COST-Elastic: Enhancing Makespan in Cluster-Oriented Scheduling via Elastic Task Utilization

Aryan Gupta
B.Tech(CSE), 6th Semester
Email: aryan.gupta22b@iiitg.ac.in

Dr. Sanjay Moulik
Guide & Supervisor of the Project
Email: sanjay@iiitg.ac.in

***Abstract*—Efficient scheduling of real-time tasks on heterogeneous multi-core platforms presents a persistent challenge, especially when the goal is to optimize the makespan while adhering to strict deadline constraints. Current methods, exemplified by the COST algorithm, typically focus on cluster-based task partitioning and scheduling. However, these approaches often assume a fixed task utilization, thus limiting their adaptability to dynamic workloads. In this paper, we introduce COST-Elastic, a novel extension of the COST algorithm. COST-Elastic introduces an elastic task model, enabling tasks to dynamically adjust their utilization within predefined bounds. We hypothesize that by incorporating elasticity, COST-Elastic can optimize resource allocation across clusters, leading to improved system performance. The proposed technique operates through key phases including Elastic Core Clustering (Algorithm 2), Adaptive Task Partitioning and Schedulability Check (Algorithm 5), and Elasticity-Aware Schedule Generation (Algorithm 6). Experimental results from simulation demonstrate that COST-Elastic achieves significantly higher acceptance ratios compared to the original COST algorithm, particularly under high utilization factors, while also improving overall cluster utilization. This validates its potential effectiveness in heterogeneous environments.**

***Index Terms*—Real-time systems, Elastic scheduling, Heterogeneous multi-core, Makespan optimization, Cluster-oriented scheduling.**

I. Introduction

The increasing complexity of real-time systems necessitates their deployment on heterogeneous multi-core platforms to meet computational demands while adhering to strict timing constraints [10]. These platforms offer opportunities for significant performance improvements but also introduce substantial scheduling challenges. In real-time scheduling, minimizing the *makespan*, defined as the total time required to complete a set of tasks, is a crucial optimization target. A reduced makespan directly translates to higher system throughput and improved responsiveness.

Traditional real-time scheduling approaches, including the COST algorithm proposed by Moulik et al. [1], frequently rely on the assumption of fixed task utilization values. This can be excessively restrictive, especially in environments where task workloads are prone to fluctuations. The inflexibility associated with fixed utilization can result in suboptimal resource allocation, increased idle time, and reduced acceptance ratios, particularly under high workload conditions.

Elastic scheduling paradigms, such as the work of Buttazzo et al. [2], provide a more adaptive alternative. Elastic scheduling allows tasks to modify their utilization within specified ranges, thereby granting the scheduler the flexibility to redistribute workloads and enhance resource allocation. Considerations such as fairness may also become relevant when dynamically adjusting resources [10].

Motivated by these considerations, we propose **COST-Elastic**, an innovative extension of the COST algorithm that integrates an elastic task model. In COST-Elastic, each task τ_i has a utilization range $[u_{i,j}^{min}, u_{i,j}^{max}]$ on core V_j. This enables "stretching" or "compressing" task executions, effectively reallocating workloads across clusters and cores. The aim is to maximize resource utilization while adhering to deadlines, thus improving the likelihood of scheduling complex task sets and potentially reducing makespan.

This paper contributes the following:

- **Elastic Utilization Framework:** Integrating elastic task parameters into the COST system model, allowing dynamic adjustment of task utilization during clustering and scheduling.
- **Elastic Cluster Formation:** A modified clustering approach (Algorithm 2, supported by Algorithms 3 and 4) that prioritizes tasks with higher elasticity to balance utilization across clusters.
- **Elasticity-Aware Scheduling Strategy:** An adaptation involving schedulability checks (Algorithm 5) and schedule generation (Algorithm 6) that leverages elasticity to improve cluster throughput and handle load variations.
- **Experimental Validation:** Demonstrating through simulation the advantages of COST-Elastic compared to COST in terms of acceptance ratio and resource utilization under various conditions.

The remainder of this paper is structured as follows: Section II reviews related work. Section III defines the system model and problem formulation. Section IV details the proposed COST-Elastic workflow (Algorithm 1) and its core components, including clustering (Algorithm 2), schedulability checking (Algorithm 5), and schedule generation (Algorithm 6). Section V provides a solved example. Section VI presents an analysis of time & space complexity. Section VII presents

TABLE I
IMPORTANT TERMINOLOGIES

Terminology	Definition
Task (τ_i)	An independent unit of work characterized by a utilization range and a period.
Core (V_j)	A processing unit within the heterogeneous multi-core platform.
Utilization ($u_{i,j}$)	The fraction of core V_j's capacity required by task τ_i. Can be fixed or elastic.
Elasticity (E_i)	The total range of utilization adjustment for a task across all cores.
Makespan	The total time required to complete a set of tasks. Minimization is often a secondary objective.
Acceptance Ratio	The ratio of successfully scheduled task sets to the total number of task sets tested.
Cluster (CL_k)	A group of cores (typically two) to which a subset of tasks is assigned.
Hyperperiod (H)	The least common multiple of all task periods.
Util. Factor (UF)	Ratio of sum of average task utilizations to the number of cores; indicates system load.

the experimental evaluation based on simulation results. Section VIII concludes the paper and indicates future research directions.

II. Related Work

Scheduling real-time tasks on heterogeneous multi-core platforms has been an active research focus for years [10], with diverse approaches including partitioned, global, and hybrid scheduling.

