ISSN Online : 2709-5088
ISSN Print : 2709-507X

INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY (IJCST) Vol. 4 No. 1 (2020)

NUMA-AWARE REAL-TIME WORKLOADS

Syed Afraz Ali
(https://orcid.org/0009-0001-6872-6786)

Abstract:
Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. With the advent of multi-core processors and complex applications, managing real-time workloads efficiently on systems with multiple NUMA nodes has become increasingly important. This paper provides a comprehensive analysis about "NUMA- Aware Real-Time Workloads". It covers various aspects of managing real-time workloads on systems with multiple NUMA nodes, including the importance of NUMA-aware resource isolation, in-depth deployment patterns, use cases, and associated caveats. The paper discusses the tools and strategies for effective NUMA-aware resource isolation, such as CPU pinning, memory binding, and workload placement strategies. It also explores various deployment patterns, their use cases, and the challenges associated with implementing NUMA-aware strategies. The paper concludes by summarizing the vital role of NUMA-aware resource isolation and deployment patterns in optimizing the performance of real-time workloads and highlights the need for careful planning, ongoing attention, rigorous testing, performance monitoring, and iterative adjustments. This analysis aims to provide insights and recommendations for organizations looking to optimizethe performance of their real-time applications while managing the inherent complexities associated with NUMA-aware resource isolation.

Keywords:
NUMA, Non-Uniform Memory Access, real-time workloads, resource isolation, CPU pinning, memory binding, workload placement, deployment patterns, load balancing, data locality optimization, thread and core affinity, dynamic resource management, scheduling policies, financial trading systems, online gaming servers, telecommunications, industrial automation, robotics, aerospace applications, defense applications, medical imaging, healthcare, video streaming, content delivery, autonomous vehicles, high-performance computing, telemedicine, remote collaboration, performance variability, deadlocks, platform-specific considerations, hardware architecture, system configuration, memory access, latency, contention, performance optimization, system management, maintenance overhead.

1. Introduction

The rapid advancement in computing technology has led to the development of multi-core processors and complex applications that demand efficient management of real-time workloads. One critical aspect of this management is the Non-Uniform Memory Access (NUMA) architecture, a computer memory design used in multiprocessing systems. This introduction aims to provide a background on NUMA and discuss the importance of NUMA-aware resource isolation in managing real-time workloads.

1.1 Background on NUMA

In a NUMA system, multiple processors, or nodes, are connected to a shared memory, and each processor has its local memory. The time it takes for a processor to access memory depends on the memory location relative to the processor. Accessing local memory (memory attached to the processor) is faster than accessing remote memory (memory attached to another processor). This design contrasts with the Uniform Memory Access (UMA) architecture, where all processors access the memory with uniform latency. NUMA architecture aims to overcome the scalability limitations of UMA by providing processors with faster access to local memory while still allowing access to remote memory when necessary.

1.2 Importance of NUMA-Aware Resource Isolation

Managing real-time workloads efficiently on systems with multiple NUMA nodes is crucial for applications that require timely and predictable performance. Real-time workloads, such as financial trading systems, online gaming servers, and autonomous vehicles, demand consistent low latency and high throughput. NUMA-aware resource isolation involves assigning tasks to specific CPU cores and memory regions to minimize data movement between NUMA nodes, thereby reducing latency and improving performance. Tools like CPU pinning, memory binding, and workload placement strategies can help achieve effective NUMA-aware resource isolation for real-time tasks. Implementing NUMA-aware strategies can significantly enhance the efficiency and responsiveness of real-time applications by minimizing memory latency, reducing contention, and ensuring predictable execution times for critical tasks. However, it also introduces complexities in system configuration, management, and maintenance, which require careful planning and ongoing attention. Ultimately, NUMA-aware resource isolation plays a vital role in optimizing the performance of real-time workloads within systems with multiple NUMA nodes.

2. NUMA-Aware Resource Isolation

2.1 Overview

In a computing environment, resource isolation is a crucial aspect of ensuring that applications run efficiently and without interference from other processes. This is particularly important in systems with multiple NUMA nodes, where the memory access time varies depending on the memory location relative to the processor. NUMA-aware resource isolation involves strategically assigning tasks to specific CPU cores and memory regions to minimize data movement between NUMA nodes, thereby reducing latency and improving overall performance.

