Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ACMR and ACMR-Soft: Contention Aware QPSO Scheduling for Heterogeneous Multiprocessor Systems

Dhruv Avhankar
Vellore Institute of Technology
Swagata Paul
Vellore Institute of Technology
Priyansh Mudela
Vellore Institute of Technology
Berlin M A
berlin.ma@vit.ac.in

Vellore Institute of Technology

Article

Keywords:

Posted Date: November 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11424312/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

ACMR and ACMR-Soft: Contention Aware QPSO Scheduling for Heterogeneous Multiprocessor Systems

Dhruv Avhankar[1], Swagata Paul[1], Priyansh Mudela[1], M.A Berlin[1*]

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu, India .

*Corresponding author(s). E-mail(s): berlin.ma@vit.ac.in;
Contributing authors: avhankar.dhruv2024@vitstudent.ac.in;
swagata.paul2024@vitstudent.ac.in;
priyansh.mudela2024@vitstudent.ac.in;

Abstract

Quantum particle swarm optimisation (QPSO) is used to schedule workloads on heterogeneous multiprocessors under energy, reliability and deadline limits. We propose Adaptive Contention Aware Manifold Reshaping (ACMR), which limits the voltage–frequency (DVFS) modes each module may use, based on how loaded its processor is in the current best schedule. Controlled experiments show that this hard limiting makes schedules worse, and that load-driven limiting performs no better than random limiting. The reason is that changing the allowed modes changes how a particle position is decoded, so the personal best positions stored by the swarm lose their meaning. We propose ACMR-Soft, which keeps the decoder fixed and gives the same load feedback as a small, slowly decreasing push on the particle position. On 30 held-out synthetic trials, ACMR-Soft has 35% fewer deadline violations than HEFT, the same as plain QPSO, and more than a static partition. On 60 real workflow traces with DAG precedence, it gives about 6% higher reliability and 10–12% lower energy than the static partition and original ACMR, with a statistically similar deadline miss rate, but a 14% worse composite objective because missed deadlines are missed by larger margins. These real-workflow gains are shared with plain QPSO.

1 Introduction

Scheduling real time jobs on heterogeneous multiprocessor platforms under joint energy, reliability, and deadline constraints is NP hard in general.[1] Classical list scheduling heuristics such as Heterogeneous Earliest Finish Time (HEFT) [2] produce fast, deterministic schedules by greedily minimizing per module finish time, but they do not explicitly consider energy or reliability. Metaheuristic approaches, in particular Particle Swarm Optimisation (PSO) [3] and its quantum behaved variant QPSO [4], can instead optimise a single composite objective spanning all four concerns simultaneously, at the cost of a harder, higher dimensional search problem.

A natural refinement is to make the optimiser's search space contention aware: restrict which dynamic voltage and frequency scaling (DVFS) modes a module may use based on how congested its assigned processor currently is, so that the optimiser spends its limited iteration budget exploring the modes most likely to matter. This is the idea behind the mechanism we propose in this paper, Adaptive Contention Aware Manifold Reshaping (ACMR), which recomputes a per module DVFS mode restriction at every QPSO iteration from a load pressure score derived from the incumbent (global best) schedule.

In the course of stress testing this mechanism, we found that it does not deliver the deadline violation reduction it is designed to produce; more usefully, we were able to identify *why*: restricting the search manifold changes the mapping from a particle's continuous position to its decoded (processor, mode) assignment. QPSO assumes that a particle's personal best position keeps the same meaning throughout the optimization process. Changing this mapping between iterations can therefore invalidate the stored information. This is a structural problem rather than a tuning problem: we verified that it persists under alternative swarm initialization schemes and delayed reshaping onset.

This paper makes four contributions:

1. We identify and empirically demonstrate the manifold reinterpretation failure mode in load pressure driven search space reshaping for QPSO scheduling (Section 6), via a controlled ablation isolating *whether reshaping helps at all* from *whether load pressure specific reshaping helps beyond generic reshaping*.
2. We propose **ACMR-Soft**, a redesign that preserves the contention feedback signal without ever changing the decode function, by injecting it as an annealed positional drift instead of a hard mode exclusion (Section 7).
3. We generalize the underlying scheduling model from linear task chains to arbitrary directed acyclic graph (DAG) precedence (Section 8.2), enabling evaluation on real workflow execution traces rather than only synthetic random chains.
4. We provide a complete empirical evaluation (a six way synthetic comparison including a genetic algorithm (GA) baseline, an ablation isolating the manifold reinterpretation effect, a manifold threshold sensitivity sweep, and a real workflow validation on 60 traces spanning seven scientific applications), all reported with paired statistics, multiple comparison correction, and effect sizes (Sections 8 to 10).

We report the negative results that motivated the redesign in full, since they are, in our view, more informative than the final numbers alone: they explain *why* naive

dynamic search space adaptation is fragile in this setting, which is a finding likely to generalize beyond this specific mechanism.

2 Related Work

2.1 Heterogeneous and DVFS Aware Scheduling

HEFT [2] remains the standard deterministic baseline for heterogeneous scheduling, using upward rank prioritization and earliest finish time processor assignment; it does not explicitly consider energy or reliability. Min-Min [5] is a task level heuristic that does not naturally extend to module level precedence within a task. GA based schedulers [6] can incorporate multiple objectives but typically require careful operator tuning and converge more slowly than swarm methods at the problem sizes considered here [7]. Energy-efficient real-time task scheduling under dynamic voltage and frequency scaling (DVFS) has also been studied for heterogeneous multicore platforms [8].

Many other list scheduling heuristics have been built on top of the HEFT idea. For example, PEFT [9] uses an optimistic cost table to look ahead before it chooses a processor, and it gives shorter schedules than HEFT on many task graphs. A detailed survey of such static task graph scheduling algorithms is given by Kwok and Ahmad [10]. The basic idea of slowing down a processor to save energy goes back to the early work of Weiser et al. [11] and Yao et al. [12]. Later, Zhu et al. [13] used the free time (slack) left in a multiprocessor schedule to lower the voltage and speed of processors without missing deadlines, and Lee and Zomaya [14] combined makespan and energy into one cost function and used DVFS to reduce the energy of a schedule. Most of these methods choose the DVFS mode with a fixed rule after the processor is chosen. In our work, the processor and the DVFS mode of every module are searched together by the optimiser.

2.2 Reliability Aware Scheduling

Assayad et al. [15] optimise schedule length, reliability and power consumption together and use DVFS to lower power, which shows the reliability and energy trade-off inherent in DVFS. In this paper, we use a per module hazard rate model in which slower modes reduce fault rate but increase completion time (Section 4). Das et al. [16] study joint optimization of energy and reliability on heterogeneous multiprocessor systems.

The link between DVFS and reliability has been studied from more than one side. Zhu et al. [17] showed that lowering the supply voltage and frequency can increase the rate of transient faults, so saving energy with DVFS can also reduce reliability. Dongarra et al. [18] treated makespan and reliability as two separate objectives on heterogeneous systems and showed that the two goals can conflict with each other. So the effect of a DVFS mode on reliability depends on the fault model that is used. In this paper, the fault rate multiplier φ_m of each mode (Section 4) is an input of the model, so a different fault model can be used without changing the scheduler itself.

2.3 Manifold Constrained and Adaptive Parameter Metaheuristics

Halimu et al. [19] keep QPSO particles on a Riemannian manifold for engineering problems, and Chen et al. [20] restrict the PSO search space to valid solutions of combinatorial problems; in both cases the restricted space is fixed prior to the optimization run. The methods surveyed by Karafotias et al. [21] and the method of Mallipeddi et al. [22] adapt step sizes, operator probabilities, or constraint handling strategies online, but in all these cases the *decision space itself* is either fixed throughout the run or adapted in a way that does not retroactively reinterpret previously recorded solutions. To our knowledge, ACMR is the first mechanism in this space to reshape the decode function itself at every iteration from the optimiser's own incumbent solution; as we show in Section 6, this specific design choice is also the source of its performance problem. This paper's contribution is best understood relative to this line of work as a cautionary finding and a fix: changing *what a position means* is less stable than changing *how positions are searched* and the latter is what we suggest instead.

Eiben et al. [23] divide parameter control into three types: deterministic control, where a value changes over time by a fixed rule; adaptive control, where a value changes using feedback from the search; and self-adaptive control, where the value is itself evolved along with the solutions. The annealed coefficient $\beta(t)$ of QPSO is a deterministic control, and its effect on the convergence of a single particle has been studied in detail by Sun et al. [24]. A similar stability analysis for classical PSO is given by Clerc and Kennedy [25]. Using these terms, ACMR-Soft is an adaptive control: it uses feedback from the incumbent schedule, but this feedback only changes how a particle moves and never changes how its position is decoded.

2.4 Encoding and Decoding in Metaheuristic Scheduling

Metaheuristics for scheduling usually do not search directly over schedules. Instead, they search over a simple space, such as vectors of real numbers, and a decoder turns each vector into a schedule. Bean [26] introduced the random key encoding, in which a vector of numbers in $[0, 1)$ is decoded into a sequence by sorting. Gonçalves and Resende [27] later built biased random key genetic algorithms on the same idea and pointed out its main benefit: the search part is general and does not depend on the problem, while all the problem knowledge is kept inside the decoder. The particle encoding used in this paper follows the same pattern, since every position value lies in $[0, 1 - \epsilon]$ and the decoder maps it to a processor and a DVFS mode. In all these works, the decoder stays fixed for the whole run, and the search operators depend on this. ACMR breaks this assumption, and this is the source of the problem we study in Section 6.

2.5 Swarm Memory in Changing Environments

A closely related problem is known in dynamic optimisation, where the objective function itself changes during the run. In such cases the personal and global best positions stored by a PSO swarm can become outdated, and the swarm keeps moving towards points that are no longer good. Hu and Eberhart [28] detect such changes

and respond by resetting the stored best positions and re-randomising part of the swarm. Blackwell and Branke [29] use several swarms that repel each other so that diversity is kept after a change. Jin and Branke [30] give a survey of such methods. The failure mode of ACMR is similar to this outdated memory problem, but there is one important difference. In dynamic optimisation, the change comes from outside, that is, from the problem itself. In ACMR, the problem stays the same and the change is created by the optimiser, because the decoder is reshaped from the optimiser's own incumbent solution. So the outdated memory in our case is self-made, and it can be avoided by design, which is exactly what ACMR-Soft does.