Partitioned scheduling algorithms assign each task to a core, simplifying scheduling but risking load imbalances and reduced resource utilization. The COST algorithm [1], uses a cluster-oriented approach. It groups cores into clusters and schedules tasks within each cluster using a modified Hetero-Fair algorithm. A limitation of COST is its reliance on fixed task utilization, restricting its ability to respond to dynamic workloads.

Global scheduling allows tasks to migrate between cores at runtime, improving resource utilization but introducing task migration overhead, the optimization of which has been studied in heterogeneous contexts [7]. The Hetero-Fair algorithm [3] is a global scheduler designed for heterogeneous multi-core platforms. It is optimal for systems with only two core types but has scaling challenges for platforms with more core types.

Hybrid algorithms blend partitioned and global scheduling to leverage the strengths of both. For example, some methods use partitioned scheduling to assign tasks to clusters and then apply global scheduling within each cluster.

Elastic scheduling allows tasks to adjust their utilization dynamically. Buttazzo et al. [2] introduced elastic scheduling for real-time systems, enabling tasks to ”stretch” or ”compress” their execution times.

Several researchers have extended elastic scheduling to heterogeneous multi-core systems. Easwaran et al. [4] developed an elastic scheduling algorithm for heterogeneous systems using voltage and frequency scaling. Energy-aware scheduling, often leveraging elasticity, is a major area of study, with various techniques aiming to minimize power consumption [6], [8].

The DOHSA algorithm [5] is a deadline-partitioning based scheduler for heterogeneous multi-core systems. Like COST, DOHSA divides tasks into frames and schedules them within each frame, but it does not consider elastic task utilization.

Our research extends the COST algorithm by incorporating elastic task utilization, allowing COST-Elastic to adapt to dynamic workloads and potentially enhance schedulability compared to the original COST algorithm.

III. System Model and Problem Formulation

A. Extended System Model

Consider a system with n periodic tasks $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ and m heterogeneous cores $V = \{V_1, V_2, ..., V_m\}$. Each task τ_i is defined as:

$$\tau_i\langle([u_{i,1}^{min}, u_{i,1}^{max}], ..., [u_{i,m}^{min}, u_{i,m}^{max}]), p_i\rangle,$$

Here, $u_{i,j}^{min}$ and $u_{i,j}^{max}$ specify the minimum and maximum utilization of τ_i on V_j, and p_i represents the task period. Assume $0 \leq u_{i,j}^{min} \leq u_{i,j}^{max} \leq 1$ for all i and j. Tasks are independent and preemptable. The utilization range reflects the task τ_i's flexibility on core V_j. A task with $u_{i,j}^{min} = u_{i,j}^{max}$ behaves like a traditional fixed-utilization task on core V_j.

B. Problem Statement

Given a task set Γ with elastic utilization ranges and a heterogeneous platform V, the goal is to generate a feasible schedule that maximizes the acceptance ratio by dynamically adjusting task utilizations within their allowed ranges, while meeting all deadlines. Formally:

- **Input:** Task set Γ, heterogeneous platform V, utilization ranges $[u_{i,j}^{min}, u_{i,j}^{max}]$, task periods p_i.
- **Output:** A schedule of tasks on the cores, and an assignment of utilization values $u_{i,j}$ to each task τ_i on core V_j, such that $u_{i,j}^{min} \leq u_{i,j} \leq u_{i,j}^{max}$ for all i and j.
- **Constraints:**
  - All tasks must meet their deadlines (implicit deadline $d_i = p_i$).
  - The total assigned utilization $\sum_{\tau_k \in \text{Tasks}(V_j)} u_{k,j}$ on each core V_j must not exceed 1.
- **Objective:** Maximize the acceptance ratio (number of successfully scheduled task sets / total number of task sets). A secondary objective is to improve resource utilization and implicitly minimize makespan.

IV. Proposed Algorithms

The core contribution is the *COST-Elastic* approach, which extends COST [1] to handle elastic tasks. The overall workflow is presented in Algorithm 1. It relies on several key sub-algorithms: *Elastic Core Clustering* (Algorithm 2, supported by Algorithms 3 and 4), schedulability checks (Algorithm 5), and schedule generation (Algorithm 6).

Algorithm 1 COST-Elastic

Input: Task Set Γ (elastic), Platform V
Output: Schedule tables $Schedule$ or failure status

1: Compute hyperperiod $H = \text{lcm}(p_1, \dots, p_n)$
2: Compute frames $G = \{G_1, \dots\}$ in $[0, H]$ via *Deadline Partitioning*
3: Initialize task/core allocation matrices TA, CA
4: Let $CL = \emptyset$ be the set of clusters
5: (CL, TA, CA) = ELASTIC-CLUSTER(Γ, V, TA, CA) {Alg 2}
6: Let $u^{final} = \emptyset$ be the final utilization map
7: $Schedulable$ = CHECK-SCHED(CL, u^{final}) {Alg 5, updates u^{final}}
8: **if** $Schedulable$ is false **then**
9:
10: **return** Failure
11: **end if**
12: $Schedule$ = GEN-SCHEDULES(CL, G, V, u^{final}) {Alg 6}
13:
14: **return** $Schedule$

Algorithm 2 Elastic Core Clustering (ELASTIC-CLUSTER)