In a NUMA system, each processor, or node, has its local memory, and the time it takes for a processor to access memory depends on whether the memory is local (attached to the processor) or remote (attached to another processor). Accessing local memory is faster than accessing remote memory. Therefore, it is beneficial to assign tasks to processors in a way that maximizes access to local memory and minimizes access to remote memory. This involves considering the NUMA topology of the system, which includes the layout of the processors, memory, and interconnections.

NUMA-aware resource isolation is particularly important for managing real-time workloads, which require timely and predictable performance. Real-time applications, such as industrial automation, telecommunications, and online gaming, demand consistent low latency and high throughput. By strategically assigning tasks to specific CPU cores and memory regions, it is possible to minimize memory latency, reduce contention, and ensure predictable execution times for critical tasks.

2.2 Tools and Strategies

There are several tools and strategies that can be employed to achieve effective NUMA-aware resource isolation for real-time tasks:

CPU Pinning: This involves binding a specific task or thread to a particular CPU core. By pinningtasks to specific cores, it is possible to control the execution of tasks and ensure that they remain on their assigned cores. This reduces thread migration and contention, improving predictability forreal-time tasks.

Memory Binding: This involves binding the memory allocation of a task to specific NUMA nodes.By binding the memory allocation of a task to the NUMA node where the task is running, it is possible to ensure that the task accesses local memory as much as possible, minimizing the overhead of inter-node communication.

Workload Placement Strategies: This involves strategically placing tasks on specific NUMA nodesbased on the availability of resources and the characteristics of the workload. For example, if a task is memory-intensive, it may be beneficial to place it on a NUMA node with a large amount of available memory. Similarly, if a task is CPU-intensive, it may be beneficial to place it on a NUMA node with a high number of available CPU cores.

NUMA-Aware Load Balancing: This involves implementing load balancing mechanisms that consider the NUMA topology. By distributing tasks across NUMA nodes based on the availability of resources and minimizing cross-node communication, it is possible to optimize resource utilization and improve overall performance.

NUMA-Aware Data Structures: This involves designing data structures that are NUMA-aware. By allocating memory for each NUMA node and managing data placement explicitly, it is possibleto minimize remote memory accesses and improve data locality.

NUMA-Aware Scheduling Policies: This involves customizing scheduling policies to prioritize real-time tasks and consider the NUMA topology when making scheduling decisions. By prioritizing real-time tasks and considering the NUMA topology, it is possible to ensure that real-time tasks are scheduled on the most appropriate NUMA nodes and that they receive the necessary resources to meet their performance requirements.

Implementing these tools and strategies requires a deep understanding of the hardware architecture and the characteristics of the workload. It may also involve fine-tuning at both the application and system levels to achieve the desired performance outcomes. Ultimately, NUMA-aware resource isolation plays a vital role in optimizing the performance of real-time workloads within systems with multiple NUMA nodes.

3. In-Depth Deployment Patterns

Managing real-time workloads efficiently on systems with multiple NUMA nodes requires strategic deployment patterns that consider the NUMA topology and the characteristics of the workload. Here are some in-depth deployment patterns that can enhance the performance and efficiency of real-time workloads:

3.1 Single NUMA Node Deployment

In this deployment pattern, the real-time workload is deployed on a single NUMA node to minimize memory latency and contention. This is suitable when the workload's resource requirements can be met by a single node. By deploying the workload on a single NUMA node, it is possible to ensure that the task accesses local memory as much as possible, minimizing the overhead of inter-node communication. However, this approach may not be suitable for larger workloads that require more resources than a single NUMA node can provide.

3.2 NUMA Node Partitioning

For larger workloads, it may be necessary to divide the application into smaller partitions and deploy each partition on a separate NUMA node. This reduces contention and improves data locality within each partition. By dividing the application into smaller partitions and deploying each partition on a separate NUMA node, it is possible to optimize resource utilization and improve overall performance. However, this approach may introduce challenges related to data sharing and communication between partitions.