2.6 Workflow Scheduling on Real Traces

Real scientific workflows have been studied in detail. Juve et al. [31] profiled several Pegasus workflows, including Montage and Epigenomics, and reported their structure and resource usage. The WfCommons project [32] makes such workflow instances openly available, along with tools to generate new ones, and the WfInstances traces [33] used in this paper come from this project. PSO based schedulers have also been applied to such workflows. Pandey et al. [34] used PSO to map workflow tasks to cloud resources so that the total cost is reduced, and Rodriguez and Buyya [35] used PSO to meet workflow deadlines while lowering the cost of cloud resources. These works mainly target virtual machines in the cloud, where money cost is the main concern. Our setting is different: we use a fixed set of heterogeneous processors, every module also needs a DVFS mode, and energy, reliability, load balance and deadlines are optimised together.

2.7 Recent Advances in Energy and Workflow Scheduling

Many new studies on this topic have appeared in the last few years. Some of them look at energy, reliability and heat. Moulik and Sharma [36] made a scheduler called FRESH. It keeps a main copy and a backup copy of each task, so the system can still work if a fault happens, and it also tries to save energy. Chakraborty et al. [37] made a scheduler that keeps the temperature of the processor under control. She et al. [38] studied how to save energy while keeping the system reliable on heterogeneous processors. Liu et al. [39] first decided the order of the tasks, then picked a suitable core for each task, and then chose the DVFS mode. They tested this on real hardware. Ahmad et al. [40] made a method called ERWS. It finishes the workflow as fast as possible without going over a fixed energy limit. They also used the Friedman test, the Wilcoxon test and the Holm correction, as we do in this paper. Other studies look at workflows. Hosseini Shirvani [41] and Namdev et al. [42] wrote surveys of workflow scheduling in the cloud. The second survey looks mainly at methods based on artificial intelligence. Zhu et al. [43] used neural networks and reinforcement learning to schedule workflows. Mahjoub et al. [44] mixed an optimisation method with ideas from quantum computing and Q-learning to schedule tasks in the cloud. All of these works are useful. But, none of them changes the way a solution is decoded while the search is running. This is the problem we study in Section 6.

3 Formal Definition of the Problem

3.1 System Setup

For a heterogeneous system (consisting of more than one type of CPU core), the following setup is defined:

1. Let $T = \{T_1, T_2, T_3, \ldots, T_n\}$ be the set of all tasks to be executed in an interleaved manner on the system. For each task $T_i \in T$, a set M_i of modules is defined such that
$$M_i = \{M_{i1}, M_{i2}, M_{i3}, \ldots, M_{ik}\} \tag{1}$$
with cardinality k which must satisfy all the defined precedence constraints (a module-based dependent order of execution). Here, each task T_i is not executed as a single block but it is instead broken into smaller executable units called Modules. Let's say a task is a job then each module is a step inside that job and a task say T_i consists of k modules: M_{i1} to M_{ik}. For e.g., if a task is to process an image then modules are load image, apply filter, compress output and save result. These modules may have precedence constraints i.e. some must run before others so, when scheduling happens at the module level instead of task level, it allows finer control over performance, energy and reliability. Equation (1) thus gives the complete list of modules that make up task T_i.
2. The system contains a set of processors P_i defined by the set P where
$$P = \{P_1, P_2, P_3, \ldots, P_t\} \tag{2}$$
where the number of processors $n(P) = t$, meaning the system consists of t processors. The system contains a set of processors, say t to execute modules where each processor can execute modules that are assigned to it. For e.g., a system might have P_1- high performance core, P_2- energy efficient core, P_3- general purpose core etc. Since, scheduling is about deciding which modules run on which processor, the number and the nature of the processor directly affect performance and energy. Equation (2) lists all the processors on which the modules can be placed.

   Since the system is defined as a heterogeneous processor system, there should exist atleast one processor, say P_i where
$$\text{type}(P_i) \neq \text{type}(P_j) \quad \text{for some } j \neq i. \tag{3}$$
All the processors are assumed to be capable of operating in multiple frequency-voltage modes. Here, the system is heterogeneous which means not all the processors are identical so, there can be at least 1 processor who is different in architecture or capability, differences can be in terms of speed, power consumption, reliability, supported voltage frequency-levels etc. Since, energy aware scheduling only makes sense when processors behave differently, the scheduler must intelligently map modules to suitable processors. For e.g., if a system has two cores, one core optimised for speed and another for low power then assigning the same task to both would give different energy outcomes. Equation (3) is the condition which makes the system heterogeneous.

3. Each processor P_j supports a set of frequency-voltage modes which are enclosed in the set S defined as

$$S = \{S_{j1}, S_{j2}, S_{j3}, \ldots, S_{jl}\} \tag{4}$$

with l different frequency-voltage modes. Each processor P_j can operate in multiple voltage-frequency modes i.e., DVFS modes, a processor doesn't always run at full speed and can switch between l operating modes where each mode corresponds to a specific voltage-frequency pair. For e.g., Let's say a processor has 3 modes, S_{j1}-high voltage and high frequency i.e. fast but power hungry, S_{j2}-medium voltage and medium frequency and S_{j3}- low voltage and low frequency i.e. energy efficient. Lower frequency corresponds to lower energy and slower execution whereas higher frequency corresponds to higher energy and fast execution. Thus, the scheduler must choose the best mode for each processor to save energy while meeting deadlines. Equation (4) lists these DVFS modes for processor P_j; in all our experiments $l = 3$ (Section 4).

3.2 System Constraints

1. **Processor exclusivity constraint:**

$$\sum_i x_{ij}\, \mathbf{1}\left[t_i^{\text{start}} \leq \tau < t_i^{\text{finish}}\right] \leq 1 \quad \forall P_j \in P,\ \forall \tau \tag{5}$$

which means that at most one module executes on a processor at any given instant. Equation (5) is checked for every processor in the system. Here τ is any time instant, and $\mathbf{1}[\cdot]$ is 1 when module M_i is running at time τ and 0 otherwise.

2. **Precedence constraint:**

$$t_{i(r+1)}^{\text{start}} \geq t_{ir}^{\text{finish}} \quad \forall M_{ir}, M_{i(r+1)} \in M_i \tag{6}$$

It guarantees that module execution order within a task abides by the task dependency. In Section 8.2, we extend Eq. (6) from a linear chain of modules to a general DAG. In general form, this becomes $t_v^{\text{start}} \geq \max_{u \in \text{pred}(v)} t_u^{\text{finish}}$, where $\text{pred}(v)$ is the set of parent modules of module v.

3. **Deadline constraint:**

$$t_{ik}^{\text{finish}} \leq D_i \quad \forall T_i \in T \tag{7}$$

where D_i represents the task-level deadline for every task T_i. Equation (7) is applied to the last module of every task, since a task is complete only when its last module finishes.

4. **Mode selection constraint:**

$$\sum_m y_{im} = 1 \quad \forall M_i \tag{8}$$

which enforces that every module is executed in exactly one frequency-voltage mode, so a processor can only operate in a single frequency-voltage mode at a time. Equation (8) makes sure that exactly one mode is selected, and never zero or more than one.

5. **Energy consumption constraint:**

$$E_{total} = \sum_{i,j,m} E(M_i, P_j, S_{jm}) \cdot x_{ij} \cdot y_{im} \tag{9}$$

This ensures that the total energy is the sum of energy consumed by each module-processor-mode combination. In our model, each term of Eq. (9) is computed using Eq. (16) in Section 4.

6. **Reliability constraint:** The reliability of the entire system, say R_{system}, depends on all the assignments of the type processor-module:

$$R_{system} = \prod_{i,j,m} R(M_i, P_j, S_{jm})^{x_{ij} y_{im}} \tag{10}$$

and these assignments must satisfy $R_{system} \geq R_{min}$, where R_{min} is the reliability defined for the minimum reliability threshold. Each factor in Eq. (10) is computed using Eq. (17) in Section 4.

7. **Load balancing constraint:**

$$L_{\text{imbalance}} = \sqrt{\frac{1}{t} \sum_{j} \left(L_j - \bar{L}\right)^2} \tag{11}$$

where L_j is the total load on processor P_j and $\bar{L}$ is the average load, which controls the uneven workload distribution. The value of Eq. (11) becomes zero when all processors carry exactly the same load.

3.3 Decision Variables

1. **Assignment of module-processor pair:**

$$x_{ij} = \begin{cases} 1, & \text{if module } M_i \text{ (from any arbitrary task } T_q) \text{ is} \\ & \text{assigned to processor } P_j, \\ 0, & \text{otherwise.} \end{cases}$$

2. **Operating mode selection:**

$$y_{im} = \begin{cases} 1, & \text{if module } M_i \text{ is executed in mode } s_{jm}, \\ 0, & \text{otherwise.} \end{cases}$$

3.4 Objective Function

1. The goal primarily lies in being able to construct a mapping f and a schedule:

$$f : (M_i) \rightarrow (P_j, s_{jm})$$

for all the modules belonging to a task, say T_λ. f thus helps to determine the optimal execution ordering of the modules in a task T_λ by abiding with the dependency constraints on the order of execution of modules.

2. **Scheduling function:** Construct a set which encompasses all the global assignments F, acting as a scheduling function used to construct the actual schedule, say G. Define σ as:

$$\sigma : (F, \text{system state}) \mapsto \{(M_{in}, P_i, s_{it}, t_{in}^{\text{start}}, t_{in}^{\text{finish}})\}$$

σ takes the assignments and produces start/finish times for every module whilst respecting precedence, exclusivity, deadlines, and transition overheads.

3.5 Problem Characterization

The optimization landscape exhibits the following characteristics:

- The decision variables x_{ij} and y_{im} are *discrete and binary*, representing assignment and mode selection respectively.
- The objective function

$$\begin{aligned} F_{\text{obj}}(F, \sigma) = \alpha E_{\text{total}}(F, \sigma) + \beta\big(1 - R_{\text{system}}(F, \sigma)\big) \\ + \gamma L_{\text{imbalance}}(F, \sigma) \end{aligned} \tag{12}$$

tries to simultaneously minimize energy use, maximize reliability, and balance the load. The nature of these goals is conflicting and the function is non-linear, implying that the relationship between the decision variables (F, σ) cannot be characterized by a straight line. Equation (12) is the base objective; the deadline penalty is added to it in Eq. (18) and Eq. (19) of Section 4.

