International Journal of Computer Applications (0975 – 8887)
Volume 186 – No.62, January 2025

A Comprehensive Study of OS Strategies and New Scheduling Techniques

Md Rakeen Islam Nahin
Military Institute of Science and Technology
Mirpur Cantonment
Dhaka, Bangladesh

Sheikh Raiyan Ahmed
Military Institute of Science and Technology
Mirpur Cantonment
Dhaka, Bangladesh

ABSTRACT

In the modern computers the scheduling algorithms have bought a significant advancement by performing tasks within a precise time frame. With the advancement of real time, dynamic scheduling algorithms and affinity based partitioning algorithms in multicore processors there lies a need for effective operating system solutions in multicore environments so that maximum CPU utilization can be made. Over the years real time scheduling algorithms, one of the solutions for multicore processors in real time environment have been proposed so that user expectation can be fulfilled effectively by following the scheduling protocols. However, none of the scheduling algorithm is universal. There comes a need for modification of the algorithms and implementation of hybrid models so that the scheduling is done more efficiently. The main objective of the research work is to have a clear view of the scheduling algorithms, its importance, discussing about its limitations and proposing theories for improving the algorithms. Among real-time scheduling algorithms discussion about Rate Monotonous Scheduling Algorithm (RMA), Earliest Deadline First Algorithm (EDF) and among dynamic scheduling algorithm discussion about Work-Stealing Algorithm and among task partitioning in heterogenous systems discussion about Affinity based partitioning Algorithms are made. We first discussed why maintaining the scheduling and Core-Affinity are the most vital among all other approaches for increasing efficiency in real time environments. Later on, we discussed about the scheduling algorithms like static partitioning algorithms (EDF, RMA) and proposed a hybrid schedule using multiple cores, dynamic partitioning algorithm like work-stealing, modification of the algorithm using clock pulse and affinity-based partitioning algorithm along with its modification using cross-checking algorithms to make it more reliable. Furthermore, we discussed about the importance of core affinity in multicore systems. Overall, this research will give a clear concept of "Theoretical Approach for Operating System Solutions for Multicore Processors" by discussing about the scheduling algorithms and core affinity; and proposing theories and algorithms for increasing the efficiency and reliability of the task-partitioning algorithms.

Keywords

Task-partitioning;Core Affinity;Rate Monotonous Algorithm; (RMA); Earliest Deadline First (EDF); Work-Stealing; Real Time Scheduling;

1. INTRODUCTION

Modern operating systems (OSs) handle requirements along with leveraging the parallel processing capabilities of multi-core by scheduling algorithms, resource management techniques and hardware optimization.

Modern scheduling algorithms like **Rate Monotonic Scheduling (RMS)** or **Dynamic Scheduling** like **Earliest Deadline First (EDF) have proven** to be effective for enhancing the performance of multi-core processors. For multi-core utilization real-time scheduling, partitioned scheduling (Each core has its own task queue, and tasks are statically assigned to specific cores. This minimizes inter-core communication and scheduling overhead), global scheduling (A single task queue is shared among all cores, allowing dynamic task allocation) and hybrid scheduling are followed. The other approaches include Synchronization, Inter-Core Communication and predictable memory management.

However, improvements need to be made so that the performance is consistent in all kinds of situation, though the schedules are having a great impact there are instances when

a particular schedule face challenges which makes hybrid model and proposed algorithms necessary**.**

The following are the reasons which made scheduling algorithms and core affinity the most vital for increasing the efficiency in real-time environments.

a. Predictable Scheduling: Assign specific tasks to particular cores eliminates uncertainties and makes the behavior of the OS more predictable.

b. Isolation: Ensures high priority tasks to run fast and not letting tasks with limited deadline and small time period wait for very long amount of time.

c. Scalability: For multi-core processors, scheduling is done to ensure efficient parallel utilization.

On the other hand, the following reasons are applicable for other approaches to be comparatively a bit insignificant when it comes to real-time systems.

a. Synchronization**:** Effective use of scheduling algorithms ensures synchronization in most of the time.

b. Memory Management: Predictable behavior of OS reduces the load of memory management.

c. Task partitioning: Effective task partitioning relies on Real-time scheduling for meeting time constraints and core affinity to enhance execution efficiently.

d. Global Scheduling algorithms: Depends on real-time scheduling greatly to decide when tasks execute while distributing them dynamically.

2. SCHEDULING ALGORITHMS WITH PROPOSED MODIFICATION

A brief study of various real-time scheduling algorithms and dynamic scheduling algorithms which are frequently used in handling the multicore-processors will be discussed.

2.1 Real-Time Scheduling Algorithms:

2.1.1 Earliest Deadline First (EDF):

The Earliest Deadline First Algorithm, the process needs to be scheduled in a manner so that the earliest deadline can get the highest priority.