3.3 Memory Binding and CPU Pinning

This involves binding the workload's memory allocation to specific NUMA nodes and pinning

CPU threads to cores on those nodes. By binding the memory allocation of a task to the NUMA node where the task is running, and pinning the CPU threads to cores on those nodes, it is possible to ensure that data remains local to the assigned memory and minimizes the overhead of inter-node communication. This approach ensures that data remains local to the assigned memory and minimizes the overhead of inter-node communication. However, it may introduce challenges related to resource fragmentation, where some nodes are underutilized while others are overburdened.

3.4 NUMA-Aware Load Balancing

This involves implementing load balancing mechanisms that consider the NUMA topology. By distributing tasks across NUMA nodes based on the availability of resources and minimizing cross-node communication, it is possible to optimize resource utilization and improve overall performance. However, this approach may introduce challenges related to dynamic workload changes, as workloads may change over time, and static deployment patterns may become suboptimal.

3.5 Hybrid NUMA Awareness

In some applications, there may be a mix of real-time and non-real-time tasks. In such cases, it may be beneficial to deploy real-time tasks on a dedicated NUMA node while letting non-real-time tasks run on other nodes. This avoids interference with critical workloads and ensures that real-time tasks receive the necessary resources to meet their performance requirements. However, this approach may introduce challenges related to resource fragmentation and may require careful planning and ongoing attention to ensure optimal performance.

3.6 Data Locality Optimization

This involves profiling the workload to understand memory access patterns and placing data structures and frequently accessed memory on the same NUMA node as the processing cores that need them. By optimizing data locality, it is possible to minimize remote memory accesses and improve overall performance. However, this approach may introduce challenges related to data sharing and communication between tasks running on different NUMA nodes.

In conclusion, these deployment patterns offer various ways to optimize the performance of real-time workloads on systems with multiple NUMA nodes. Each approach has its benefits and challenges, and the choice of deployment pattern depends on the nature of the real-time workload, the system architecture, and the performance goals. It is important to assess and experiment with different patterns to determine the best fit for a specific use case.

3.7 NUMA-Aware Data Structures

Data structures play a crucial role in the performance of real-time applications. Traditional data structures do not consider the NUMA topology, which can lead to suboptimal performance due to remote memory accesses and contention. NUMA-aware data structures are designed to optimize data placement and access patterns based on the NUMA topology. These data structures allocate memory for each NUMA node and manage data placement explicitly to minimize remote memory accesses and improve data locality.

For example, a NUMA-aware queue may have separate queues for each NUMA node, and tasks running on a node will enqueue and dequeue from the queue associated with that node. This minimizes contention and remote memory accesses, improving overall performance. However, this approach may introduce challenges related to data sharing and communication between tasks running on different NUMA nodes.

Implementing NUMA-aware data structures requires a deep understanding of the hardware architecture and the characteristics of the workload. It may also involve fine-tuning at both the application and system levels to achieve the desired performance outcomes.

3.8 Thread and Core Affinity

Thread and core affinity is a strategy to bind specific threads to specific cores or sets of cores. This is particularly important in a NUMA architecture, as it ensures that threads access local memory as much as possible, minimizing the overhead of inter-node communication. Setting thread and core affinity involves configuring the operating system scheduler to assign specific threads to specific cores.

For example, a real-time application may have multiple threads, each responsible for a different task. By assigning each thread to a specific core on a specific NUMA node, it is possible to optimize data locality and minimize contention. However, this approach may introduce challenges related to resource fragmentation, where some cores are underutilized while others are overburdened.

3.9 Dynamic Resource Management

Dynamic resource management involves continuously monitoring the system and workload and adjusting the deployment pattern, thread and core affinity, and data placement based on changing conditions. This is particularly important for real-time workloads, which may exhibit dynamic behavior and require timely and predictable performance.

For example, a real-time application may experience bursts of activity that temporarily increase the demand for CPU and memory resources. By dynamically adjusting the deployment pattern, thread and core affinity, and data placement, it is possible to accommodate these changes and maintain optimal performance. However, this approach may introduce challenges related to the complexity of the dynamic resource management algorithms and the overhead of continuously monitoring and adjusting the system.

3.10 NUMA-Aware Scheduling Policies

NUMA-aware scheduling policies involve customizing the operating system scheduler to consider the NUMA topology when making scheduling decisions. This includes prioritizing real-time tasks, assigning tasks to the most appropriate NUMA nodes, and minimizing cross-node communication.