- The constraints help to characterize the factors of precedence, exclusivity, mode dependencies, delineated in (5)–(11) which make the search space irregular and highly combinatorial as the number of possible combinations grows exponentially with the number of modules.

3.6 Constraint Handling

The solutions which are not admissible due to them being not able to satisfy at least one of the constraint are handled through a penalty-based mechanism [45] wherein each infeasible solution is associated with an objective value $P_{penalty}$. Each constraint violation contributes to a penalty term $P_{penalty}$ which is added to the objective function (see (5)–(11)):

$$F'_{\text{obj}} = F_{\text{obj}} + \lambda P_{penalty} \tag{13}$$

where λ is a penalty coefficient, dynamic in nature. Making the penalty coefficient dynamic is important so that the algorithm does not disregard the infeasible solutions, closer to the set of feasible solutions early on in the process. λ can be made stronger later in the process when the search becomes narrower and focused. Equation (13) therefore shows how the penalty is added on top of the base objective of Eq. (12). In

the experiments reported in this paper, we keep λ fixed (Table 1), and the penalty consists of the deadline overrun term P_{dl} and the reliability floor term of Eq. (19).

3.7 Expected Outcome

The proposed approach aims to derive a near-optimal mapping-scheduling pair (F^*, σ^*) that:

$$(F^*, \sigma^*) = \arg\min_{(F,\sigma)} F_{\text{obj}}(F, \sigma) \tag{14}$$

adheres to all system constraints. The result achieves a trade-off between computational load, system reliability and temporal constraints, which implicitly makes the solution scalable and adjustive across heterogeneous architectures. Equation (14) states this goal in formal terms: among all possible mapping-scheduling pairs, we want the one with the lowest objective value.

4 System Model

We use a module level heterogeneous scheduling model and extend it in Section 8.2 to support arbitrary directed acyclic precedence. Its mathematical core is given below.

4.1 Processors and DVFS Modes

The system comprises N_p heterogeneous processors, each with base execution speed s_j, base power pw_j, and base fault rate λ_j^{base}. Each processor supports $L = 3$ DVFS modes $\{S_1, S_2, S_3\}$, characterized by a frequency multiplier f_m, voltage multiplier v_m, and fault rate multiplier φ_m. For module M_i assigned to processor P_j in mode S_m:

$$t_{ijm} = \frac{w_i}{s_j \cdot f_m} \tag{15}$$

$$e_{ijm} = pw_j \cdot v_m^2 \cdot f_m \cdot t_{ijm} \tag{16}$$

$$r_{ijm} = \exp\!\left(-\lambda_j^{\text{base}} \cdot \varphi_m \cdot t_{ijm}\right) \tag{17}$$

Here, Eq. (15) gives the execution time of the module, Eq. (16) gives the energy it uses, and Eq. (17) gives the probability that it runs without a fault. System reliability R_{sys} is the product of per module reliabilities across the batch. Faster/higher voltage modes reduce execution time but increase both power draw and fault rate.

Notation. To avoid confusion, $\beta(t)$, written with the iteration t, always means the QPSO contraction expansion coefficient (Sections 5 and 7), while β without an argument is the reliability weight in the objective. Similarly, λ is the deadline penalty weight while λ_j^{base} is a fault rate; K is the number of tasks in the synthetic benchmark while k is the number of modules in one task; and in Algorithms 1 and 2, S and T are the number of particles and iterations, not the mode set and the task set.

4.2 Objective Function

$$\begin{aligned} F'_{obj} = \alpha \cdot E_{total} + \beta \cdot (1 - R_{sys}) + \gamma \cdot L_{imb} \\ + \lambda \cdot P_{dl} + \mu \cdot \max(0,\, R_{min} - R_{sys}) \end{aligned} \tag{18}$$

Table 1 Objective weights used in all experiments

Symbol	Objective term	Value
α	Normalised energy, E_{total}/E_{norm}	0.4
β	Unreliability, $1 - R_{sys}$	0.4
γ	Load imbalance, L_{imb}	0.2
λ	Deadline penalty, P_{dl}	50
μ	Reliability floor penalty, $\max(0, R_{min} - R_{sys})$	50
E_{norm}	Energy normalisation constant	1000
R_{min}	Minimum system reliability	0.70

where E_{total} is total energy, L_{imb} is the standard deviation of per processor utilization (busy time over makespan), and P_{dl} is the sum of relative deadline overruns across all tasks. The last term is the reliability floor penalty, which is applied when R_{sys} falls below the minimum reliability R_{min} of Eq. (10). Equation (18) is the base objective of Eq. (12) with the deadline penalty term added. **The normalization of E_{total} before it enters the objective must be stated**, since with typical processor power/speed magnitudes, raw (unnormalized) energy can dominate the objective by an order of magnitude even when λ is deliberately set large to prioritize deadline compliance. We normalize energy by a fixed constant $E_{norm} = 1000$, chosen so $\alpha \cdot E_{total}$ sits on the same $O(1)$ scale as $\beta \cdot (1 - R_{sys})$ and $\gamma \cdot L_{imb}$, and report this explicitly so the objective is fully reproducible:

$$F'_{obj} = \alpha \cdot \frac{E_{total}}{E_{norm}} + \beta \cdot (1 - R_{sys}) + \gamma \cdot L_{imb} + \lambda \cdot P_{dl} + \mu \cdot \max(0,\, R_{min} - R_{sys}). \tag{19}$$

In all experiments the weights of Eq. (19) are fixed to the values in Table 1.

5 System Architecture

Fig. 1 shows how the proposed scheduler, ACMR-Soft, is built. It has two parts. The first is a *main loop* that searches for a good schedule. The second is a *feedback path* that checks how busy the processors are and uses this to gently guide the search.

5.1 Main Loop

The main loop has the following blocks.

- **Problem Instance.** This is the input: the workflow (a set of modules with precedence constraints), the heterogeneous processors, and the DVFS modes (Section 4).
- **Swarm State.** Each particle stores its current position x_i and its personal best p_i. The swarm also stores one global best g.
- **QPSO Move.** Every particle is moved to a new position using the QPSO update. The step size is controlled by $\beta(t)$, which is reduced from 1.0 to 0.4 during the run. This block also receives the small push (drift) from the feedback path.

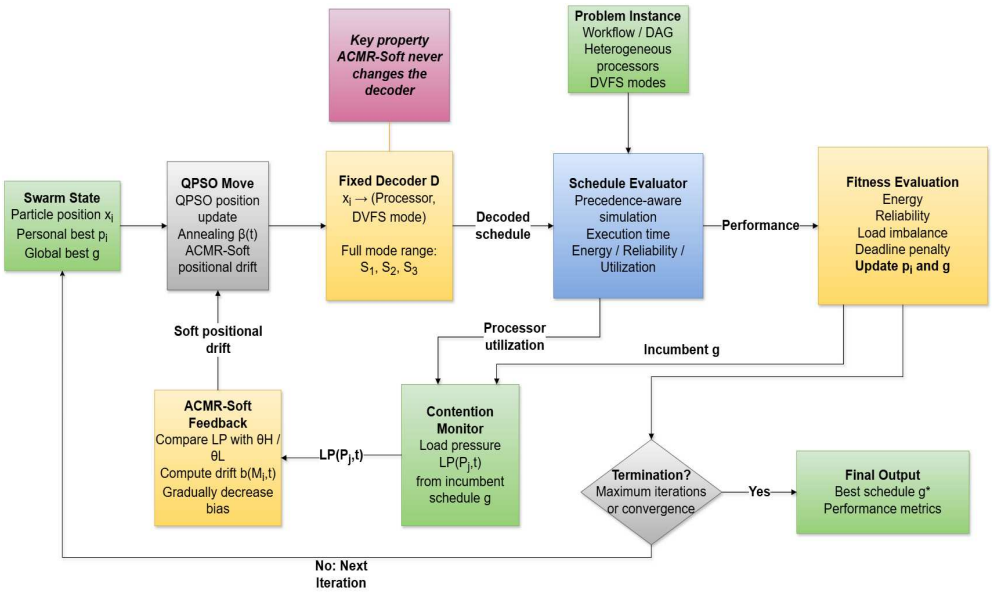

Fig. 1 System architecture of ACMR-Soft. Solid arrows show the main optimisation loop. Dashed arrows show the contention feedback. The decoder is never changed by the feedback.

- **Fixed Decoder $\mathcal{D}$.** The decoder turns a particle position into a real schedule: for every module it picks a processor and a DVFS mode. It always uses the full range of modes S_1, S_2, S_3 and never changes during the run.
- **Schedule Evaluator.** This block simulates the decoded schedule while respecting the precedence between modules. It gives the execution time, energy, reliability and processor utilization.
- **Fitness Evaluation.** The results are combined into the single score F'_{obj} (Eq. (19)). If a particle gets a better score than before, p_i is updated, and if it beats the whole swarm, g is updated.

5.2 Contention Feedback

The feedback path has two blocks.

- **Contention Monitor.** It takes the processor utilization from the evaluator and the current best schedule g. From these it computes the load pressure $LP(P_j, t)$ of every processor. A high value means that the processor is busier than the others.
- **ACMR-Soft Feedback.** It compares LP with the two thresholds θ_H and θ_L and decides a small bias $b(M_i, t)$ (Eq. (21)). This bias pushes the mode part of a particle position a little up or down the mode range (Eq. (22)). The push becomes weaker as the run goes on.

The feedback goes only to the QPSO Move block (dashed arrow in Fig. 1). It never goes to the decoder.

5.3 Why the Decoder Stays Fixed

A personal best p_i is only a position. What it means depends on the decoder. If the decoder changes during the run, an old p_i can turn into a different schedule, and the swarm can no longer trust what it has stored. This is the problem of the original ACMR, which removed modes from the decoder at every iteration (Section 6). In ACMR-Soft, the decoder is fixed, so stored positions always keep the same meaning (Section 7).