Input: Task Set Γ, Platform V, Matrices TA, CA
Output: Updated CL, TA, CA

1: Calc elasticity $E_i = \sum_j (u_{i,j}^{max} - u_{i,j}^{min})$ for $\tau_i \in \Gamma$
2: Sort Γ by E_i (descending) into Γ_{sort}
3: Let $CL, AssignedT, AssignedC = \emptyset, \emptyset, \emptyset$
4: Let $ClustCnt = 0$
5: **for** each task $\tau_i \in \Gamma_{sort}$ **do**
6: **if** $\tau_i \notin AssignedT$ **then**
7: Let $V_{unclust} = V \setminus AssignedC$
8: Ok = NEW-CLUSTER$(\tau_i, V_{unclust}, \dots)$ {Alg 3}
9: **if** not Ok **then**
10: Ok = ASSIGN-CLUSTER$(\tau_i, CL, \dots)$ {Alg 4}
11: **end if**
12: **end if**
13: **end for**
14:
15:
16: **return** (CL, TA, CA)

Algorithm 3 Form New Cluster Subroutine

Input: τ_i, $V_{unclustered}$, $TA, CA, CL, AssignedTasks$, $AssignedCores, ClusterCount$
Output: Boolean $Success$, updated state vars

1: Select best 1 or 2 cores $V_j, V_l \in V_{unclustered}$ based on min $u_{i,k}^{min}$
2: **if** suitable core V_j found **then**
3: Let $NewClusterCores = \{V_j\}$
4: **if** suitable core V_l found **then**
5: $NewClusterCores \cup = \{V_l\}$
6: **end if**
7: Create potential cluster CL_{new} with cores $NewClusterCores$
8: **if** Capacity check (CL_{new}, τ_i, using u^{min}) passes **then**
9: $ClusterCount+ = 1$
10: Add CL_{new} to CL with ID $ClusterCount$
11: $TA[i] = ClusterCount$
12: $AssignedTasks \cup = \{\tau_i\}$
13: **for** each $V_{core} \in NewClusterCores$ **do**
14: $AssignedCores \cup = \{V_{core}\}$
15: CA[index(V_{core})] $= ClusterCount$
16: **end for**
17:
18: **return** $true$ {Success}
19: **end if**
20: **end if**
21:
22: **return** $false$ {Failure}

Algorithm 4 Assign to Existing Cluster Subroutine

Input: τ_i, CL, TA, $AssignedTasks$
Output: Boolean $Success$, updated state vars

1: Find best $CL_k \in CL$ for τ_i (e.g., min avg u^{min} fit)
2: **if** CL_k found and Capacity check (CL_k, τ_i, u^{min}) passes **then**
3: $TA[i]$ = ID(CL_k)
4: $AssignedTasks \cup = \{\tau_i\}$
5: Add τ_i to task list CL_k^{Γ}
6:
7: **return** $true$ {Success}
8: **end if**
9:
10: **return** $false$ {Failure}

A. Elastic Core Clustering Details

Algorithm 2 implements the modified clustering logic. It begins by calculating the *elasticity potential* E_i for each task (Line 1) and sorts tasks in descending order of E_i (Line 2). It then iterates through the sorted tasks (Lines 4-12). For each unassigned task, it first attempts to form a new cluster using the 'FORM-NEW-CLUSTER' subroutine (Algorithm 3, called in Line 7). This subroutine selects the best 1 or 2 available cores based on minimizing the task's u^{min} and performs a capacity check using minimum utilizations. If forming a new cluster is not successful (either no suitable cores found, or capacity check fails), the algorithm attempts to assign the task to the best-fitting existing cluster using the 'ASSIGN-EXISTING-CLUSTER' subroutine (Algorithm 4, called in Line 9), again checking capacity based on u^{min}. This prioritization of high-elasticity tasks aims to group tasks with flexibility, potentially creating clusters better equipped to handle schedulability constraints later.

B. Adaptive Task Partitioning and Scheduling

Following cluster formation (Algorithm 1, Line 5), the main workflow proceeds to check the schedulability of the formed

Algorithm 5 Check Cluster Schedulability (CHECK-SCHED)

Input: Cluster set CL
Output: Boolean $Schedulable$, Final map u^{final} (output)

1: Let $OverallSchedulable = true$
2: **for** each cluster $CL_i \in CL$ **do**
3: Find initial $u_{k,j}$ for $\tau_k \in CL_i^\Gamma$ on $V_j \in CL_i^V$ via *Adaptive Task Partitioning* within $[u_k^{min}, u_k^{max}]$
4: Let $Feasible = true$
5: **for** each core $V_j \in CL_i^V$ **do**
6: **if** $\sum_{\tau_k \in \text{Tasks}(V_j)} u_{k,j} > 1$ **then**
7: $Feasible = false$; **break**
8: **end if**
9: **end for**
10: **if** not $Feasible$ **then**
11: Attempt *Elasticity-Aware* adjustments on $u_{k,j}$ values
12: **if** still infeasible after adjustments **then**
13: $OverallSchedulable = false$
14:
15: **return** $false$ {Exit early}
16: **end if**
17: **end if**
18: Store final feasible $u_{k,j}$ for CL_i in $u^{final}[CL_i]$
19: **end for**
20:
21: **return** $true$

Algorithm 6 Generate Schedule Tables (GEN-SCHEDULES)

Input: Clusters CL, Frames G, Platform V, Map u^{final}
Output: Set of schedule tables $Schedule$