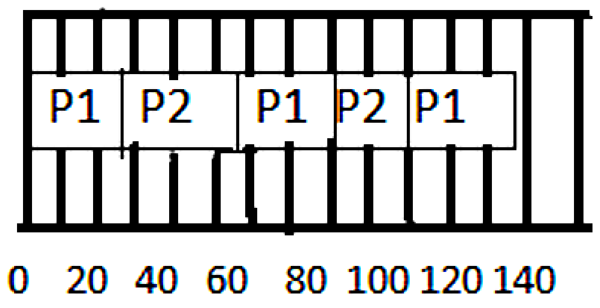

Fig.1: Earliest Deadline First Scheduling Algorithm.

Let the period of P1 be p1 = 50
Let the processing time of P1 be t1 = 25
Let the period of P2 be period2 = 75
Let the processing time of P2 be t2 = 30

1. Deadline of P1 is earlier, so priority of P1>P2.
2. Initially P1 runs and completes its execution of 25 time.
3. After 25 times, P2 starts to execute until 50 times, when P1 is able to execute.
4. Now, comparing the deadline of (P1, P2) = (100, 75), P2 continues to execute.
5. P2 completes its processing at time 55.
6. P1 starts to execute until time 75, when P2 is able to execute.
7. Now, again comparing the deadline of (P1, P2) = (100, 150), P1 continues to execute.
8. Repeat the above steps…
9. Finally at time 150, both P1 and P2 have the same deadline, so P2 will continue to execute till its processing time after which P1 starts to execute.

In the example P1 and P2 the process having earlier deadline is said to schedule first till the time period it is assigned with has been completed.

Earliest dead line first scheduling is very much efficient when the tasks are having dynamic deadline this scheduling algorithm can make the CPU utilization up to almost 100%.

Although, the tasks which are periodic and are doesn't need to address dynamic deadline Earliest deadline first is an unnecessary burden for the CPU. The tasks like periodic monitoring of blood pressure, updating machine states, audio processing software where sound needs to be adjusted periodically Earliest deadline first becomes an overhead because the dynamic deadline scheduling is not required. In those case we use RMA.

2.1.2 Rate Monotonic Scheduling Algorithm (RMA): In this scheduling the process with shortest period will get the highest priority.

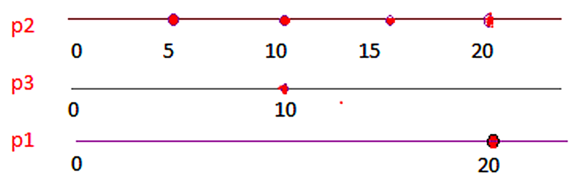

Fig.2: Rate monotonous Algorithm Scheduling.

The above figure says that,

1. Process P2 will execute two times for every 5 time units
2. Process P3 will execute two times for every 10 time units
3. Process P1 will execute three times in 20 time units. This has to be kept in mind for understanding the entire execution of the algorithm below.

i. Process P2 will run first for 2 time units because it has the highest priority.

ii. After completing its two units, P3 will get the chance and thus it will run for 2 time units.

iii. As we know that process P2 will run 2 times in the interval of 5 time units and process P3 will run 2 times in the interval of 10 time units, they have fulfilled the criteria.

iv. Now process P1 which has the least priority will get the chance and it will run for 1 time. And here the interval of 5 time units have completed.

v. Higher priority P2 will preempt P1 and thus will run 2 times.

vi. As P3 have completed its 2 time units for its interval of 10 time units, P1 will get chance and it will run for the remaining 2 times, completing its execution which was thrice in 20 time units.

vii. Now 9-10 interval remains idle as no process needs it.

viii. At 10 time units, process P2 will run for 2 times completing its criteria for the third interval (10-15).

ix. Process P3 will now run for two times completing its execution.

x. Interval 14-15 will again remain idle for the same reason mentioned above.

xi. At 15 time unit, process P2 will execute for two times completing its execution. This is how the rate monotonic scheduling works. In short, the processes are divided into periods and in each period the process assigned with the shortest period gets the priority to execute.

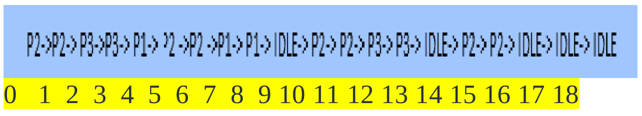
P2->P2->P3->P3->P1->P2->P2->P1->P1->IDLE->P2->P2->P3->P3->IDLE->P2->P2->IDLE->IDLE->IDLE
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Fig.3: Pictorial view of RMA Scheduling

Advantages**:**

- It is easy to implement.
- If any static priority assignment algorithm can meet the deadlines, then rate monotonic scheduling can also do the same. It is optimal.
- It consists of a calculated copy of the time periods, unlike other time-sharing algorithms as Round-robin which neglects the scheduling needs of the processes.