5.4 Stopping

After every iteration, the system checks the termination condition (maximum iterations or convergence). If it is not met, the loop starts again with the next iteration. If it is met, the system returns the best schedule g^* with its performance metrics. In our experiments, we use 30 particles and stop after 60 iterations (Section 8). All compared methods are scored with the same objective F'_{obj}.

6 Diagnosing the Manifold Reshaping Failure Mode

6.1 The Original Mechanism

ACMR computes, at each QPSO iteration, a load pressure score $LP(P_j, t) = U_j(t)/\bar{U}(t)$ from the incumbent (global best) schedule's per processor utilization, and restricts each module's DVFS mode search set accordingly:

$$W(M_i, t{+}1) = \begin{cases} \{S_1, S_2, S_3\} & \text{if } LP(P_j, t) \geq \theta_H \\ \{S_1, S_2\} & \text{if } \theta_L < LP(P_j, t) < \theta_H \\ \{S_2, S_3\} & \text{if } LP(P_j, t) \leq \theta_L \end{cases} \tag{20}$$

In Eq. (20), a busy processor keeps all three modes, a processor with medium load loses the slowest mode S_3, and a lightly loaded processor loses the fastest mode S_1. Algorithm 1 summarises ACMR step by step.

6.2 Empirical Failure to Deliver the Intended Benefit

Implementing this mechanism faithfully from its governing equations (Section 4) and validating it on 30 paired synthetic trials ($K = 12$ tasks, with the HEFT seeded initialization used for all QPSO variants; second row of Table 2), we found ACMR produces **significantly more** deadline violations than both unrestricted QPSO (+61.5%, Holm $p = 0.0003$) and a QPSO variant using a single, precomputed static manifold partition (+114.3%, Holm $p < 0.0001$); this is the opposite of the intended effect. We examined one possible alternative explanation:

- **Initialization and reshaping onset.** Seeding the swarm with a HEFT derived solution, delaying reshaping onset until iteration 15, and combining both, each reduced but did not eliminate the gap (Table 2).

Algorithm 1 ACMR (original): shrink the allowed modes

Require: workflow, processors and DVFS modes; number of particles S; number of iterations T; thresholds θ_H, θ_L
Ensure: best schedule g

Create S particles (the swarm). Make one of them from the HEFT schedule.
At the start, every module may use all three modes: $W(M) \leftarrow \{S_1, S_2, S_3\}$.
Turn each particle into a schedule, choosing each module's mode from $W(M)$. Compute its score F'_{obj} (lower is better). Set each personal best p_i and the global best g.
for $t = 1$ to T **do**
    Take the schedule of g. For each processor, compute the load pressure $LP(P_j, t)$: how busy it is compared with the others.
    for each module M (on processor P_j) **do**
        Key step: shrink the list of allowed modes $W(M)$.
        if $LP(P_j, t) \geq \theta_H$ (busy) **then**
            $W(M) \leftarrow \{S_1, S_2, S_3\}$
        else if $LP(P_j, t) \leq \theta_L$ (lightly loaded) **then**
            $W(M) \leftarrow \{S_2, S_3\}$
        else
            $W(M) \leftarrow \{S_1, S_2\}$
        end if
    end for
    for each particle i **do**
        Move the particle with the QPSO rule, using step size $\beta(t)$.
        Turn it into a schedule, choosing each module's mode only from $W(M)$. Compute its score.
        If the score is better than before, update p_i. If it is the best so far, update g.
    end for
    Stop early if the search has converged.
end for
return g and its performance metrics

Table 2 Recovery attempts (30 paired trials, $K = 12$, mean deadline violations)

Variant	Violations	vs. static partition
Original ACMR	3.83	+135% ($p < 0.0001$)
+ HEFT seeded init	3.50	+114% ($p < 0.0001$)
+ delayed reshaping ($t \geq 15$)	3.37	+106% ($p = 0.0016$)
+ both	3.20	+96% ($p = 0.0058$)
Static partition (reference)	1.63	n/a

6.3 Isolating the Cause: An Ablation

We hypothesized that the problem was not *which* processors were restricted, but the fact that the search space was being restricted *at all*, because restricting the manifold changes the particle decode function ($x_i \rightarrow$ (processor, mode)), and QPSO's mechanism depends on a particle's personal best position p_i retaining a stable meaning across iterations. To test this, we compared ACMR against an ablation that reshapes

the manifold every iteration exactly like ACMR, but assigns each module's restriction *uniformly at random* rather than from load pressure.

Finding: ACMR is statistically indistinguishable from random reshaping on both deadline violations (Holm $p = 1.00$) and composite objective (Holm $p = 1.00$), while using significantly more energy (+8.3%, $p = 0.0001$) and achieving significantly worse reliability (−0.9%, $p < 0.0001$) than the random ablation. This is the central diagnostic result of this paper: **the specific load pressure signal contributes no measurable benefit over generic random reshaping** in this mechanism's current form. The damage comes from reshaping the decode function itself, not from getting the reshaping "direction" wrong.

7 ACMR-Soft: A Redesigned Contention Feedback Mechanism

7.1 Design

ACMR-Soft never changes the decode function: the mode subvector is always decoded against the full three mode range, exactly as in unrestricted QPSO. Instead, contention feedback is introduced as a small adjustment to the particle position during the move step. The size of this adjustment gradually decreases over iterations:

$$b(M_i, t) = \begin{cases} +1 & \text{if } LP(P_j, t) \geq \theta_H \\ -1 & \text{if } LP(P_j, t) \leq \theta_L \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

$$x_i[\text{mode}] \leftarrow \text{clip}\big(x_i[\text{mode}] - b(M_i, t) \cdot \eta \cdot \beta(t),\ 0,\ 1-\epsilon\big) \tag{22}$$

where $\beta(t)$ is QPSO's own annealing schedule (already used for the contraction expansion coefficient) and η (`bias_strength`) is a new, tunable drift magnitude. In simple words, Eq. (21) decides the direction of the push for each module, and Eq. (22) applies this push to the mode value of the particle, keeping it inside the valid range. Because the decode function is untouched, a particle's personal best position always means the same thing across iterations; the swarm's memory is never invalidated. We also implemented and tested a second, more conservative candidate (restricting the hard manifold only for the single most congested processor's modules each iteration, leaving all others permanently unrestricted), which partially recovers performance, but less effectively than removing hard restriction altogether. On the same 30 trials and seeds as Table 3, this variant has 2.73 ± 1.26 deadline violations, while ACMR-Soft has 1.63 ± 1.22. The original ACMR has 3.50 ± 1.80, so this variant has 21.9% fewer violations than the original ACMR (uncorrected $p = 0.018$). Its other results are: energy 1254.1 ± 112.2, reliability 0.86 ± 0.01, load imbalance 0.10 ± 0.04 and F'_{obj} 42.9 ± 31.4. It is not clearly different from HEFT (−14.6%, Holm $p = 0.19$). It has more violations than plain QPSO (+26.2%, Holm $p = 0.048$), the static partition (+67.3%, Holm $p = 0.0003$) and GA (+90.7%, Holm $p = 0.0002$). The Holm correction is applied over

Algorithm 2 ACMR-Soft: keep all modes, only nudge the particle

Require: workflow, processors and DVFS modes; number of particles S; number of iterations T; thresholds θ_H, θ_L; drift strength η

Ensure: best schedule g

Create S particles (the swarm). Make one of them from the HEFT schedule.
Every module may always use all three modes $\{S_1, S_2, S_3\}$. The decoder $\mathcal{D}$ never changes.
Turn each particle into a schedule using $\mathcal{D}$. Compute its score F'_{obj} (lower is better). Set each personal best p_i and the global best g.
for $t = 1$ to T **do**
    Take the schedule of g. For each processor, compute the load pressure $LP(P_j, t)$: how busy it is compared with the others.
    for each particle i **do**
        Move the particle with the QPSO rule, using step size $\beta(t)$.
        for each module M (on processor P_j) **do**
            Key step: nudge the particle's mode value.
            Set $b \leftarrow +1$ if $LP(P_j, t) \geq \theta_H$ (busy), $b \leftarrow -1$ if $LP(P_j, t) \leq \theta_L$ (lightly loaded), else $b \leftarrow 0$.
            $x_i[\text{mode}] \leftarrow \text{clip}\big(x_i[\text{mode}] - b\,\eta\,\beta(t),\, 0,\, 1{-}\epsilon\big)$
        end for
        Turn the particle into a schedule using the same $\mathcal{D}$. Compute its score.
        If the score is better than before, update p_i. If it is the best so far, update g.
    end for
    Stop early if the search has converged.
end for
return g and its performance metrics

these four comparisons. Algorithm 2 summarises ACMR-Soft. Compared with Algorithm 1, only the key step differs: ACMR shrinks the list of allowed modes, whereas ACMR-Soft keeps all modes and only nudges the particle.

7.2 Hyperparameter Sensitivity

We swept $\eta \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.35\}$ on 15 trials each (seeds 42 to 56); $\eta = 0.10$ gave the lowest mean violations on these seeds (1.40 ± 1.20 vs. 2.07 ± 1.39 at $\eta = 0.15$). These seeds are part of the seeds used for Table 3, so $\eta = 0.10$ was chosen on development data. A repeat of the sweep on seeds 500 to 514 did not confirm it: it gave 2.47, 2.80, 2.40, 2.47, 1.73 and 2.07 mean violations for $\eta = 0.05$, 0.10, 0.15, 0.20, 0.25 and 0.35 (Friedman test across the six settings, $p = 0.015$), so $\eta = 0.10$ was not the best setting on these seeds. We therefore treat $\eta = 0.10$ as a value chosen on development data and not as an optimum, and report the results at this value on held-out seeds in Table 4. We suggest that any future use of this mechanism include this sweep as a standard part of the method, analogous to the θ_H/θ_L sensitivity analysis in Section 9.

8 Experimental Setup

8.1 Synthetic Benchmark

We use a synthetic benchmark: $K = 12$ tasks per trial, $n_k \sim U\{3, 6\}$ modules per task, module workload $w_i \sim U(8, 40)$ abstract cycles, 4 heterogeneous processors, 3 DVFS modes, deadline $D_k = 3.5 \sum_\ell w_{k,\ell}/(\bar{s} \cdot f_2)$. QPSO configuration: 30 particles, 60 iterations, β annealed $1.0 \to 0.4$. 30 paired trials, shared random seeds across algorithms per trial.