For example, a NUMA-aware scheduler may prioritize real-time tasks and assign them to NUMA nodes with the most available resources. It may also consider the communication patterns between tasks and assign tasks that frequently communicate with each other to the same NUMA node. This minimizes the overhead of inter-node communication and improves overall performance. However, this approach may introduce challenges related to the complexity of the scheduling algorithms and the potential for resource fragmentation.

In conclusion, these strategies offer various ways to optimize the performance of real-time workloads on systems with multiple NUMA nodes. Each approach has its benefits and challenges, and the choice of strategy depends on the nature of the real-time workload, the system architecture, and the performance goals. It is important to assess and experiment with different strategies to determine the best fit for a specific use case.

4. Use Cases

The efficient management of real-time workloads on systems with multiple NUMA nodes is crucial for various applications that require timely and predictable performance. This section will present a tabular representation of different deployment patterns and their use cases for real-time workloads and provide a detailed discussion of specific use cases.

4.1 Tabular Representation of Deployment Patterns and Use Cases

Deployment Pattern	**Use Case and Description**

ISSN Online : 2709-5088
ISSN Print : 2709-507X

Single NUMA Node	Small real-time workloads that can fit entirely within the resources of a single NUMA node.
NUMA Node Partitioning	Large real-time workloads divided into smaller partitions to improve data locality and reduce contention.
Memory Binding and CPU Pinning	Critical real-time tasks that require minimal memory latency and are isolated on specific cores and memory.
NUMA-Aware Load Balancing	Workloads with varying task demands distributed across NUMA nodes to optimize resource utilization.
Hybrid NUMA Awareness	Combining real-time and non-real-time tasks on separate NUMA nodes to avoid interference and contention.
Data Locality Optimization	Workloads with specific memory access patterns that benefit from placing data and cores on the same node.
NUMA-Aware Data Structures	Applications with custom data structures explicitly designed to minimize remote memory accesses.
Thread and Core Affinity	Real-time tasks that need to remain on specific cores to avoid thread migration and reduce contention.
Dynamic Resource Management	Fluctuating workloads that require adaptive resource allocation and load balancing for real-time tasks.
NUMA-Aware Scheduling Policies	Real-time workloads with strict timing requirements, using custom scheduling strategies based on NUMA.

4.2 Detailed Discussion of Specific Use Cases

Financial Trading Systems: Financial trading systems require timely and predictable performance to execute trades quickly and accurately. These systems often involve complex algorithms and high-frequency trading, which demand consistent low latency and high throughput. NUMA-aware resource isolation, such as memory binding and CPU pinning, can help optimize the performance of financial trading systems by minimizing memory latency and ensuring predictable execution times for critical tasks.

Online Gaming Servers: Online gaming servers host multiplayer online games that require real-time interaction between players. These servers must handle a large number of concurrent connections and provide a smooth and responsive gaming experience. NUMA-aware load balancing and dynamic resource management can help optimize the performance of online gaming servers by distributing tasks across NUMA nodes based on the availability of resources and minimizing cross-node communication.

Telecommunications and Networking: Telecommunications and networking applications often involve real-time processing of data packets and require consistent low latency and high throughput. NUMA-aware data structures and data locality optimization can help optimize the performance of telecommunications and networking applications by minimizing remote memory accesses and improving data locality.

Industrial Automation and Robotics: Industrial automation and robotics applications often involve real-time control of machines and robots. These applications require timely and predictable performance to ensure the safety and efficiency of the operations. NUMA-aware scheduling policies and thread and core affinity can help optimize the performance of industrial automation and robotics applications by prioritizing real-time tasks and assigning them to the most appropriate NUMA nodes.

Aerospace and Defense Applications: Aerospace and defense applications often involve real-time processing of sensor data and control of aircraft and defense systems. These applications require consistent low latency and high throughput to ensure the safety and effectiveness of the operations. NUMA-aware resource isolation and deployment patterns, such as NUMA node partitioning and hybrid NUMA awareness, can help optimize the performance of aerospace and defense applications by minimizing memory latency and ensuring predictable execution times for critical tasks.

In conclusion, these use cases illustrate the importance of NUMA-aware resource isolation and deployment patterns in optimizing the performance of real-time workloads. Each use case has its unique requirements and challenges, and the choice of deployment pattern and strategy depends on the nature of the workload, the system architecture, and the performance goals. It is important to assess and experiment with different deployment patterns and strategies to determine the best fit for a specific use case.