Disadvantages**:**

- It is very difficult to support aperiodic and sporadic tasks under RMA.
- RMA is not optimal when the task period and deadline differ.

Overall, RMA is effective in the schedules which does not need to address deadlines dynamically.

The graph represents a comparison between EDF and RMA.

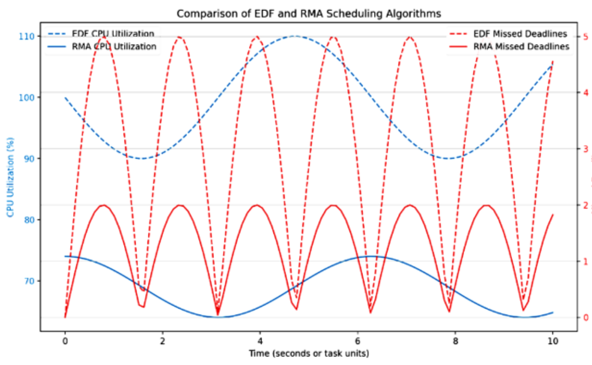

Fig.4: **Comparison of EDF and RMA scheduling**.

In the graph the CPU utilization of EDF is more and RMA is less. The more CPU utilization the higher probability of missing deadlines as more tasks are available in the CPU.

So, RMA is suitable for the periodic tasks with less dependencies on deadline and less CPU utilization and EDF is suitable where higher CPU utilization is required and higher deadlines need to be met. It is thus an overhead to apply EDF in the situations which RMA can handle.

2.1.3 Hybrid Scheduling Model (proposed model):

The model is divided into 3 parts:

2.1.3.1. Task Controller:

a. Acts as a **decision-making unit**.
b. Examines incoming tasks to determine their characteristics:
   i. **Tasks with a period but no explicit deadline** → Assign to **Core 1 (RMA)**.
   ii. **Tasks with a deadline** → Assign to **Core 2 (EDF)**.
   iii. **Any new task** (regardless of type) → Check the **deadline first** and assign to Core 2.

2. Core 1 (Follow Rate Monotonous Algorithm): Handles periodic tasks without explicit deadlines.

3. Core 2 (Follow Earliest Deadline First Algorithm): Handles tasks with deadlines, including:

i. Periodic tasks with a deadline.

ii. Aperiodic and sporadic tasks: Dynamically adjust priorities based on nearest deadline.

2.1.3.2 Result and Analysis:

Aperiodic Task Example:

- **Task Name:** T5
- **Arrival Time**: 100 ms
- **Execution Time**: 25 ms
- **Deadline:** 150 ms (only applicable to EDF, not RMA)

Now in this case the task controller will schedule it for core 2 in hybrid model thus EDF will execute it and

TAT=125−100=25ms, WTEDF=25−25=0ms

If this task was for RMA,

TAT will be 35ms and WT will be 10ms

Thus, hybrid model reduces the TAT and WT by selecting the correct scheduler.

On the other hand, **RMA** generally has lower runtime overhead compared to **EDF** because it does not need to continuously check and compare deadlines**. EDF**, on the other hand, requires constant recalculation of task deadlines to decide which task to schedule next.

Overall, the hybrid approach will allow acquire the advantages of both RMA and EDF.

According to “Real Time Systems in Hospital” by Velibor Bozic[3][v], North Shore University Hospital, New York successfully launched the application of real-time systems.

The domains include:

i. Patient monitoring systems (alarms & alerts for abnormal vitals) which can be categorized as EDF

ii. RFID-based asset tracking which can be categorized as EDF.

iii. Medical devices with periodic tasks (e.g., ventilators, infusion pumps) this can be categorized as RMA

iv. Real-time surgical equipment that requires deterministic execution this can be categorized as RMA.

Now analyzing the CPU utilization when the hospital is simultaneously performing EDF and RMA and the task-controller is present the CPU utilization will be more.

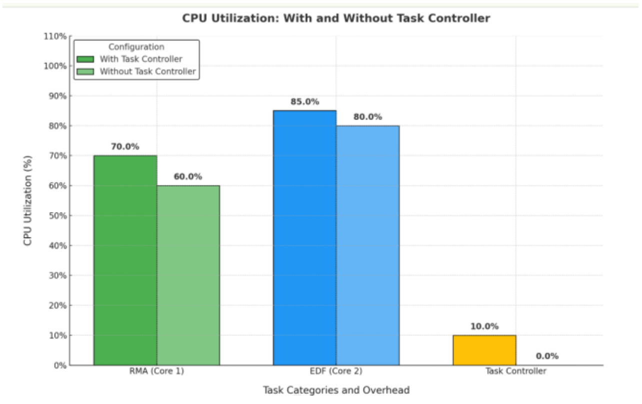

Fig.5: CPU utilization with and without task controller in North Shore University Hospital.