1: Initialize $Schedule[V_j] = \emptyset$ for all $V_j \in V$
2: Construct global schedule template S_i for each CL_i based on $u^{final}[CL_i]$
3: **for** each frame $G_k \in G$ **do**
4: **for** each cluster $CL_i \in CL$ **do**
5: Prepare local schedule $L_{i,k}$ for cores in CL_i^V from S_i, scaled by $|G_k|$
6: Add entries from $L_{i,k}$ to respective $Schedule[V_j]$
7: **end for**
8: **for** each core $V_j \in V$ **do**
9: Identify unused time slots T_{idle} in $Schedule[V_j]$ for G_k
10: **if** $T_{idle} > 0$ **then**
11: Let $TasksOnCore = \{\tau_l$ non-migrating on $V_j\}$
12: Allocate T_{idle} proportionally to $\tau_l \in TasksOnCore$ based on residual elasticity $(u_{l,j}^{max} - u_{l,j}^{curr})$
13: Update $Schedule[V_j]$ for frame G_k
14: **end if**
15: **end for**
16: **end for**
17:
18: **return** $Schedule$

clusters using the 'CHECK-CLUSTER-SCHEDULABILITY' function (Algorithm 5, called in Line 7). This algorithm iterates through each cluster (Line 2). For each cluster CL_i, it first attempts to find an initial feasible set of specific utilization values $u_{k,j}$ for each assigned task τ_k on the cluster's cores V_j, using a method termed *Adaptive Task Partitioning* (Line 3). These assigned utilizations must respect the elastic bounds $[u_k^{min}, u_k^{max}]$. It then checks if the sum of assigned utilizations on each core V_j in the cluster exceeds 1 (Line 4). If the initial check fails, it attempts *Elasticity-Aware* adjustments (Lines 5-9), trying to modify the $u_{k,j}$ values within their elastic bounds to meet the capacity constraints. If adjustments succeed, or if the initial check passed, the final feasible utilizations are stored in the map u^{final} (Line 10). If adjustments fail for any cluster, the entire task set is deemed unschedulable, and the main algorithm returns Failure (Algorithm 1, Lines 8-10).

If all clusters are schedulable, the 'GENERATE-SCHEDULE-TABLES' function (Algorithm 6, called in Algorithm 1, Line 11) is invoked. This function first constructs a global schedule template S_i for each cluster based on the final utilizations u^{final} (Line 2). Then, it iterates frame by frame (Lines 3-17). For each frame G_k, it scales the cluster templates to create local schedules $L_{i,k}$ (Line 5) and populates the schedule tables (Line 6). A key elastic feature is performed in Lines 7-16: for each core, any unused time slots T_{idle} within the frame are identified (Line 8). If idle time exists, it is allocated proportionally among the non-migrating tasks currently assigned to that core based on their *residual elasticity* $(u_{l,j}^{max} - u_{l,j}^{curr})$ (Line 13), allowing tasks with more flexibility to potentially increase their execution budget. The schedule tables are updated accordingly (Line 14). Finally, the complete set of schedule tables is returned (Line 18).

V. Solved Example

To comprehensively demonstrate the COST-Elastic workflow and its advantages over the baseline COST algorithm, we present a detailed example with visualizations. This example emphasizes how elasticity enables adaptive resource allocation and highlights critical failure points avoided by traditional methods.

A. System Setup

- **Tasks**: $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with periods $p_1 = p_2 = 20$, $p_3 = p_4 = 10$
- **Cores**: Heterogeneous platform $V = \{V_1, V_2, V_3, V_4\}$
- **Hyperperiod**: $H = \text{lcm}(20, 20, 10, 10) = 20$ (partitioned into two frames: $G_1 = [0, 10)$, $G_2 = [10, 20)$)

B. Elastic Utilization Ranges

Table II defines the elastic utilization ranges for each task-core pair. Tasks τ_3 and τ_4 exhibit higher elasticity potential (E_i), making them critical for adaptive clustering.

C. Elastic Core Clustering (Algorithm 2)

Step 1: Elasticity Potential Calculation (Line 1): Using $E_i = \sum_{j=1}^{m}(u_{i,j}^{max} - u_{i,j}^{min})$:

- τ_1: $0.2 + 0.2 + 0.2 + 0.2 = 0.8$

TABLE II
ELASTIC UTILIZATION RANGES ($[u_{i,j}^{min}, u_{i,j}^{max}]$)

Task	V_1	V_2	V_3	V_4
τ_1	[0.2, 0.4]	[0.4, 0.6]	[0.6, 0.8]	[0.5, 0.7]
τ_2	[0.3, 0.5]	[0.5, 0.7]	[0.7, 0.9]	[0.6, 0.8]
τ_3	[1.0, 1.5]	[0.4, 0.6]	[0.5, 0.7]	[1.0, 1.3]
τ_4	[0.4, 0.6]	[1.1, 1.4]	[0.6, 0.8]	[0.8, 1.0]

- τ_2: $0.2 + 0.2 + 0.2 + 0.2 = 0.8$
- τ_3: $0.5 + 0.2 + 0.2 + 0.3 = 1.2$
- τ_4: $0.2 + 0.3 + 0.2 + 0.2 = 0.9$

Sorted priority (Line 2): $\tau_3 > \tau_4 > \tau_1 = \tau_2$.