5. Caveats and Challenges

Implementing NUMA-aware strategies for real-time workloads involves several caveats and challenges that need to be carefully considered to achieve the desired performance outcomes.

5.1 Complexity

Implementing NUMA-aware strategies adds complexity to the system configuration, deployment, and management. It involves a deep understanding of the hardware architecture, the characteristics of the workload, and the performance goals. It may also involve fine-tuning at both the application and system levels, customizing scheduling policies, implementing NUMA-aware data structures, and dynamically managing resources based on changing conditions. This complexity may make it challenging to implement and maintain NUMA-aware strategies, particularly for organizations with limited expertise and resources.

5.2 Resource Fragmentation

NUMA-aware strategies, such as thread and core affinity and NUMA node partitioning, may lead to resource fragmentation, where some nodes are underutilized while others are overburdened. This may result in suboptimal performance and inefficient resource utilization. It is important to carefully plan the deployment pattern and continuously monitor and adjust the system to minimize resource fragmentation and optimize resource utilization.

5.3 Dynamic Workload Changes

Real-time workloads may exhibit dynamic behavior, and static deployment patterns may become suboptimal over time. For example, a real-time application may experience bursts of activity that temporarily increase the demand for CPU and memory resources. Dynamic resource management algorithms are needed to continuously monitor and adjust the system based on changing workload demands and system conditions. However, this introduces additional complexity and overhead,

ISSN Online : 2709-5088
ISSN Print : 2709-507X

and it may be challenging to implement dynamic resource management algorithms that are both efficient and effective.

5.4 Interference with Other Workloads

NUMA-aware deployment may inadvertently affect the performance of non-real-time tasks sharing the same hardware. For example, prioritizing real-time tasks and assigning them to specific NUMA nodes may lead to resource contention and suboptimal performance for non-real-time tasks. It is important to carefully plan the deployment pattern and consider the impact on non-real-time tasks to strike a balance between optimizing real-time tasks and minimizing the impact on non-real-time tasks.

5.5 Increased Maintenance Overhead

Implementing NUMA-aware strategies may increase the maintenance overhead. It may involve customizing the operating system scheduler, implementing NUMA-aware data structures, and continuously monitoring and adjusting the system based on changing conditions. This may increase the maintenance burden and require ongoing attention and expertise to ensure optimal performance.

5.6 Limited System Understanding

Implementing NUMA-aware strategies requires a deep understanding of the hardware architecture, the characteristics of the workload, and the performance goals. However, many organizations may have limited expertise and understanding of these aspects, making it challenging to implement and maintain NUMA-aware strategies effectively.

In conclusion, implementing NUMA-aware strategies for real-time workloads involves several caveats and challenges that need to be carefully considered. It is important to assess the complexity, resource fragmentation, dynamic workload changes, interference with other workloads, increased maintenance overhead, and limited system understanding to determine the best approach for a specific use case. It is also important to continuously monitor and adjust the system based on changing conditions to ensure optimal performance.

5.7 Performance Variability

Performance variability is a significant concern in real-time systems. The performance of real-time tasks must be consistent and predictable to meet their timing requirements. However, implementing NUMA-aware strategies may introduce performance variability due to several factors. For example, dynamic resource management algorithms may continuously adjust the system based on changing workload demands and system conditions, leading to variability in task execution times. Similarly, NUMA-aware load balancing may distribute tasks across NUMA nodes based on the availability of resources, leading to variability in task execution times. It is important to carefully design and test the system to minimize performance variability and ensure that real-time tasks meet their timing requirements.

5.8 Potential for Deadlocks

Implementing NUMA-aware strategies may introduce the potential for deadlocks. For example, NUMA-aware data structures may involve complex synchronization mechanisms to ensure data consistency across NUMA nodes. Similarly, dynamic resource management algorithms may involve complex decision-making processes to allocate and deallocate resources based on changing conditions. These complexities may introduce the potential for deadlocks, where tasks are unable to progress because each task is waiting for another task to release a resource. It is important to carefully design and test the system to minimize the potential for deadlocks and ensure that tasks can progress smoothly.

