balancing methods contribute to the scalability and flexibility of HPC environments, enabling them to adapt changes made in workloads based on their resource availability.

Dynamic load balancing techniques in multicore clusters can be categorized into two main types based on their approach: Supportive Node-Based and Recitation-Based.

Supportive Node-Based:Supportive Node-Based Techniques involve assigning additional

support nodes to assist with load balancing. These support nodes monitor the system's workload and dynamically redistribute tasks among the processing nodes to ensure an even load distribution and optimal performance.

Recitation-Based:Recitation-Based Techniques rely on periodic reevaluation and redistribution of tasks. In this approach, the system regularly assesses the current load on each node and reassigns tasks to balance the load dynamically, responding to changes in workload and system performance in real-time.

Dynamic load balancing techniques in multicore clusters can be further classified based on their components, specifically focusing on load measurement and policy-based approaches.

Load Measurement involves assessing the current load on each processing node. This component ensures that the system continuously monitors the workload distribution, enabling timely and accurate decisions for load balancing.

Policy-Based Components define the rules and strategies for how tasks are assigned and redistributed among the nodes. These are further divided into three categories:

Information Policy: Determines how and when the system collects information about the current load on each node. This policy ensures that the system has up-to-date data to make informed load balancing decisions.

Negotiation Policy: Establishes the rules for how nodes communicate and negotiate with each other to balance the load. This policy facilitates coordination between nodes to achieve an even distribution of tasks.

Transfer Policy: Dictates the conditions under which tasks are transferred from one node to another. This policy helps in deciding which tasks to move and where to move them to optimize the overall system performance.

These components work together to ensure dynamic and efficient load balancing in multicore cluster environments. We have thoroughly analyzed and summarized general load-balancing techniques in Table 2.5. Zhu et al. [87] developed a Load Cognition-Based Migration algorithm using a two-step VM migration approach, which effectively reduces traffic by 35%. However, this approach requires more extensive real-world testing to confirm its effectiveness.

Table 2.5: Summary of Load Balancing Techniques

Reference	Proposed Algorithm	Technique Used	Advantages	Disadvantages
Zhu et al. [87]	Load Cognition-Based Migration	Two-step VM migration	Reduces traffic by 35%	Needs more real-world testing
Nagamani H. et al. [88]	Resource Management Model	VM migration	Addresses overload by data redistribution	Dependent on accurate overload detection
Ivanoe et al. [89]	Extremal Optimization	Parallel task execution in dynamic settings	Lower execution time, fewer task transfers	Complexity in dynamic environments
Tan et al. [33]	PGAS + X Programming Style	Multithreading and one-sided communication	Improved performance, reduced memory per core	High programming complexity
Hamidreza et al. [12]	Model-Based Data Partitioning	Data partitioning algorithm for heterogeneous HPC	Correctness, efficiency, performance optimization	High complexity, limited memory for accelerators
Omar et al. [81]	Isolated Clusters Framework	Core isolation for deterministic performance	Resilient and secure execution environment	Performance-resilience and performance-security tradeoffs
Manumachu et al. [80]	Parallel Data Partitioning Algorithms	Dynamic programming approach	Low runtime and memory costs, high speedups	High complexity for extreme-scale platforms
Hu et al. [90]	FinD	Fine-grained data processing with NUMA-aware work stealing	Significant speedup, reduced execution time	Complexity in handling data skew
Li et al. [79]	HASpMV	Heterogeneity-aware SpMV on AMPs	Improved cache locality and load balancing, significant speedup	Limited testing on diverse architectures

Nagamani H. et al. [88] proposed a Resource Management Model based on VM migration, focusing on addressing overload issues through data redistribution. The success of this model depends heavily on the accuracy of overload detection.

Ivanoe et al. [89] developed an Extremal Optimization technique for parallel task execution in dynamic environments. This method reduces execution time and task transfers but is complex to implement in dynamic settings.

Tan et al. [33] implemented a PGAS + X Programming Style for the Barnes-Hut algorithm on multi-core clusters. This method integrates multithreading and one-sided

communication, leading to improved performance and reduced memory usage per core. However, it has high programming complexity.

Hamidreza et al. [12] introduced a model-based data partitioning algorithm tailored for heterogeneous HPC platforms, optimizing performance while ensuring correctness and efficiency. The approach is complex and has limited memory for accelerators.