2.1.3.3 Challenges and Considerations:

1. If the tasks are not evenly distributed one core will be over burdened with tasks. So, this method should be utilized based on the situations if required more cores may be assigned for particular algorithms.
2. Latency will occur if tasks are not synchronized properly.
3. Task controller may also cause delay to decide which core to schedule.
4. Tasks on Core 1 and Core 2 may compete for shared resources or deadlines.

2.2 Dynamic Scheduling Algorithms:

2.2.1 Work-stealing: In a work-stealing scheme, each thread has its own pool of tasks. When a thread has finished a task, it acquires a new one from its own work-pool, and, when a new subtask is created, the new task is added to the same work-pool. If a thread discovers that it has no more tasks in its own work-pool, it can try to steal a task from the work-pool of another thread (source: Dynamic Load Balancing Using Work-Stealing, Daniel Cederman and Philippas Tsigas)[3][ii].

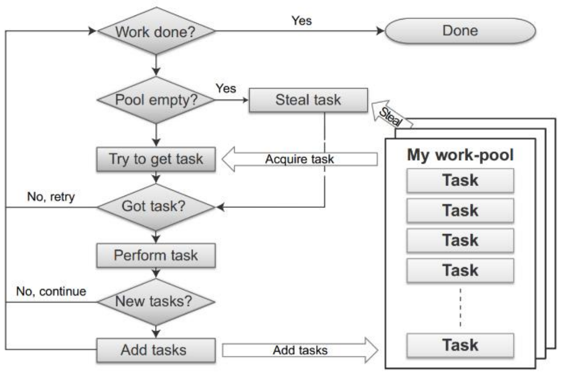

Fig.6: Task sharing using work-pools and work-stealing (source: Dynamic Load Balancing Using Work-Stealing, Daniel Cederman and Philippas Tsigas)[3][ii]

The advantages of this schedule are:

- Dynamic Load Balancing: Idle processors steal work from busy ones, ensuring even distribution of tasks.
- Increased Parallelism: It helps utilize available processors effectively, reducing idle time.
- Scalability: The algorithm can easily scale as the number of processors increases without significant overhead.
- Fault Tolerance: Tasks can be redistributed if processors fail or slow down, maintaining performance.
- Flexibility: It works well with both fine-grained (task divided into smaller units) and coarse-grained (tasks divided into larger units) tasks and can handle heterogeneous systems.

Performance comparison between work stealing and baseline scheduling (no work stealing).

CPU utilization/time(sec)

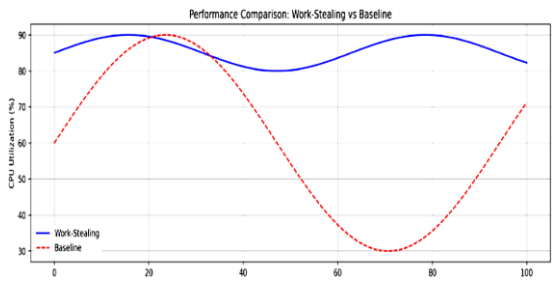

Fig.7: Performance Comparison between base scheduling and work stealing algorithm.

In the graph more CPU utilization is being done in work stealing algorithm than baseline scheduling.

Task stolen/time(sec)

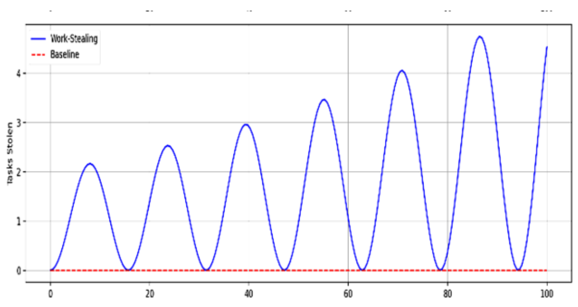

Fig.8: Performance Comparison between base scheduling and work stealing algorithm.

The task stolen varies by time in work stealing on the other hand it remains constant for baseline scheduling.

Response time (ms)/Time

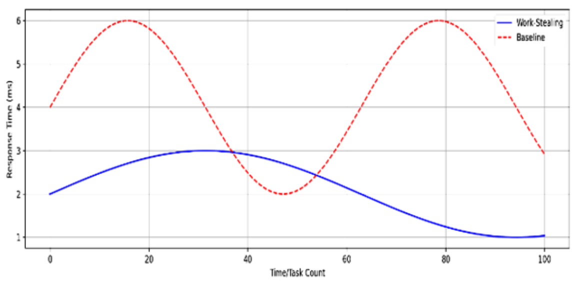

Fig.9: **Performance Comparison between base scheduling and work stealing algorithm**.