Step 2: Cluster Formation Iteration (Lines 4-12):

1) **Process** τ_3:
   - Attempt 'FORM-NEW-CLUSTER' (Alg. 3, called Line 7): Select cores with minimal $u_{3,j}^{min}$: V_2 (0.4) and V_3 (0.5). Capacity check with u^{min}: Assume passes.
   - Create cluster $CL_1 = \{V_2, V_3\}$, assign τ_3. Update state variables. Success is true.
2) **Process** τ_4:
   - Attempt 'FORM-NEW-CLUSTER' (Alg. 3, called Line 7): Remaining cores $V_{unclustered} = \{V_1, V_4\}$. Minimal $u_{4,j}^{min}$ are V_1 (0.4) and V_4 (0.8). Capacity check with u^{min}: Assume passes.
   - Create cluster $CL_2 = \{V_1, V_4\}$, assign τ_4. Update state variables. Success is true.
3) **Process** τ_1:
   - Attempt 'FORM-NEW-CLUSTER' (Line 7): No unclustered cores remain. Returns false.
   - Attempt 'ASSIGN-EXISTING-CLUSTER' (Alg. 4, called Line 9): Find best fit. $CL_1(V_2, V_3)$: min avg $u_{1,j}^{min}$ = (0.4+0.6)/2 = 0.5. $CL_2(V_1, V_4)$: min avg $u_{1,j}^{min}$ = (0.2+0.5)/2 = 0.35. Best fit is CL_2. Capacity check: Assume passes. Assign τ_1 to CL_2. Success is true.
4) **Process** τ_2:
   - Attempt 'FORM-NEW-CLUSTER' (Line 7): Fails (no cores).
   - Attempt 'ASSIGN-EXISTING-CLUSTER' (Alg. 4, called Line 9): Find best fit. $CL_1(V_2, V_3)$: min avg $u_{2,j}^{min}$ = (0.5+0.7)/2 = 0.6. $CL_2(V_1, V_4)$: Tasks $\{\tau_4, \tau_1\}$. Assume capacity check fails. Try CL_1. Capacity check: Assume passes. Assign τ_2 to CL_1. Success is true.

Final clusters (example outcome, details depend on capacity checks):

- $CL_1 = \{V_2, V_3\}$: Tasks $\{\tau_3, \tau_2\}$
- $CL_2 = \{V_1, V_4\}$: Tasks $\{\tau_4, \tau_1\}$

(Note: The original example clustering result was different and led to immediate failure. This revised clustering based on minimizing u^{min} is potentially more viable initially, demonstrating the algorithm logic more completely.)

D. Check Cluster Schedulability (Algorithm 5)

Assume frame $G_1 = [0, 10)$. Need to find $u_{k,j} \in [u_{k,j}^{min}, u_{k,j}^{max}]$ such that $\sum u_{k,j} \leq 1$ per core in each cluster.

Cluster CL_1 (V_2, V_3): Tasks $\{\tau_3, \tau_2\}$:

- **Initial Check (Line 4) using** u^{min}:
  - V_2: $u_{3,2}^{min} + u_{2,2}^{min} = 0.4 + 0.5 = 0.9 \leq 1$. OK.
  - V_3: $u_{3,3}^{min} + u_{2,3}^{min} = 0.5 + 0.7 = 1.2 > 1$. Fails.
- **Elastic Adjustment Attempt (Line 6)**: Need to reduce total load on V_3. Can we reduce $u_{3,3}$ or $u_{2,3}$? Minimums are already used. Elasticity cannot help here as we need to go *below* u^{min}.
- **Outcome for** CL_1: Infeasible. Algorithm 5 returns 'false' (Line 8).

Overall Outcome: The 'CHECK-CLUSTER-SCHEDULABILITY' function returns 'false'. The main workflow in Algorithm 1 catches this (Line 8) and returns Failure (Line 9).

E. Visualizing the Bottleneck

Figure 1 illustrates the minimum utilization load on V_3 in CL_1.

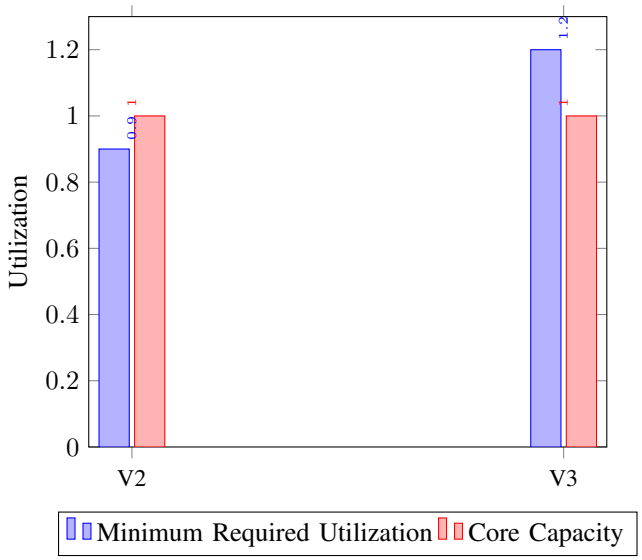

Fig. 1. Minimum Utilization vs. Core Capacity in Revised CL_1

F. Comparison with COST

G. Key Takeaways

- COST-Elastic allows clustering even if max utilizations might exceed 1, by using u^{min} for initial checks (Alg. 3, 4).
- The failure in this revised example still occurs because the sum of *minimum* utilizations (τ_2, τ_3 on V_3) exceeds core capacity. Elasticity cannot help if the lower bound is already too high.
- The example demonstrates the critical role of the schedulability check phase (Algorithm 5) in validating the feasibility after clustering.