8.2 Real Workflow Benchmark and Model Generalization

The synthetic benchmark's task model restricts each task to a strict linear chain of modules, which is a real limitation, since actual scientific workflows are branching DAGs. We generalize the precedence constraint from "module i's only predecessor is module $i-1$" to "module i's ready time is the maximum finish time over *all* of its parents". This generalization includes the linear chain as a special case, where every module has exactly one parent. HEFT's upward rank recursion and the deadline formula are generalized correspondingly (rank via the standard DAG upward rank recursion over children; deadline via the DAG's critical path length rather than total chain length).

We evaluate on 60 real workflow execution traces spanning seven scientific applications (1000Genome, Cycles, Epigenomics, Montage, Seismology, SoyKB, sraSearch; 22 to 676 modules each), sourced from WfCommons/WfInstances [33], which provide real task graphs with real measured wall clock runtimes from Chameleon Cloud executions. Instances were selected to provide distinct module counts for each application (avoiding near duplicate replicate runs of the same nominal workflow size) and allocated as evenly as availability permits across the seven applications (1000Genome 12, Cycles 11, Epigenomics 11, Montage 6, Seismology 6, SoyKB 9, sraSearch 5). Since these traces report runtime in seconds on real machines of unknown specification rather than the model's abstract cycles, we apply one explicit conversion: measured runtime is treated as if produced at the model's mean processor speed in mode S_2, the same reference point already used by the synthetic deadline formula. Each trial schedules one real workflow instance across the four processor testbed (see Section 8.3 for why we did not batch multiple large workflows per trial).

8.3 A Methodological Note on Batching Real Workloads

An initial design scheduled several real workflows concurrently per trial, mirroring the synthetic benchmark's "K tasks at once" structure. This produced roughly 370 modules competing for 4 processors, which is contention far outside the range the four processor testbed was calibrated for (the synthetic benchmark totals roughly 54 modules across 12 tasks). Every algorithm missed every deadline by two orders of magnitude, and between algorithm differences were swamped by this saturation effect. We therefore schedule one real workflow instance per trial, which matches how workflow scheduling benchmarks of this kind are evaluated elsewhere in the literature and keeps the modules per processor ratio comparable to the synthetic design point.

Table 3 Six way comparison on the development seeds 42 to 71 (30 paired trials, $K = 12$, mean $\pm$ SD). The value $\eta = 0.10$ was chosen on seeds 42 to 56, so this table is not an independent test of ACMR-Soft (see Table 4).

Algorithm	Energy	Reliability	Load imbalance	Violations	F'_{obj}
HEFT	1091.4 ± 88.8	0.88 ± 0.01	0.02 ± 0.02	3.20 ± 1.22	52.0 ± 31.4
Baseline QPSO	1232.7 ± 117.8	0.86 ± 0.01	0.11 ± 0.06	2.17 ± 1.49	26.7 ± 25.4
Static partition	1306.3 ± 128.8	0.86 ± 0.01	0.10 ± 0.05	1.63 ± 1.49	16.2 ± 17.6
Original ACMR	1250.4 ± 107.7	0.86 ± 0.01	0.13 ± 0.04	3.50 ± 1.80	65.1 ± 57.5
ACMR-Soft (η=0.10)	1313.8 ± 114.1	0.86 ± 0.01	0.10 ± 0.05	$\mathbf{1.63 \pm 1.22}$	20.8 ± 22.1
GA	1260.7 ± 115.7	0.86 ± 0.01	0.12 ± 0.05	1.43 ± 1.45	14.2 ± 16.7

8.4 Algorithms Compared

HEFT [2]; unrestricted QPSO; QPSO with a static, precomputed manifold partition; the original (hard restriction) ACMR mechanism; ACMR-Soft (proposed); and a budget matched genetic algorithm (tournament selection, uniform crossover, Gaussian mutation, elitism= 1; population 30 $\times$ generations 60, matching QPSO's particle/iteration budget). All QPSO variants and the GA use a shared per trial random seed for paired comparison; all QPSO variants additionally use HEFT seeded swarm initialization, applied uniformly for fairness.

8.5 Statistical Protocol

All pairwise comparisons use paired t tests, with Wilcoxon signed rank tests [46] used as a nonparametric check, Holm correction [47] across the full family of tests at each experiment, and paired Cohen's d effect sizes. This protocol follows common practice for comparing swarm and evolutionary algorithms [48].

9 Results: Synthetic Benchmark

Table 3 gives the mean and standard deviation of every metric for all six algorithms on the synthetic benchmark (seeds 42 to 71). On these seeds, ACMR-Soft ties the static partition on deadline violations (mean 1.63 for both, $p = 1.00$) and is significantly better than HEFT (-49.0%, Holm $p = 0.0001$, $d = -0.89$) and unrestricted QPSO (-24.6%, Holm $p = 0.006$, $d = -0.62$). It is not significantly different from GA ($+14.0\%$, Holm $p = 0.98$). The original ACMR mechanism is significantly worse than every other algorithm except HEFT. Because $\eta = 0.10$ was chosen on seeds 42 to 56, these ACMR-Soft numbers are not an independent test.

We therefore ran all six algorithms again on 30 seeds that were not used for any tuning (seeds 500 to 529, same protocol, $\eta = 0.10$; Table 4). On these seeds, ACMR-Soft has fewer deadline violations than HEFT (-34.7%, Holm $p = 0.015$, $d = -0.55$), is not different from unrestricted QPSO ($+1.6\%$, Holm $p = 0.89$), and has more violations than the static partition ($+68.4\%$, Holm $p = 0.010$, $d = +0.61$); the difference from GA ($+28.0\%$, Holm $p = 0.17$) is not significant. So the advantage over unrestricted QPSO and the tie with the static partition seen on the development seeds do not hold on new seeds. The original ACMR is again worse than the static partition ($+118.4\%$, Holm $p = 0.0002$) and GA ($+66.0\%$, Holm $p = 0.003$), and is not significantly different from

Table 4 Six way comparison on held-out seeds 500 to 529 (30 paired trials, $K = 12$, $\eta = 0.10$, mean ± SD)

Algorithm	Energy	Reliability	Load imbalance	Violations	F'_{obj}
HEFT	1106.7 ± 116.1	0.88 ± 0.01	0.02 ± 0.01	3.27 ± 1.15	58.4 ± 33.5
Baseline QPSO	1245.1 ± 140.8	0.86 ± 0.01	0.10 ± 0.05	2.10 ± 1.54	33.0 ± 30.2
Static partition	1337.7 ± 180.3	0.86 ± 0.02	0.09 ± 0.05	$\mathbf{1.27 \pm 1.15}$	19.2 ± 22.9
Original ACMR	1297.6 ± 151.7	0.86 ± 0.02	0.14 ± 0.05	2.77 ± 1.86	58.9 ± 81.7
ACMR-Soft (η=0.10)	1289.9 ± 134.2	0.86 ± 0.01	0.09 ± 0.04	2.13 ± 1.56	37.4 ± 40.9
GA	1278.7 ± 154.4	0.86 ± 0.02	0.11 ± 0.05	1.67 ± 1.56	17.2 ± 21.6

Table 5 Real workflow comparison (60 real instances, mean ± SD)

Algorithm	Energy	Reliability	Load imbalance	Violations	F'_{obj}
HEFT	29714 ± 35624	0.31 ± 0.35	0.04 ± 0.07	0.73 ± 0.44	118.7 ± 100.8
Baseline QPSO	26667 ± 29252	0.31 ± 0.35	0.06 ± 0.10	0.72 ± 0.45	118.5 ± 103.0
Static partition	30083 ± 31891	0.29 ± 0.35	0.06 ± 0.10	0.72 ± 0.45	$\mathbf{104.0 \pm 89.0}$
Original ACMR	30866 ± 32748	0.29 ± 0.35	0.07 ± 0.10	0.70 ± 0.46	104.2 ± 88.8
ACMR-Soft	27128 ± 30144	$\mathbf{0.31 \pm 0.35}$	0.06 ± 0.09	0.72 ± 0.45	118.7 ± 102.9
GA	27975 ± 30551	0.30 ± 0.35	0.06 ± 0.10	0.73 ± 0.44	120.1 ± 106.9

unrestricted QPSO (+31.7%, Holm $p = 0.21$) or HEFT (−15.3%, Holm $p = 0.26$); the difference between ACMR-Soft and the original ACMR (−22.9%, uncorrected $p = 0.084$) is not significant at this sample size. Holm correction is applied over four comparisons for each of the two methods.

9.1 Threshold Sensitivity

We swept $\theta_H \in \{1.10, 1.25, 1.50\}$, $\theta_L \in \{0.50, 0.75, 0.90\}$ (15 trials per cell) for the original ACMR. Mean deadline violations ranged from 1.87 to 4.00 across the grid (a 114% relative spread), and the commonly assumed default ($\theta_H = 1.25$, $\theta_L = 0.75$) was not the best performing cell; the cell $\theta_H = 1.5$, $\theta_L = 0.5$ achieved 1.87 mean violations. In Tables 3 and 5, ACMR, ACMR-Soft and the static partition all use $\theta_H = 1.25$ and $\theta_L = 0.75$, and ACMR-Soft uses $\eta = 0.10$. The grid above was run with the original ACMR, which has no η. It did not use HEFT seeded initialization. It used seeds 300 to 314 (15 trials per cell, $K = 12$), which are different from the seeds 42 to 71 of Table 3. In this grid, the default cell gave 2.87 mean violations. The η sweep of Section 7 used seeds 42 to 56, which are part of the seeds of Table 3, so these are not different seeds. We also repeated the threshold sweep for ACMR-Soft with $\eta = 0.10$, HEFT seeded initialization and seeds 500 to 514 (15 trials per cell). The mean violations are as follows, with $\theta_H = 1.10$, 1.25, 1.50 as rows and $\theta_L = 0.50$, 0.75, 0.90 as columns: 1.47, 1.53, 2.13 for the first row; 2.07, 2.80, 3.40 for the second row; and 2.13, 3.00, 3.40 for the third row.