5.9 Limited Benefit for Small Workloads

NUMA-aware strategies may provide limited benefits for small workloads that do not fully utilize the available resources. For example, a small workload may not generate enough tasks to fully utilize the available CPU cores and memory, making it unnecessary to implement complex NUMA-aware strategies. Similarly, a small workload may not generate enough memory accesses to justify the overhead of implementing NUMA-aware data structures. It is important to carefully assess the characteristics of the workload and the performance goals to determine whether implementing NUMA-aware strategies is necessary and beneficial.

5.10 Platform-Specific Considerations

NUMA architectures and configurations may vary significantly across different hardware platforms. For example, the number of NUMA nodes, the number of CPU cores per node, the amount of memory per node, and the interconnection topology may vary across different platforms. These platform-specific considerations may affect the performance of NUMA-aware strategies and may require platform-specific optimizations. For example, a NUMA-aware data structure optimized for a platform with a small number of NUMA nodes and a large amount of memory per node may not perform well on a platform with a large number of NUMA nodes and a small amount of memory per node. It is important to carefully assess the platform-specific considerations and customize the NUMA-aware strategies accordingly.

In conclusion, implementing NUMA-aware strategies for real-time workloads involves several caveats and challenges that need to be carefully considered. Performance variability, the potential for deadlocks, limited benefits for small workloads, and platform-specific considerations are significant challenges that may affect the performance and effectiveness of NUMA-aware strategies. It is important to carefully design, test, and customize the system to address these challenges and ensure optimal performance for real-time workloads.

6. Conclusion

The management of real-time workloads on systems with multiple NUMA nodes is a critical aspect of ensuring applications run efficiently and without interference from other processes. This paper has discussed various strategies for NUMA-aware resource isolation and deployment patterns, their use cases, and the associated caveats and challenges.

6.1 Summary of Key Findings

Importance of NUMA-Aware Strategies: Implementing NUMA-aware strategies is crucial for optimizing the performance of real-time workloads on systems with multiple NUMA nodes. These strategies involve strategically assigning tasks to specific CPU cores and memory regions, implementing NUMA-aware data structures, customizing scheduling policies, and dynamically managing resources based on changing conditions.

Variety of Deployment Patterns: There are several deployment patterns that can be employed to achieve effective NUMA-aware resource isolation for real-time tasks, including single NUMA node deployment, NUMA node partitioning, memory binding and CPU pinning, NUMA-aware load balancing, hybrid NUMA awareness, and data locality optimization.

Caveats and Challenges: Implementing NUMA-aware strategies involves several caveats and challenges, including increased complexity, resource fragmentation, dynamic workload changes, interference with other workloads, increased maintenance overhead, limited system understanding, performance variability, potential for deadlocks, limited benefits for small workloads, and platform-specific considerations.

6.2 Recommendations for Implementation

Assess Workload Characteristics: It is important to carefully assess the characteristics of the workload and the performance goals to determine the most appropriate NUMA-aware strategies and deployment patterns.

Consider Platform-Specific Considerations: NUMA architectures and configurations may vary significantly across different hardware platforms, and it is important to customize the NUMA- aware strategies accordingly.

Minimize Performance Variability: It is important to carefully design and test the system to minimize performance variability and ensure that real-time tasks meet their timing requirements.

Address Resource Fragmentation: It is important to carefully plan the deployment pattern and continuously monitor and adjust the system to minimize resource fragmentation and optimize resource utilization.

Consider Maintenance Overhead: Implementing NUMA-aware strategies may increase the maintenance overhead, and it is important to consider the available expertise and resources.

6.3 Future Directions

Automated Resource Management: Developing automated resource management algorithms that can continuously monitor and adjust the system based on changing workload demands and system conditions can help address the challenges associated with dynamic workload changes and resource fragmentation.

Advanced Scheduling Algorithms: Developing advanced scheduling algorithms that consider the NUMA topology, the characteristics of the workload, and the performance goals can help address the challenges associated with performance variability and interference with other workloads.

Platform-Specific Optimizations: Developing platform-specific optimizations that consider the unique characteristics of different hardware platforms can help address the challenges associated with platform-specific considerations.