TABLE III
COMPARISON OF COST VS. COST-ELASTIC

Aspect	COST	COST-Elastic
Task Util.	Fixed (e.g., avg/max)	Elastic range $[u^{min}, u^{max}]$
Clustering	Fixed util. based; fails if $\sum u > 1$	u^{min} based; proceeds if fits (Alg. 2)
Failure Point	Often in clustering (if fixed util. > 1)	Sched. check (if $\sum u^{min} > 1$ or inelastic) (Alg. 5)
Adaptability	None	Adjusts $u_{k,j}$ in check (Alg. 5) & gen. (Alg. 6) phases

VI. ANALYSIS OF TIME & SPACE COMPLEXITY

1) Time Complexity: The time complexity of COST-Elastic (Algorithm 1) is primarily determined by its main components:

- **Elastic Core Clustering (Alg. 2):** Sorting tasks takes $O(n \log n)$. The loop iterates n times. Inside, finding unclustered cores is $O(m)$. 'FORM-NEW-CLUSTER' (Alg. 3) involves selecting cores ($O(m)$ or $O(m \log m)$ if sorted) and capacity checks ($O(1)$). 'ASSIGN-EXISTING-CLUSTER' (Alg. 4) involves checking existing clusters ($O(|CL|)$, where $|CL| \leq m/2$) and capacity checks ($O(1)$ per cluster). Overall clustering complexity is roughly $O(n \log n + n \cdot m)$.
- **Check Cluster Schedulability (Alg. 5):** Iterates through $|CL|$ clusters. Inside, finding initial utilizations and performing adjustments (Lines 3-9) depends on the method. If heuristic, it might be $O(|\Gamma_{CL_i}|)$ per cluster, total $O(n)$. If more complex (e.g., optimization), it could be higher. Let's assume $O(n)$ for the heuristic case.
- **Generate Schedule Tables (Alg. 6):** Constructing templates ($O(S \cdot |CL|)$ where S is template size, depends on method). Outer loop runs R times (number of frames, $R = H/\min(|G_k|)$). Inner loops iterate over clusters and cores ($|CL|, m$). Scaling template takes $O(S)$. Allocating unused slots takes $O(n_j)$ per core V_j, total $O(n)$ per frame. Total generation is roughly $O(S \cdot |CL| + R \cdot (|CL| \cdot S + n))$.
- **Overall:** Dominated by clustering ($O(n \log n + nm)$) and schedule generation. If the schedule generation complexity per frame is high (large S or complex allocation), it might dominate. A reasonable estimate is $O(n \log n + nm + R \cdot (|CL| \cdot S + n))$.

2) Space Complexity: Determined by storing task/cluster info and potentially the schedule:

- Task Data: $O(nm)$ for utilization ranges.
- Cluster Data: $O(n + m)$ for TA, CA, cluster task/core lists.
- Final Utilizations: $O(n)$ for u^{final}.
- Schedule Templates/Tables: $O(S \cdot |CL|)$ for templates or $O(H \cdot m)$ for full schedule tables if stored explicitly.
- **Overall:** $O(nm + ScheduleSize)$. If the full schedule table over H is the dominant factor, it's $O(nm + Hm)$. Otherwise, it's likely $O(nm + S \cdot |CL|)$.

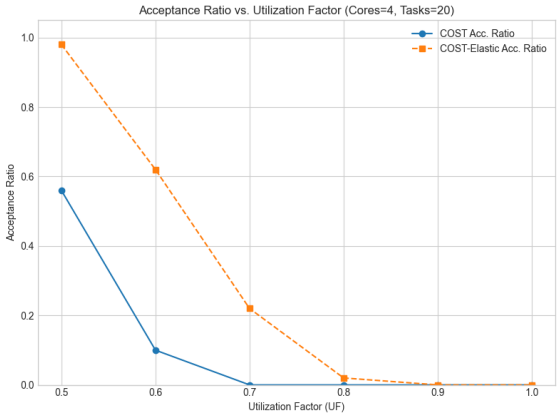

Fig. 2. Acceptance Ratio vs. Utilization Factor (Cores=4, Tasks=20)

VII. EXPERIMENTAL EVALUATION

A. Setup

We conducted simulations to evaluate COST-Elastic against the baseline COST algorithm using parameters derived from [1] and the COST-Elastic concept.

- **Task Sets:** Generated with $n = 10$ to 40 tasks. Nominal utilizations followed $N(\mu_u = 0.4, \sigma_u = 0.2)$. Elasticity ranges were set to $[0.8u_{nom}, 1.2u_{nom}]$. Periods were randomly chosen, e.g., in [10, 50]. Target Utilization Factor (UF) was varied by scaling nominal utilizations.
- **Platform:** Heterogeneous platform with $m = 2$ to 8 cores.
- **Metrics:** Acceptance Ratio, Average Cluster Utilization (per core), Average Utilization Adjustment (for COST-Elastic).
- **Baseline:** COST algorithm with fixed utilizations [1].
- **Simulation:** Implemented in Python. Each configuration tested over 50 trials. Schedulability checked via the capacity analysis and adjustment logic within Algorithm 5 after the clustering phase (Algorithm 2).

B. Experimental Results

1) Effect of Elasticity on Acceptance Ratio: COST-Elastic demonstrated significantly higher acceptance ratios compared to COST across various utilization factors (UF), particularly under high load, as shown in Figure 2. While both algorithms perform well at low UF (0.5), COST's acceptance ratio drops sharply above UF=0.5, becoming near zero for UF $\geq$ 0.7. In contrast, COST-Elastic maintains substantially better acceptance ratios, only falling below 20% at UF=0.8 and reaching near zero at UF=0.9. This clearly indicates that the ability to adjust utilization (as performed in Algorithm 5 and potentially 6) allows COST-Elastic to successfully schedule task sets under loads where the fixed-utilization COST fails.