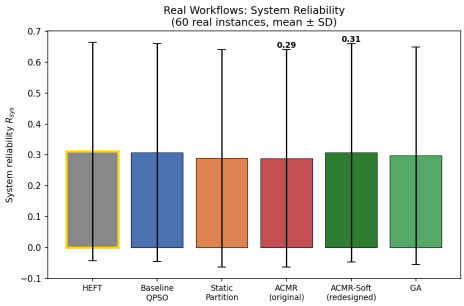

Fig. 2 System reliability of all six algorithms (60 real instances, mean ± SD). HEFT, baseline QPSO and ACMR-Soft have the highest mean reliability.

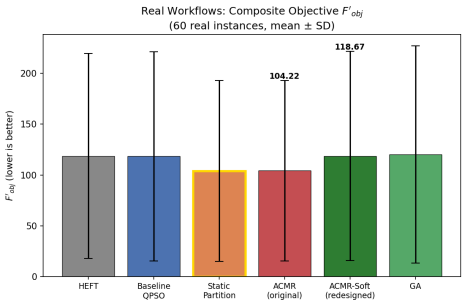

Fig. 3 Composite objective F'_{obj} of all six algorithms (60 real instances, mean ± SD). Lower is better. The static partition and the original ACMR have the lowest values. ACMR-Soft, baseline QPSO, HEFT and GA are clearly higher.

10 Results: Real Workflow Benchmark

We now test the methods on 60 real workflow instances. This sample is large enough to detect several statistically significant effects, which gives a clear picture of what ACMR-Soft gains and what it gives up. Table 5 gives the mean and standard deviation of every metric for all six algorithms on these instances.

Reliability. ACMR-Soft is significantly more reliable than the static partition (6.3% higher, Holm $p < 0.0001$, $d = +0.90$) and the original ACMR (6.4% higher, Holm $p < 0.0001$, $d = +1.02$), as shown in Fig. 2.

Energy and composite objective. First, ACMR-Soft uses significantly less energy than the static partition (9.8% less, Holm $p = 0.0008$, $|d| = 0.57$) and the original ACMR (12.1% less, Holm $p < 0.0001$, $|d| = 0.70$). Second, its composite objective is significantly worse than both (14.1% higher than the static partition and 13.9% higher than the original ACMR, both Holm $p < 0.0001$; Fig. 3).

This second result is *not* because ACMR-Soft misses more deadlines. All six algorithms miss deadlines at statistically the same rate (Friedman test across all six, $p = 0.54$, Table 9). Instead, when ACMR-Soft misses a deadline, it tends to miss it by more. The P_{dl} term in Eq. (19) penalizes how late each task is, not only how many tasks are late.

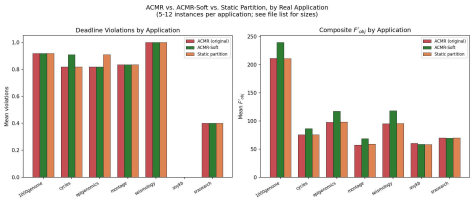

Fig. 4 ACMR-Soft, original ACMR and static partition, split by source application. The composite objective tradeoff (right panel) appears in most applications, not just one.

Results by application. Fig. 4 splits the results by the seven source applications. The pattern is similar across applications and is not caused by one unusual application. ACMR-Soft has a worse composite objective than the static partition and the original ACMR on 1000Genome, Cycles, Epigenomics, Montage and Seismology. Its reliability advantage is also spread across workflow types.

Summary. ACMR-Soft gives a real gain in reliability and energy, at the same deadline violation rate as every other method. In return, its composite objective is worse.

10.1 Where the Composite Objective Penalty Comes From

The weights in Eq. (19) are fixed (Table 1). This means we can split F'_{obj} exactly into three parts:

- the weighted base terms, $\alpha E_{total}/E_{norm} + \beta(1 - R_{sys}) + \gamma L_{imb}$;
- the reliability floor penalty, $\mu \cdot \max(0, R_{min} - R_{sys})$ with $\mu = 50$ and $R_{min} = 0.70$;
- the deadline overrun penalty, λP_{dl}.

Table 6 and Fig. 5 show the result. For every algorithm, the deadline penalty is by far the largest part, making up 66% to 72% of F'_{obj}. The base terms make up only 9% to 12%.

ACMR-Soft has a *lower* base term than the static partition and the original ACMR (11.14, compared with 12.33 and 12.64). This matches its better energy and reliability. However, its deadline penalty is about 17 units higher (85.62, compared with 69.01 and 68.86). So the whole gap in F'_{obj} between ACMR-Soft and these two methods comes from the deadline penalty.

Table 6 Parts of the composite objective F'_{obj} on real workflows (mean over 60 instances). Each row adds up to F'_{obj}.

Algorithm	Weighted base	Reliability penalty	Deadline penalty	F'_{obj}
HEFT	12.17	21.69	84.83	118.69
Baseline QPSO	10.96	21.88	85.62	118.46
Static partition	12.33	22.70	69.01	104.04
Original ACMR	12.64	22.71	68.86	104.22
ACMR-Soft	11.14	21.91	85.62	118.67
GA	11.48	22.33	86.32	120.13

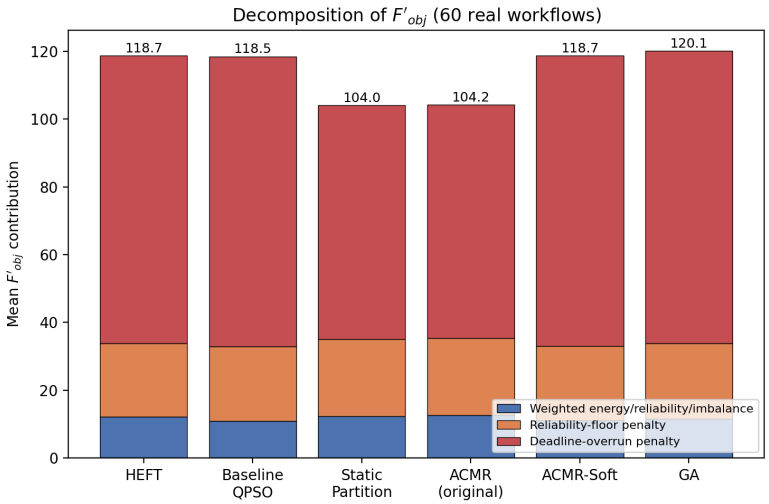

Fig. 5 The three parts of F'_{obj}: weighted base terms, reliability floor penalty and deadline overrun penalty (60 real instances, mean). The gap between ACMR-Soft and the static partition or original ACMR comes entirely from the deadline overrun penalty.

Table 7 and Fig. 6 look only at the instances that miss their deadline. All six algorithms miss about the same number of deadlines (42 to 44 out of 60). However, ACMR-Soft misses them by more. Its mean normalised overrun $(C - D)/D$ is 2.39, compared with 1.93 for the static partition and 1.97 for the original ACMR.

Across all 60 paired instances, the total normalised overrun P_{dl} of ACMR-Soft is 24.1% higher than that of the static partition and 24.3% higher than that of the original ACMR (both Holm $p < 0.0001$, $d = +1.01$). We apply the Holm correction to these five overrun tests as a separate family, so the main family of tests stays the same. ACMR-Soft never has a smaller overrun than the original ACMR: it ties on 17 instances and is worse on 43. In short, ACMR-Soft does not miss deadlines more often, but it misses them by a larger margin.

Table 7 Normalised deadline overrun $(C - D)/D$ on real workflow instances that miss their deadline.

Algorithm	Violating instances	Mean	Median	Max
HEFT	44	2.31	2.08	7.24
Baseline QPSO	43	2.39	2.20	7.35
Static partition	43	1.93	1.82	6.18
Original ACMR	42	1.97	1.83	6.18
ACMR-Soft	43	2.39	2.20	7.35
GA	44	2.35	2.06	7.60

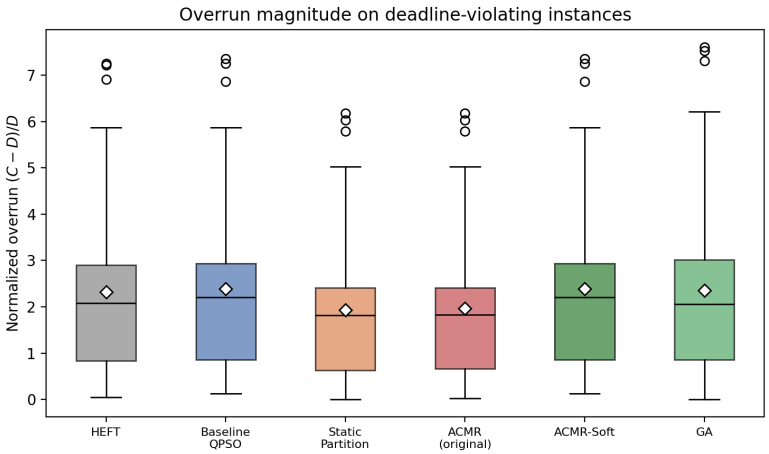

Fig. 6 Normalised overrun $(C - D)/D$ on instances that miss their deadline. The box shows the middle 50% of values, the line is the median and the diamond is the mean. The static partition and the original ACMR miss deadlines by smaller margins than the other four methods.

10.2 Per Instance Comparisons and Friedman Test

Table 8 and Fig. 7 count, for each metric, how many of the 60 instances ACMR-Soft wins, ties or loses against each baseline.

Against the static partition and the original ACMR, the energy and reliability gains of ACMR-Soft are not caused by a few large instances. ACMR-Soft uses less energy on 53 and 57 of the 60 instances, and it is more reliable on 53 and 56.

The same table also shows two limits.

- **ACMR-Soft is very close to baseline QPSO.** The two give the same schedule on 45 of the 60 instances and never differ in violations. None of their differences remain significant after the Holm correction. So on real workflows, the gains over the static partition and the original ACMR are shared with plain QPSO. This suggests that they come from keeping the decoder fixed, not from the contention drift at $\eta = 0.10$.
- **The energy gain over HEFT is not robust.** The mean energy of ACMR-Soft is 8.7% lower than that of HEFT, but ACMR-Soft uses more energy than HEFT on 39 of the 60 instances (Wilcoxon $p = 0.34$). The difference in the mean comes mostly from the SoyKB instances.