The Work stealing algorithm gives lower and more stable response time.

The disadvantages of this schedule are:

- Stealing Overhead: Task transfer and synchronization introduce overhead, potentially reducing efficiency.
- Non-Determinism: The random nature of task stealing can lead to unpredictable performance.
- Load Imbalance: In some cases, too many idle processors may cause inefficient task redistribution.
- Inefficiency for Small Tasks: Overhead can outweigh benefits when dealing with small, quick tasks.
- Increased Latency: Task transfer across processors or nodes adds latency, especially in distributed systems.

Overall performance analysis of work stealing algorithm:

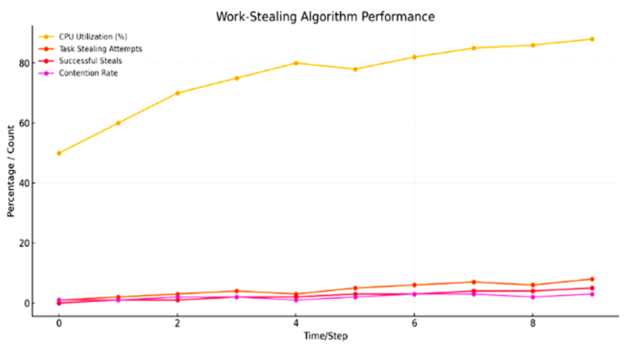

Fig.10: Work stealing algorithm performance

In the graph contention rate is the attempts of multiple threads competing for the same task. The lower the contention rate (purple line) the higher the CPU utilization (yellow line).

2.2.2 Work-stealing algorithm with clock pulse (proposed model):

1. A clock pulse can be introduced to govern when task stealing can occur. If a core has been idle for a specific number of clock pulses, only then can it steal tasks from others. This approach can reduce task stealing and can reduce the probability of multiple threads trying to steal task at the same time.
2. Categorize the tasks into pools based on their size (small, medium, and large tasks). Only similar sized tasks can be taken by a thread.
3. Furthermore, introduction of another clock pulse that dynamically adjusts the percentage of threads allocated to different categories based on recent activity is done. This allows the system to respond to dynamic changes in workload. If a category is particularly busy, it gets more resources, ensuring that no category becomes a bottleneck. Thus, by distributing threads based on activity, the system can prevent overloading one category while others are idle.
4. Combining the three models reduced task switching, predictable and balanced loading in Work-stealing algorithm will be achieved. Thus, it will enhance the performance of the algorithm significantly.

Expected reduce in contention rate by following the above algorithm.

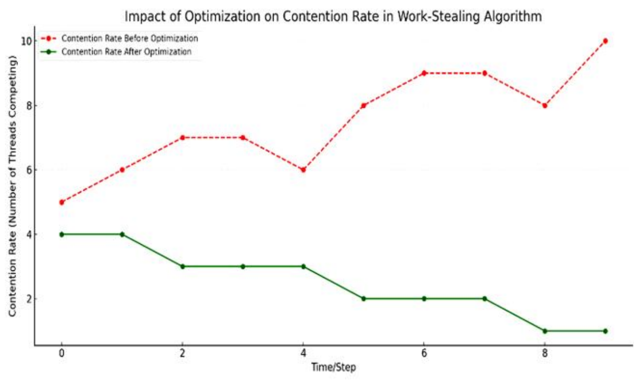

Fig.11: Impact of Optimization and Contention in Work-Stealing Algorithm.

2.2.2.1 Result and Analysis:

In the paper “Work-Stealing Algorithm Distilled” by Ryan Zheng[3][iii] **ForkJoinPool** uses the **work-stealing** algorithm to balance the workload on different threads.

Now analyzing the CPU utilization incase, of multiple work-stealing where there is a chance of contention between the tasks (i.e. multiple workers trying to steal the same task).

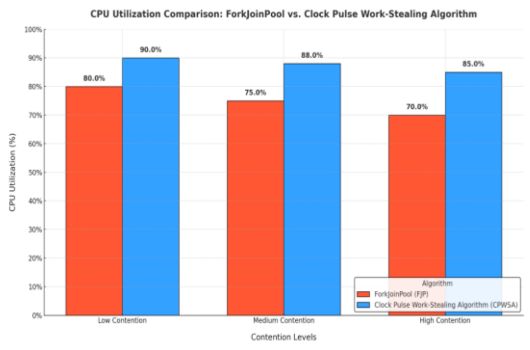

Fig.12: Comparison between ForkJoinPool and clock pulse work-stealing algorithm.