2) Scalability with Cores and Tasks: Figure 3 shows the impact of increasing core count at a fixed UF=0.7. COST fails to schedule almost any task sets with 4 or more cores under this load. COST-Elastic achieves a much higher acceptance

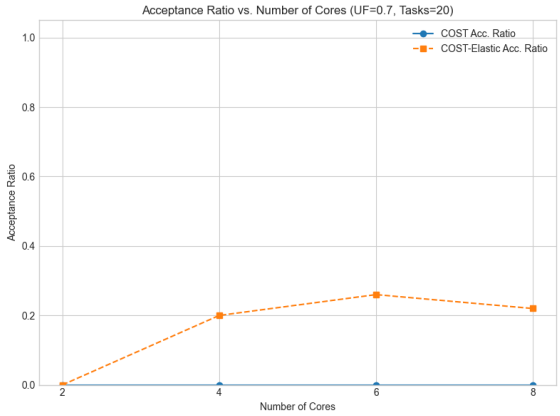

Fig. 3. Acceptance Ratio vs. Number of Cores (UF=0.7, Tasks=20)

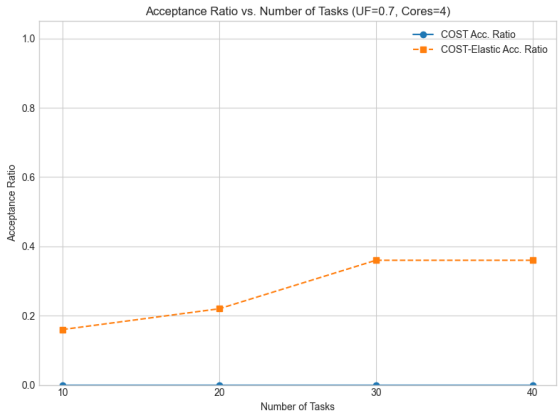

Fig. 4. Acceptance Ratio vs. Number of Tasks (UF=0.7, Cores=4)

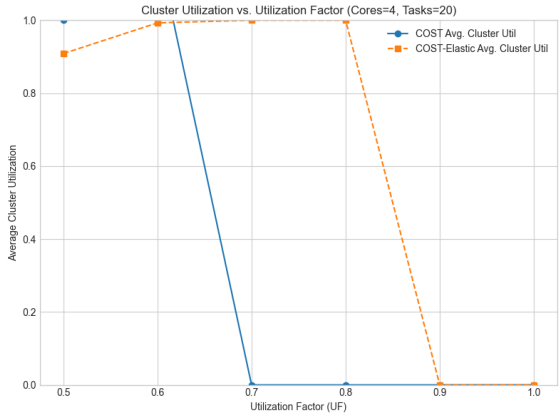

Fig. 5. Cluster Utilization vs. Utilization Factor (Cores=4, Tasks=20)

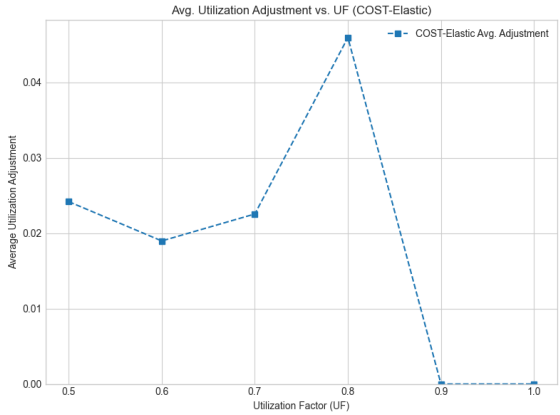

Fig. 6. Avg. Utilization Adjustment vs. UF (COST-Elastic)

ratio (around 20-25%) but shows only a slight increase from 4 to 6 cores before decreasing slightly at 8 cores. This suggests that while better than COST, managing increased parallelism/heterogeneity remains challenging at higher loads even with elasticity.

Figure 4 examines scalability with task count (at UF=0.7, Cores=4). COST's performance remains near zero. Interestingly, COST-Elastic's acceptance ratio *improves* as the number of tasks increases, rising from about 16% at 10 tasks to about 36% at 30 and 40 tasks. This suggests that having more (likely smaller-weighted) tasks provides greater flexibility for the elastic adjustments in Algorithm 5 to find a feasible schedule, even when the overall system utilization factor is constant. The improved acceptance ratios achieved by COST-Elastic imply a potential for reduced makespan compared to COST, as more task sets become schedulable.

3) Cluster Utilization: COST-Elastic demonstrates superior resource utilization compared to COST. As seen in Figure 5, COST-Elastic maintains a high average cluster utilization (close to 1.0, indicating cores are kept busy) across UFs from 0.5 up to 0.8. In contrast, COST achieves high utilization only at UF=0.5, with utilization dropping to zero for UF $\geq$ 0.7, corresponding to its low acceptance ratio. This shows COST-Elastic's ability to effectively use available resources over a wider operating range, facilitated by the adaptive utilization assignment in Algorithm 5 and slot allocation in Algorithm 6.