We also ran a Friedman test across all six algorithms (Table 9). The algorithms differ significantly on energy, reliability, load imbalance, F'_{obj} and normalised overrun (all $p < 10^{-9}$). They do not differ on deadline violations ($p = 0.54$).

The mean ranks show the same tradeoff. On energy and reliability, ACMR-Soft ranks third, behind HEFT and baseline QPSO and well ahead of the static partition and the original ACMR. On F'_{obj} and overrun, the static partition and the original ACMR rank best.

Table 8 Wins/ties/losses of ACMR-Soft against each baseline (60 paired real instances). For reliability, a win means a higher value. For all other metrics, a win means a lower value.

Metric	vs. HEFT	vs. Baseline	vs. Static	vs. ACMR	vs. GA
Energy	20/1/39	2/45/13	53/0/7	57/0/3	46/0/14
Reliability	19/2/39	2/45/13	53/0/7	56/0/4	49/0/11
Load imbalance	17/1/42	9/45/6	37/0/23	40/0/20	34/0/26
Violations	1/59/0	0/60/0	1/58/1	0/59/1	1/59/0
F'_{obj}	37/1/22	1/45/14	10/0/50	14/0/46	33/0/27
Normalised overrun	21/19/20	1/57/2	1/16/43	0/17/43	24/16/20

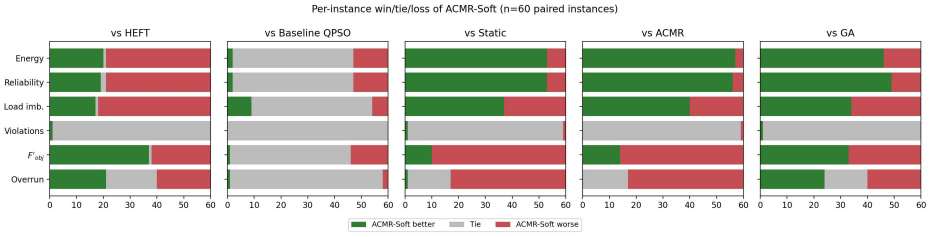

Fig. 7 Wins, ties and losses of ACMR-Soft against each baseline (60 paired real instances). Green means ACMR-Soft is better, grey means a tie and red means ACMR-Soft is worse. Against the static partition and the original ACMR, ACMR-Soft wins on energy and reliability on most instances but loses on F'_{obj} and overrun. Against baseline QPSO, most instances are ties.

Table 9 Friedman test across the six algorithms, with mean ranks (1 is best; 60 real instances).

Metric	p	HEFT	Baseline	Static	ACMR	ACMR-Soft	GA
Energy	1.3×10^{-30}	2.30	2.30	4.92	5.18	2.65	3.65
Reliability	2.1×10^{-30}	2.28	2.27	4.88	5.12	2.63	3.82
Load imbalance	2.1×10^{-10}	2.13	3.36	4.08	4.37	3.34	3.72
Violations	0.54	3.54	3.49	3.49	3.44	3.49	3.54
F'_{obj}	8.9×10^{-20}	4.55	3.62	2.00	2.42	4.04	4.37
Normalised overrun	2.0×10^{-19}	4.16	4.07	2.25	2.16	4.11	4.26

11 Discussion and Threats to Validity

What ACMR-Soft demonstrates. ACMR-Soft is not shown here to be uniformly the best algorithm: on held-out synthetic seeds it has fewer deadline violations than HEFT, is statistically indistinguishable from unrestricted QPSO and GA, and has more violations than the static partition, and on real workflows it trades a significantly worse composite objective for significantly better reliability and energy efficiency. What it demonstrates is that the underlying idea (using the optimiser's own incumbent schedule as a contention signal) can be made to work once the specific failure mode (decode function reinterpretation) is corrected, whereas the original hard restriction mechanism does not show the intended benefit under the evaluated conditions; and that once corrected, it provides a measurable tradeoff rather than improvement

or decline across all metrics. The importance of these tradeoffs depends on the application; e.g., a reliability critical system may reasonably accept ACMR-Soft's larger occasional overruns for its lower fault rate and energy draw. This is not something this paper claims to resolve.

Synthetic model limitations. The synthetic benchmark's four processor, three mode configuration and its specific workload distribution are a narrow slice of possible heterogeneous platforms; results should be read as relative comparisons under identical conditions, not as absolute claims about real deployed systems.

Sample size. WfInstances contains substantially more traces than the 60 used here; further scaling is natural future work, particularly for the smaller per application effect sizes in Fig. 4.

Energy normalization. The constant used to bring energy onto a comparable scale with the other objective terms (Section 4) is a documented modeling choice, not a value derived from first principles; we suggest it be treated as a sensitivity parameter in follow up work, the same way θ_H/θ_L and η are treated here.

Hyperparameter η. ACMR-Soft introduces one new tunable parameter not present in the original mechanism. We report its sensitivity (Section 7) but a fuller characterization (e.g. adaptive annealing of η itself) is left to future work.

12 Conclusion

We propose ACMR, a contention aware search space reshaping mechanism for QPSO based heterogeneous scheduling, and show that in its original hard restriction form it does not deliver its intended deadline violation reduction, and, more importantly, why: hard restriction of the search space changes the meaning of a particle's personal best position, making the stored information unreliable regardless of whether the restriction is driven by a meaningful contention signal or applied at random. We redesign the mechanism (ACMR-Soft) to inject the same contention signal without ever altering the decode function, which on held-out synthetic data gives fewer deadline violations than HEFT and matches unrestricted QPSO (the static partition remains better) and, on 60 real scientific workflow traces, showing a consistent reliability improvement and an energy efficiency advantage over both the static partition and the original mechanism, alongside a significantly worse composite objective outcome driven by larger (not more frequent) deadline overruns. On real workflows, these gains are shared with unrestricted QPSO (Section 10.2), which suggests that they mainly come from keeping the decoder fixed. Along the way we generalize the underlying task model from linear chains to arbitrary DAG precedence, add a genetic algorithm baseline, and report a threshold sensitivity analysis. We view the diagnostic finding (that changing *what a position means* during optimization is less stable than changing *how positions are searched*) as a potentially generalizable design issue for other swarm intelligence scheduling mechanisms built on similar reshaping ideas, and suggest the ablation technique used here (reshaping at all vs. informed reshaping) as a standard diagnostic for future proposals in this space.

Data Availability

This study did not generate new workflow execution traces. All workflow traces used in the experiments are publicly available from the WfCommons project (WfInstances repository) at https://github.com/wfcommons/WfInstances. We selected 60 real workflow instances covering seven application types (1000Genome, Cycles, Epigenomics, Montage, Seismology, SoyKB, and sraSearch), with task runtimes measured on Chameleon Cloud.

The data generated in this study are publicly available at https://github.com/avhankardhruv06-svg/Adaptive_Contention_Aware_Manifold_Reshaping_-ACMR-. The repository contains:

- the list of the 60 selected WfInstances files and the script used to select them;
- the raw per-instance results for all six evaluated schedulers (HEFT, GA, baseline QPSO, static partitioning, the original ACMR, and ACMR-Soft);
- the source code of the Python simulator and all schedulers; and
- the analysis script that computes the paired statistical tests (paired t-tests, Wilcoxon signed-rank tests, Holm–Bonferroni correction, and Cohen's d) from the raw results.

An archived snapshot of this repository is available on Zenodo at https://doi.org/10.5281/zenodo.23717199.

Code availability

The simulator, all scheduling algorithms and the scripts used to produce the results and figures are available at https://github.com/avhankardhruv06-svg/Adaptive_Contention_Aware_Manifold_Reshaping_-ACMR-.

References

[1] Garey, M. R. & Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman, New York, NY, USA, 1979).

[2] Topcuoglu, H., Hariri, S. & Wu, M.-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* **13**, 260–274 (2002).

[3] Kennedy, J. & Eberhart, R. *Particle swarm optimization.* in *Proceedings of the International Conference on Neural Networks (ICNN)*, 1942–1948 (1995).

[4] Sun, J., Xu, W. & Feng, B. *A global search strategy of quantum-behaved particle swarm optimization.* in *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, 111–116 (2004).

[5] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D. & Freund, R. F. *Dynamic matching and scheduling of a class of independent tasks onto heterogeneous*

computing systems. in *Proceedings of the Heterogeneous Computing Workshop (HCW)*, 30–44 (1999).

[6] Hou, E. S. H., Ansari, N. & Ren, H. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems* **5**, 113–120 (1994).

[7] Zhang, W., Xie, H., Cao, B. & Cheng, A. M. K. Energy-aware real-time task scheduling for heterogeneous multiprocessors with particle swarm optimization algorithm. *Mathematical Problems in Engineering* **2014**, 287475 (2014).

[8] Moulik, S., Das, Z., Devaraj, R. & Chakraborty, S. SEAMERS: A semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems. *Journal of Systems Architecture* **114**, 101953 (2021).

[9] Arabnejad, H. & Barbosa, J. G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems* **25**, 682–694 (2014).

[10] Kwok, Y.-K. & Ahmad, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* **31**, 406–471 (1999).

[11] Weiser, M., Welch, B., Demers, A. & Shenker, S. *Scheduling for reduced CPU energy.* in *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 13–23 (1994).

[12] Yao, F., Demers, A. & Shenker, S. *A scheduling model for reduced CPU energy.* in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, 374–382 (1995).

[13] Zhu, D., Melhem, R. & Childers, B. R. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems* **14**, 686–700 (2003).

[14] Lee, Y. C. & Zomaya, A. Y. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems* **22**, 1374–1381 (2011).

[15] Assayad, I., Girault, A. & Kalla, H. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems: The TSH tricriteria scheduling heuristic. *International Journal on Software Tools for Technology Transfer* **15**, 229–245 (2013). Preprint available as HAL hal-00923926.

[16] Das, A., Singh, A. K. & Kumar, A. Execution trace-driven energy-reliability optimization for multimedia MPSoCs. *ACM Transactions on Reconfigurable Technology and Systems* **8**, 18:1–18:19 (2015).

[17] Zhu, D., Melhem, R. & Mossé, D. *The effects of energy management on reliability in real-time embedded systems.* in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 35–40 (2004).