2.2.2.2 Challenges and considerations:

The algorithm is complex and waiting for a specific clock-pulse may increase latency. A condition can be applied for the application of clock pulse; if the switching rate is greater than a certain amount only then clock pulse model will be implemented. This will reduce the latency.

2.3 Task Partitioning in Heterogeneous Systems:

2.3.1 Affinity based partitioning:

Technique used in computer science and distributed computing to divide a set of tasks in such a way so that the relationships between them is preserved. The goal is to keep related entities together in the same partition to reduce communication overhead and increase efficiency.

The advantages of affinity-based partitioning are:

Technique used in computer science and distributed computing to divide a set of tasks in such a way so that the relationships between them is preserved. The goal is to keep related entities together in the same partition to reduce communication overhead and increase efficiency.

The advantages of affinity-based partitioning are:

- Reduced Communication Overhead: Minimizes data transfer between partitions.
- Improved Performance: Optimizes processing by grouping related data or tasks together.
- Better Load Balancing: Ensures even distribution of workloads across partitions.
- Optimized Resource Utilization: Efficient use of system resources like CPU and memory.
- Faster Data Access: Improves query and data retrieval times.
- Scalability: Facilitates smooth horizontal scaling in distributed systems.
- Minimized Latency: Reduces system response times.
- Improved Fault Tolerance: Limits the impact of failures to specific partitions.

The disadvantages of affinity-based partitioning are:

- Increased Complexity: Difficult to determine and maintain affinity relationships.
- Skewed Load Distribution: Risk of imbalanced partitions if affinities are uneven.
- High Computational Cost: Partitioning large datasets based on affinity can be resource-intensive.
- Reduced Flexibility: Tight coupling of related data/tasks can make rebalancing or scaling harder.
- Potential Data Duplication: Replicating related data across partitions might increase storage needs.
- Dependency on Accurate Affinity Data: Inaccurate or outdated affinity information can lead to inefficiencies.
- Limited Generality: May not work well for systems with weak or unpredictable relationships.
- Maintenance Overhead: Frequent updates to partitions may be required as relationships change.

2.3.2 Affinity based partitioning with sub-partitions (proposed model)

Basing on the advantages and disadvantages it can be concluded that affinity-based partitioning helps to manage the tasks more efficiently and it is more reliable as the related tasks are adjusted to run in the same partition. However, cost issue is there and maintaining related tasks may sometimes provide overhead like a task to decide whether to go in partition 1 or partition 2 and the conditions for the partition are runtime and deadline; the task may have runtime eligible for partition 1 whereas deadline eligible for partition 2 in that case CPU utilization may not be done properly and deadline may also be avoided. So, cross checking the task is done multiple times before executing it because if the task is in the correct place for selecting the right partition some time may be compromised but the execution time will be reduced significantly.

for example,

```
1. if (A has runtime 15 or more)
2. partition 1
3. if (A has deadline <1.5)
4. enter subpartition1
5. else if(deadline<3)
6. enter subpartition2
7. else go to
8. partition2
9. if (A has runtime less than 15)
10. partition2
11. start
12. if(deadline<3)
13. go to partition1
14. else if(deadline>4 and <10)
15. enter subpartition1
16. else
17. enter subpartition2
```

Partition1 has higher priority than partition2 same goes for subpartition1 and subpartition2.

This algorithm can improve the performance of affinity- based partitioning particularly for workloads, where optimizing execution time significantly outweighs preprocessing costs. Fine-tuning is needed for edge cases and scalability.

This algorithm further provides improved placement accuracy, enhanced priority management, reduced execution time and dynamic adaptability by dynamically adjusting partition selection.

Expected performance if the algorithm is implemented:

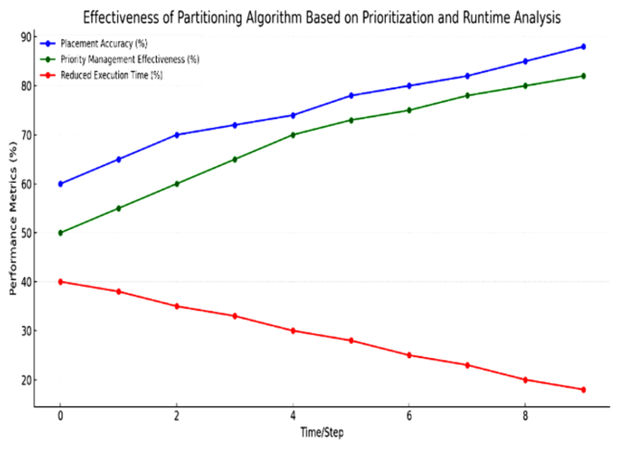

Fig.13: Effectiveness of Partitioning algorithm based on prioritization and runtime analysis.