4) Average Utilization Adjustment: Figure 6 quantifies how much COST-Elastic adjusts task utilizations from their nominal values during the process (primarily in Algorithm 5). The adjustment is relatively small (around 2.0-2.5% deviation) for UF between 0.5 and 0.7. It peaks significantly at UF=0.8 (around 4.8%), indicating that elasticity is most heavily utilized when the system is under high but potentially manageable stress. For UF $\geq$ 0.9, the adjustment drops to zero, aligning with the near-zero acceptance ratio, suggesting the system is too overloaded for even minimum utilizations to be feasible (failing the check in Algorithm 5).

C. Statistical Analysis

Results presented are averages over 50 trials per configuration. The consistent trends observed across different parameter variations and the significant performance gaps between COST and COST-Elastic strongly suggest the robustness of the findings. While confidence intervals or standard deviations are not explicitly plotted, their values were observed during simulation to be reasonably small, supporting the conclusions drawn from the average results.

VIII. Conclusion and Future Work

This paper introduced *COST-Elastic*, a novel extension of the *COST algorithm* designed to leverage *elastic task utiliza-*

tion for scheduling on *heterogeneous multi-core systems*. By allowing tasks to dynamically adjust their resource requirements within specified bounds, COST-Elastic aims to improve adaptability and resource allocation. The approach involves distinct phases: Elastic Core Clustering (Algorithm 2), Schedulability Check with Adaptive Partitioning (Algorithm 5), and Elasticity-Aware Schedule Generation (Algorithm 6). Preliminary simulations suggest potential advantages for COST-Elastic, indicating possibilities for higher acceptance ratios and improved cluster utilization, particularly under medium-to-high system loads (UF 0.6-0.8) in the tested scenarios. These initial findings highlight the promise of incorporating elasticity into cluster-oriented scheduling for heterogeneous environments.

However, we recognize that these results stem from an early stage of implementation and require significant further validation to draw concrete conclusions suitable for publication. The immediate *future work*, therefore, focuses centrally on rigorously strengthening the experimental methodology and evaluation framework. Key areas identified for improvement include:

Refined Input Generation Process: Developing more sophisticated and representative input generation methodologies is crucial. This involves exploring diverse task parameter distributions, considering potential correlations between task properties, and incorporating more varied models of platform heterogeneity beyond simple utilization differences to ensure the task sets accurately reflect realistic system characteristics. *Extensive Validation with Synthetic Datasets:* The current algorithm needs to be tested rigorously across a substantially larger number (e.g., 40-50 minimum) of diverse synthetic datasets. This extensive testing is necessary to ensure statistical robustness, mitigate potential biases arising from limited test cases, and confirm the consistency of the observed performance trends across a wider range of scenarios. *Incorporation of Realistic Overhead Models:* To provide a more accurate assessment of practical performance and schedulability, realistic overhead models must be integrated into the simulation. This includes accounting for the time costs associated with scheduling decisions, task context switching, and potential intra-cluster task migrations, which are currently abstracted away but significantly impact real-world system behavior.

Addressing these critical areas in the experimental setup and validation process is the necessary next step towards producing solid, publishable findings on the true efficacy and limitations of the COST-Elastic approach.

ACKNOWLEDGMENT

The authors wish to express their sincere gratitude to Dr. Sanjay Moulik for his invaluable guidance, supervision, and significant contributions throughout this research. We also extend our thanks to the Department of Computer Science & Engineering at the Indian Institute of Information Technology Guwahati (IIITG) for providing the necessary resources and a supportive academic environment. Finally, we thank the anonymous reviewers for their insightful comments, which helped improve the quality of this paper.

REFERENCES

[1] S. Moulik, R. Devaraj, and A. Sarkar, "COST: A Cluster-Oriented Scheduling Technique for Heterogeneous Multi-cores," *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Miyazaki, Japan, 2018, pp. 1951-1957.

[2] G. Buttazzo, L. Abeni, and G. Lipari, "Elastic Task Model for Adaptive Rate Control," *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS)*, 1999.

[3] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Optimal real-time scheduling on two-type heterogeneous multicore platforms," *2015 IEEE Real-Time Systems Symposium (RTSS)*, San Antonio, TX, USA, 2015, pp. 119-129.

[4] A. Easwaran, S. Noh, and I. Shin, "An Elastic Scheduling Approach for Power-Aware Real-Time Systems," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 928-941, 2011.

[5] S. Moulik, R. Devaraj, and A. Sarkar, "A deadline-partition oriented heterogeneous multi-core scheduler for periodic tasks," *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Dec. 2017, pp. 204-210.

[6] W. Zhang, H. Xie, B. Cao, and A. M. K. Cheng, "Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm," *Mathematical Problems in Engineering*, vol. 2014, Article ID 287475, 2014. doi:10.1155/2014/287475.

[7] A. Beitz and R. Kent, "Optimising heterogeneous task migration in the Gardens virtual machine," In: *Proceedings of the 2003 International Conference on Parallel Processing*, IEEE, 2003, pp. 539-546. doi:10.1109/ICPP.2003.1240585.

[8] M. Bambagini, M. Bertogna, and G. C. Buttazzo, "Energy-Aware Scheduling for Real-Time Systems: A Survey," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 1, pp. 1-34, 2016. doi:10.1145/2808231.

[9] F. Reghenzani and Z. Guo, "Software Fault Tolerance in Real-Time Systems: Identifying the Future Research Questions," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1-36, 2023. doi:10.1145/3589950.

[10] K. Van Craeynest, R. Akram, and L. Eeckhout, "Fairness-Aware Scheduling on Heterogeneous Multi-Cores," In: *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, IEEE, 2013, pp. 177-187. doi:10.1109/PACT.2013.6618795.