In conclusion, implementing NUMA-aware strategies for real-time workloads is crucial for optimizing performance on systems with multiple NUMA nodes. It involves several caveats and challenges that need to be carefully considered and addressed. Future directions include developing automated resource management algorithms, advanced scheduling algorithms, and platform-specific optimizations to further enhance the performance of real-time workloads on systems with multiple NUMA nodes.

ACRONYMS:

NUMA - Non-Uniform Memory Access
CPU - Central Processing Unit
OS - Operating System
RAM - Random Access Memory
HPC - High Performance Computing
SMP - Symmetric Multiprocessing
UMA - Uniform Memory Access
API - Application Programming Interface
VM - Virtual Machine
QoS - Quality of Service
I/O - Input/Output
GPU - Graphics Processing Unit
FPGA - Field-Programmable Gate Array
IPC - Inter-Process Communication
ISR - Interrupt Service Routine
RTOS - Real-Time Operating System
SMT - Simultaneous Multithreading
HT - Hyper-Threading

DMA - Direct Memory Access
MMU - Memory Management Unit
TLB - Translation Lookaside Buffer
LRU - Least Recently Used
FIFO - First In, First Out
LIFO - Last In, First Out
SSD - Solid State Drive
HDD - Hard Disk Drive
RAID - Redundant Array of Independent Disks
NAS - Network Attached Storage
SAN - Storage Area Network
NIC - Network Interface Card
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
IP - Internet Protocol
HTTP - HyperText Transfer Protocol
HTTPS - HyperText Transfer Protocol Secure
FTP - File Transfer Protocol
SNMP - Simple Network Management Protocol
QEMU - Quick Emulator
KVM - Kernel-based Virtual Machine
VMX - Virtual Machine Extensions
SVM - Secure Virtual Machine

REFERENCES:

[1] Tim Kiefer, Thomas Kissinger, Benjamin Schlegel, Dirk Habich (2014). A NUMA-aware in-memory storage engine for tera-scale multiprocessor systems. Researchgate.net.

[2] Jianmin Qian, Jian Li, Ruhui Ma, Haibing Guan (2018). Optimizing Virtual Resource Management for Consolidated NUMA Systems. IEEE 36th International Conference on Computer Design (ICCD).

[3] Richard Wu; Xiao Zhang; Xiangling Kong; Yangyi Chen; Rohit Jnagal; Robert Hagm (2019). Evaluation of NUMA-Aware Scheduling in Warehouse-Scale Clusters. IEEE 12th International Conference on Cloud Computing (CLOUD).

[4] Mulya Agung, Muhammad Alfian Amrizal, Ryusuke Egawa, Hiroyuki Takizawa (2019). The Impacts of Locality and Memory Congestion-aware Thread Mapping on Energy Consumption of Modern NUMA Systems. IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS).

[5] Puya Memarzia, Suprio Ray, Virendra C. Bhavsar (2020). The Art of Efficient In-memory Query Processing on NUMA Systems: a Systematic Approach. IEEE Xplore IEEE 36th International Conference on Data Engineering (ICDE).

[6] Mughal, A. A. (2019). Cybersecurity Hygiene in the Era of Internet of Things (IoT): Best Practices and Challenges. *Applied Research in Artificial Intelligence and Cloud Computing*, *2*(1), 1-31.

[7] Mughal, A. A. (2020). Cyber Attacks on OSI Layers: Understanding the Threat Landscape. *Journal of Humanities and Applied Science Research*, *3*(1), 1-18.

[8] Mughal, A. A. (2019). A COMPREHENSIVE STUDY OF PRACTICAL TECHNIQUES AND METHODOLOGIES IN INCIDENT-BASED APPROACHES FOR CYBER

ISSN Online : 2709-5088
ISSN Print : 2709-507X

FORENSICS. *Tensorgate Journal of Sustainable Technology and Infrastructure for Developing Countries*, *2*(1), 1-18.

[9] Mughal, A. A. (2018). The Art of Cybersecurity: Defense in Depth Strategy for Robust Protection. *International Journal of Intelligent Automation and Computing*, *1*(1), 1-20.

[10] Mughal, A. A. (2018). Artificial Intelligence in Information Security: Exploring the Advantages, Challenges, and Future Directions. *Journal of Artificial Intelligence and Machine Learning in Management*, *2*(1), 22-34.