In the graph it represents that the more tasks are partitioned into sub-partition the more placement accuracy and priority management will take place reducing the execution time.

2.3.2.1 Result and Analysis:

According to "Affinity-Based Task Scheduling on Heterogeneous Multicore Systems Using CBS and QBICTM" by Shoaib Iftikhar Abbasi the task are partitioned based on the following principles:

Chunk-Based Scheduler (CBS): Tasks are allocated based on core processing speeds, ensuring fair load distribution.

Quantum-Based Intra-Core Task Migration (QBICTM): Tasks are divided into equal chunks and allowed time on the fastest core before migrating.

Now comparing these two models with the proposed model in Aerospace Industry in case of CPU utilization, the following outcome is achieved.

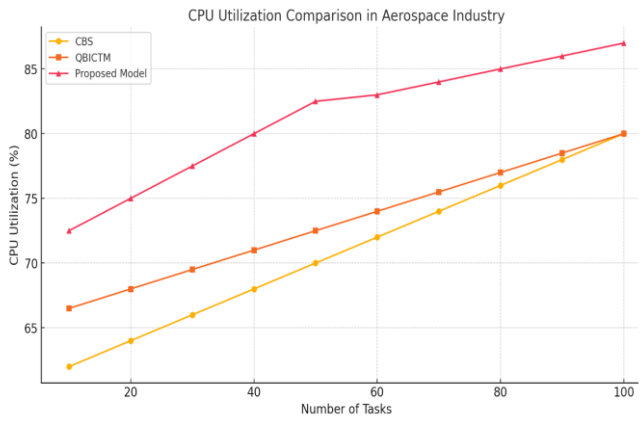

Fig.14: CPU utilization comparison in Aerospace industry.

The proposed model is costlier but gives the best performance in high-load industries like aerospace industries.

The proposed model causes a drawback in response time when the tasks exceed a certain level.

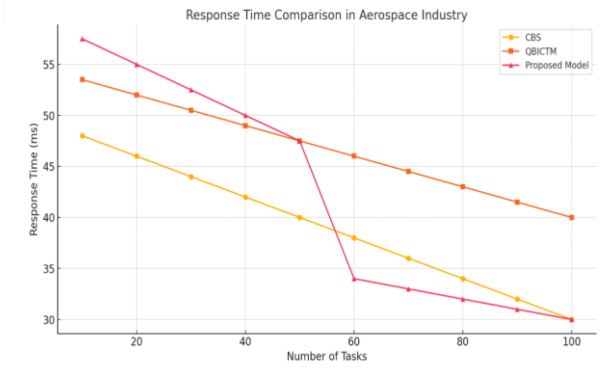

Fig.15: Response time comparison in Aerospace industry.

2.3.2.2 Challenges and considerations:

The main challenge of this algorithm is that it increases the time complexity and cost for having multiple partitions and cross checking.

A small fraction of time can be compromised for having a reliable execution because for saving time complexity if partitioning is not reliable the execution time will be increased.

The foundational study by Liu and Layland in 1973 demonstrated that scheduling overhead in real-time systems is typically minimal compared to the task execution times.[3][i]

If the decision making logic operates in time-complexity of O(1) or O(log N). This is negligible compared to O(N^2) (task execution time-complexity).

3. IMPORTANCE OF CORE AFFINITY IN MULTICORE SYSTEMS

CPU affinity enables binding a process or multiple processes to a specific CPU core in a way that the process(es) will run from that specific core only. When trying to perform performance testing on a host with many cores, it is wise to run multiple instances of a process, each one on different core. This enables higher CPU utilization.

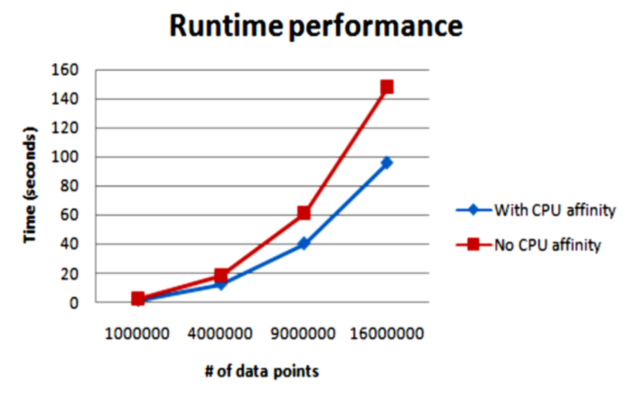

Fig.1: Run time performance CPU affinity vs no CPU affinity

Source:lanvu.wordpress.com[5][2]

When tasks that share similar resources or communicate frequently, they are placed on the same core, cache locality is enhanced which leads to faster data retrieval and reduced latency. Additionally, core affinity helps avoid contention for resources, such as CPU time or memory bandwidth, which can occur when tasks are unnecessarily spread across different cores. By maintaining tasks on the same core, the overhead of transferring data between cores can be minimized, improving overall system efficiency. In real-time systems, core affinity is particularly critical, as it helps meet stringent timing constraints by ensuring that tasks have predictable execution patterns and reduce the risk of missing deadlines. This concept is widely explored in works like those of Liu and Layland[3][i], as well as in more recent studies on scheduling algorithms and multicore processor optimization, where core affinity is often a key factor for performance improvement and resource management in complex systems.