Omar et al. [81] presented an isolated clusters framework that ensures deterministic performance in multi-core systems, providing a resilient and secure execution environment. However, it involves tradeoffs between performance, resilience, and security.

Manumachu et al. [80] developed parallel data partitioning algorithms using a dynamic programming approach, achieving low runtime and memory costs with significant speedups. This approach is complex, especially for extreme-scale platforms.

Hu et al. [90] introduced FinD, a fine-grained data processing framework with NUMA-aware work stealing, resulting in significant speedups and reduced execution time. However, handling data skew remains complex.

Li et al. [79] proposed HASpMV, a heterogeneity-aware SpMV algorithm for asymmetric multicore processors, enhancing cache locality and load balancing with significant speedups. The method has limited testing on diverse architectures.

2.5 Integration of Memory Congestion, Resource Allocation, and Load Balancing

In a heterogeneous cluster environment, resources are distributed across different nodes and categorized into clusters based on parameters such as server performance and storage capacity. This section reviews various cluster-based techniques, with a summary provided in Table 2.6. The table includes works focused on integrating memory congestion, resource allocation, and load-balancing techniques, such as VM migration and load estimation algorithms, according to load-balancing parameters.

Ahmed Eleliemy and Florina M. Ciorba [91] proposed a distributed self-scheduling model for improving load balancing. This model focuses on scalability and overall performance

Table 2.6: Summary of Cluster-Based Load-Balancing Techniques

Reference	Algorithm Used	Technique Used	Advantages	Disadvantages
Eleliemy and Ciorba [91]	Distributed Self-Scheduling Model	Utilizes distributed self-scheduling schemes for load balancing	Enhances scalability and performance, reduces communication overhead	Initial setup can be complex
Liu et al. [72]	Heterogeneity-Aware Scheduler	Proportional training workload assignment, feasible solution categorization, matching markets	Minimizes Job Completion Time (JCT), high scheduling fairness, reduced computational complexity	Limited testing on diverse workloads
Yao et al. [92]	Load-Balanced Batching (LBB)	Performance modeling, coordinating local batch sizes	Eliminates stragglers, significantly reduces training time, improves training efficiency	Implementation complexity in PyTorch
Yan et al. [93]	HSAS	Systolic array performance and energy models, task decomposition, subtask priority management	Improves turnaround time, system throughput, fairness, and efficiency	Complexity in managing heterogeneous systolic arrays
Taha et al. [51]	Machine Learning-Based Load Balancing	Real-time load monitoring, load estimation, application assignment	Reduces system imbalance, workload execution time, and device idle periods, improves energy efficiency	Complexity in implementing and validating the approach in real-time systems

by reducing communication overhead, leading to improved system scalability and performance.

Liu et al. [72] proposes a heterogeneity-aware scheduler that addresses multi-job placement in heterogeneous deep learning clusters. The algorithm uses proportional training workload assignment, feasible solution categorization, and matching markets to minimize average Job Completion Time (JCT) and improve scheduling fairness while maintaining low computational complexity.

Yao et al. [92] introduces Load-Balanced Batching (LBB) for distributed deep learning (DDL) tasks on heterogeneous GPU clusters. By modeling performance and coordinating local batch sizes, LBB eliminates stragglers and significantly reduces training time, improving overall training efficiency. However, it requires complex implementation in

PyTorch.

Yan et al. [93] develops HSAS, a scheduling algorithm for heterogeneous systolic array accelerator clusters. HSAS uses performance and energy models, task decomposition, and subtask priority management to optimize processing efficiency, improving turnaround time, system throughput, and fairness. Managing heterogeneous systolic arrays adds complexity.

Taha et al. [51] offers a machine learning-based load balancing approach for heterogeneous systems. It monitors real-time load distribution, estimates loads, and assigns applications to underloaded devices, minimizing system imbalance and improving energy efficiency. The approach requires complex implementation and validation in real-time systems.

2.6 Summary

This chapter provides a comprehensive review of techniques aimed at reducing memory congestion, optimizing resource allocation, implementing load-balancing strategies, and assessing performance through various algorithms. The majority of the reviewed research focuses on homogeneous user requests and environments, with limited exploration of resource allocation in heterogeneous settings. It is evident that most studies emphasize uniform scenarios, leaving a significant gap in addressing the complexities of heterogeneous environments.