[18] Dongarra, J. J., Jeannot, E., Saule, E. & Shi, Z. *Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems.* in *Proceedings of the 19th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 280–288 (2007).

[19] Halimu, Y., Zhou, C., You, Q. & Sun, J. A quantum-behaved particle swarm optimization algorithm on Riemannian manifolds. *Mathematics* **10**, 4168 (2022).

[20] Chen, W.-N. *et al.* A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation* **14**, 278–300 (2010).

[21] Karafotias, G., Hoogendoorn, M. & Eiben, A. E. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation* **19**, 167–187 (2015).

[22] Mallipeddi, R., Suganthan, P. N., Pan, Q.-K. & Tasgetiren, M. F. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing* **11**, 1679–1696 (2011).

[23] Eiben, A. E., Hinterding, R. & Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **3**, 124–141 (1999).

[24] Sun, J., Fang, W., Wu, X., Palade, V. & Xu, W. Quantum-behaved particle swarm optimization: Analysis of individual particle behavior and parameter selection. *Evolutionary Computation* **20**, 349–393 (2012).

[25] Clerc, M. & Kennedy, J. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* **6**, 58–73 (2002).

[26] Bean, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing* **6**, 154–160 (1994).

[27] Gonçalves, J. F. & Resende, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* **17**, 487–525 (2011).

[28] Hu, X. & Eberhart, R. C. *Adaptive particle swarm optimization: Detection and response to dynamic systems.* in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC)*, 1666–1670 (2002).

[29] Blackwell, T. & Branke, J. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation* **10**,

459–472 (2006).

[30] Jin, Y. & Branke, J. Evolutionary optimization in uncertain environments—a survey. *IEEE Transactions on Evolutionary Computation* **9**, 303–317 (2005).

[31] Juve, G. *et al.* Characterizing and profiling scientific workflows. *Future Generation Computer Systems* **29**, 682–692 (2013).

[32] Coleman, T. *et al.* WfCommons: A framework for enabling scientific workflow research and development. *Future Generation Computer Systems* **128**, 16–27 (2022).

[33] WfCommons Project. WfInstances: Real workflow execution traces. https://github.com/wfcommons/WfInstances. Workflow execution instances from Pegasus WMS on Chameleon Cloud.

[34] Pandey, S., Wu, L., Guru, S. M. & Buyya, R. *A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments.* in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 400–407 (2010).

[35] Rodriguez, M. A. & Buyya, R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing* **2**, 222–235 (2014).

[36] Moulik, S. & Sharma, Y. FRESH: Fault-tolerant real-time scheduler for heterogeneous multiprocessor platforms. *Future Generation Computer Systems* **161**, 214–225 (2024).

[37] Chakraborty, S., Sharma, Y. & Moulik, S. TREAFET: Temperature-aware real-time task scheduling for FinFET based multicores. *ACM Transactions on Embedded Computing Systems* **23**, 1–31 (2024).

[38] She, R., Wu, Y. & Cui, E. Energy-aware scheduling for reliability-oriented real-time parallel applications allocation on heterogeneous computing systems. *Future Generation Computer Systems* **168** (2025).

[39] Liu, Y., Qu, H., Chen, S. *et al.* Energy efficient task scheduling for heterogeneous multicore processors in edge computing. *Scientific Reports* **15**, 11819 (2025).

[40] Ahmad, W. *et al.* Energy-constrained workflow scheduling for DVFS-enabled heterogeneous systems in cloud computing environments. *Scientific Reports* **16**, 19452 (2026).

[41] Hosseini Shirvani, M. A survey study on task scheduling schemes for workflow executions in cloud computing environment: classification and challenges. *The Journal of Supercomputing* **80**, 9384–9437 (2024).

[42] Namdev, K., Rajak, R., Sajid, M. & Rajak, N. The role of AI-driven technologies in cloud workflow scheduling: a structured review. *Journal of Ambient Intelligence and Humanized Computing* **17**, 19–31 (2026).

[43] Zhu, K., Zhang, Z., Zeadally, S. & Sun, F. Learning to optimize workflow scheduling for an edge-cloud computing environment. *IEEE Transactions on Cloud Computing* (2024).

[44] Mahjoub, A., Khalilian, M. & Mohammadzadeh, J. QQLAOA: task scheduling with multi-objectives quantum mutation and Q-learning based arithmetic optimizer algorithm in cloud data centers. *Computing* **107**, 109 (2025).

[45] Coello Coello, C. A. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering* **191**, 1245–1287 (2002).

[46] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin* **1**, 80–83 (1945).

[47] Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics* **6**, 65–70 (1979).

[48] Derrac, J., García, S., Molina, D. & Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation* **1**, 3–18 (2011).

Funding

This research received no grant from any funding agency.

Author contributions

Dhruv Avhankar: Original draft, Methodology, Formal Analysis and Visualization. Swagata Paul: Methodology, Formal Analysis, Conceptualization. Priyansh Mudela: Software: Wrote code and ran the experiments, Visualization. M.A. Berlin: Conceptualization, Investigation, Supervision. All authors reviewed the manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Figure legends

Figure 1. System architecture of ACMR-Soft. The main loop (solid arrows) takes the problem instance and the swarm state, moves the particles with the QPSO update, turns each particle into a schedule with a fixed decoder, evaluates the schedule, scores it with the composite objective and checks the stopping rule. The contention feedback (dashed arrows) computes the load pressure from the current best schedule and sends a small push to the QPSO move block. The feedback never changes the decoder.

Figure 2. System reliability of the six algorithms on 60 real workflow instances. Bars show the mean and error bars show one standard deviation (SD). Higher is better. The outlined bar marks the highest mean, and the numbers above the ACMR and ACMR-Soft bars are their mean values.

Figure 3. Composite objective F'_{obj} of the six algorithms on 60 real workflow instances. Bars show the mean and error bars show one SD. Lower is better. The outlined bar marks the lowest mean, and the numbers above the ACMR and ACMR-Soft bars are their mean values.

Figure 4. Results of the original ACMR, ACMR-Soft and the static partition for each of the seven source applications (5 to 12 instances per application). The left panel shows the mean deadline violations, which is the share of instances that miss their deadline. The right panel shows the mean composite objective F'_{obj}. Along the horizontal axis the applications are 1000Genome, Cycles, Epigenomics, Montage, Seismology, SoyKB and sraSearch.

Figure 5. Parts of the composite objective F'_{obj} on 60 real workflow instances. Each bar is the mean over the instances and is split into the weighted base terms, the reliability floor penalty and the deadline overrun penalty (see key). The number above each bar is the mean F'_{obj}.

Figure 6. Normalised overrun $(C-D)/D$ of the instances that miss their deadline, where C is the completion time and D is the deadline. Each box shows the middle 50% of values, the line is the median and the diamond is the mean. Points beyond the whiskers are plotted one by one. The number of instances is 44, 43, 43, 42, 43 and 44 for HEFT, baseline QPSO, static partition, original ACMR, ACMR-Soft and GA.

Figure 7. Per instance comparison of ACMR-Soft with each baseline on 60 paired real workflow instances. Each bar shows how many instances ACMR-Soft is better, tied or worse (see key), for energy, reliability, load imbalance, deadline violations, composite objective and normalised overrun. For reliability a higher value is better. For all other metrics a lower value is better. The panels compare ACMR-Soft with HEFT, baseline QPSO, the static partition, the original ACMR and GA.

Table legends

Table 1. Objective weights and constants used in all experiments. The weights α, β, γ, λ and μ multiply the normalised energy, the unreliability, the load imbalance, the deadline overrun and the reliability floor penalty in the composite objective. E_{norm} scales the energy and R_{min} is the minimum system reliability. The same values are used for every algorithm.

Table 2. Recovery attempts for the original ACMR. The table gives the mean number of tasks that miss their deadline (out of 12) over 30 paired synthetic trials for the original ACMR, with HEFT seeded initialisation, with delayed reshaping and with both. The last column gives the difference from the static partition and the p value from a paired t test with Holm correction. Lower is better.

Table 3. Six way comparison of the algorithms on the synthetic benchmark (development seeds 42 to 71). Values are mean ± standard deviation (SD) over 30 paired trials with $K = 12$ tasks. Energy is in model units, reliability is the probability of no fault, load imbalance is the standard deviation of the processor utilisation, violations is the number of tasks (out of 12) that miss their deadline and F'_{obj} is the composite objective. Lower is better, except for reliability. ACMR-Soft uses $\eta = 0.10$, which was chosen on seeds 42 to 56, so this table is not an independent test of ACMR-Soft. Bold marks ACMR-Soft.

Table 4. Six way comparison of the algorithms on held-out synthetic seeds 500 to 529, which were not used for any tuning ($\eta = 0.10$). Values are mean ± SD over 30 paired trials with $K = 12$ tasks, with the same metrics and units as Table 3. Lower is better, except for reliability. Bold marks ACMR-Soft and the lowest mean violations.

Table 5. Comparison of the algorithms on 60 real workflow instances. Values are mean ± SD over the instances, and each trial schedules one workflow. Energy is in model units, and violations is the share of instances that miss their deadline. The other metrics are as in Table 3. Lower is better, except for reliability. Bold marks the lowest mean composite objective and the ACMR-Soft reliability that are discussed in the text.

Table 6. Parts of the composite objective F'_{obj} on the 60 real workflow instances (mean over the instances). The weighted base is $\alpha E_{total}/E_{norm} + \beta(1 - R_{sys}) + \gamma L_{imb}$, the reliability penalty is $\mu \max(0, R_{min} - R_{sys})$ and the deadline penalty is λP_{dl}. Each row adds up to its F'_{obj} value.

Table 7. Normalised deadline overrun $(C - D)/D$ on the real workflow instances that miss their deadline, where C is the completion time and D is the deadline. For each algorithm the table gives the number of violating instances (out of 60) and the mean, median and maximum overrun.

Table 8. Wins, ties and losses of ACMR-Soft against each baseline on the 60 paired real workflow instances, written as wins/ties/losses. For reliability a win means a higher value. For all other metrics a win means a lower value. A tie means that the two values are equal.

Table 9. Friedman test across the six algorithms on the 60 real workflow instances. The column p is the p value of the Friedman test for each metric. The other columns give the mean rank of each algorithm, where 1 is the best (the lowest value, or the highest value for reliability).