4. DISCUSSION

Combining effective scheduling algorithms and core affinity is thus the most significant factor for enhancement in multicore systems.

The three types of scheduling algorithms are mainly discussed static, dynamic and task partitioning for heterogenous systems and none of the category is universal. In case of static scheduling EDF category will not perform the way RMA will perform in certain situation so a hybrid model has been proposed so that irrespective of the case the CPU can perform at its peak and maximum utilization can be taken place. In the field of dynamic algorithm discussion has been made about work-stealing algorithm and implementation of clock-pulse over it to control its stealing and resist its behavior from getting unpredictable. Lastly, discussions were made about affinity-based partitioning and how implementation of sub-partitioning increased the reliability of the partitions by implementation of multiple cross checking and dividing the partitions into sub-partitions.

Core affinity is also a vital part in the multicore systems the importance of core affinity has been discussed.

By using the proper scheduling algorithm in proper place and implementing core affinity where necessary should be the prime objective for bringing out a successful process execution.

5. CONCLUSION

Scheduling algorithms and core affinity stands as a formidable pillar for enhancing the performance of OS. Scheduling algorithms like static, dynamic, or task partitioning for heterogeneous systems, plays a significant role in managing computational resources. By meticulously selecting algorithms like Rate Monotonic, Earliest Deadline First, work stealing, or affinity-based partitioning and adapting them to the specific requirements of the system it is possible to maximize CPU utilization while ensuring timely task execution. However, no single algorithm provides a universal solution, that is why hybrid models and algorithms are proposed for making the scheduling algorithm more predictable and assigning the tasks effectively so that CPU can reach to its maximum utilization.

Furthermore, core affinity is essential for minimizing resource contention and ensures that tasks with similar characteristics are localized to specific cores. This practice increases the Run time performance of CPU significantly over time.

Lastly, the combination of the scheduling algorithms by implementing hybrid models involving multiple cores and modifying the algorithms by implementing clock pulse and sub-partitioning the algorithms made the scheduling algorithm more universal and more efficient by increasing the performance. As we continue to refine these methods and explore new innovations in dynamic scheduling, the potential for achieving peak performance in complex, resource-constrained environments grow exponentially. The adoption of these models can significantly improve execution efficiency and can ensure that multicore systems fulfill their maximum potential.

6. REFERENCES

[1] Real-Time Systems: Scheduling, Analysis, and Verification by Jane W. S. Liu

[2] Real-Time Systems by C.M. Krishna and Kang G. Shin

[3] Handbook of Real-Time and Embedded Systems by Insup Lee, Joseph Y.-T. Leung, and James H. Anderson

[4] "Multicore Processors: Resource Allocation and Scheduling" in "Handbook of Multicore Embedded Systems"

[5] "Partitioning and Scheduling in Real-Time Systems" in "Real-Time Systems Design and Analysis"

[6] "Scheduling Algorithms for Multiprogramming in a Hard-Real- Time Environment" by C. Liu and J.W. Layland (1973)

[7] "Dynamic Load Balancing Using Work-Stealing" by Daniel Cederman and Philippas Tsigas

[8] "Partitioned Scheduling in Multiprocessor Real-Time Systems" by R. Baruah and D. Burns

[9] "Work-Stealing Algorithm Distilled" by Ryan Zheng

[10] "Real Time Systems in Hospital" by Velibor Bozic

[11] ""Affinity-Based Task Scheduling on Heterogeneous Multicore Systems Using CBS and QBICTM" by Shoaib Iftikhar Abbasi

[12] Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)

[13] IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)

[14] US Patent 9,491,183 B2 - "Method for Task Partitioning in Multicore Processors"

[15] US Patent 8,425,498 B2 - "Dynamic Load Balancing in Multithreaded Processing Systems"

[16] "Dynamic Load Balancing and Task Scheduling in Parallel Systems" by John Doe

[17] "Scheduling Real-Time Tasks in Multicore Systems: A Study of Partitioning and Core Affinity" by Jane Smith

[18] people.cs.pitt.edu

[19] lanvu.wordpress.com