International Journal of Science and Engineering,2026;3(2): 01-13
Doi: 10.62304/ijse.v3i02.256

Multi-Objective Cloud Resource Optimization with NSGA-III: Design, Deployment, and Open Problems

Mohammad Nowsher Ali[1]
[1] *Department of Computer Science, Maharishi International University, Fairfield, Iowa, USA*
[1] *ORCID: https://orcid.org/0009-0009-8575-1551*
Corresponding Author: mohammad.ali@miu.edu

Keywords

NSGA-III
Multi-objective optimization
Cloud computing
Resource scheduling
Pareto optimality
Evolutionary algorithms
energy efficiency
SaaS
Public deployment
r=Research agenda

Article Information

***Received**: 04, June, 2026*
***Accepted**: 03, July, 2026*
***Published**: 07, July, 2026*

Doi: 10.62304/ijse.v3i02.256

ABSTRACT

Cloud computing has become the backbone of modern digital infrastructure, yet the simultaneous optimization of competing operational objectives, makespan, monetary cost, and energy consumption remains an open and practically significant research challenge. Existing commercial schedulers and the majority of academic proposals address only one or two of these objectives, discarding trade-off information that is essential for heterogeneous operator preferences. This paper presents a conceptual framework for applying the Non-dominated Sorting Genetic Algorithm III (NSGA-III) to the multi-objective cloud resource scheduling problem, with the explicit goal of informing the design of a publicly deployable optimization service. Drawing on established theoretical properties of NSGA-III, published workload characterization studies, and cloud service architecture literature, we identify the core design decisions that a realization of such a framework must resolve: chromosome encoding for workflow-structured workloads, cloud-topology-aware genetic operators, scalability management under real-time SLA constraints, and Pareto-front visualization for non-expert decision-makers. We further map four concrete implementation pathways, open-source library, cloud-provider plugin, SaaS scheduling API, and embedded data-center advisors and analyze each against criteria of deplorability, latency, governance, and reproducibility. It synthesizes existing knowledge to establish the conceptual and architectural foundations upon which future empirical work can be built and articulates a research agenda of seven open problems that must be solved before NSGA-III-based multi-objective cloud scheduling can reach production at scale.

1 Introduction

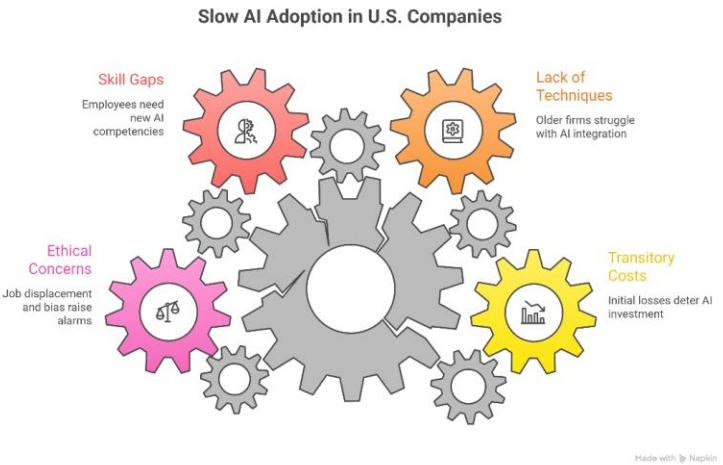
Cloud computing, broadly defined as the delivery of on-demand computing resources over a network with pay-as-you-go pricing [1, 2], has transformed how organizations provision and consume computational capacity. Infrastructure-as-a-Service (IaaS) offerings from providers such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform expose vast pools of heterogeneous virtual machines (VMs), enabling users to scale workloads from a single instance to tens of thousands within minutes. Yet this elasticity introduces a non-trivial allocation challenge: which VMs should execute which tasks, in which order, and for how long.

The challenge is, at its core, a multi-objective combinatorial optimization problem. Cloud users and operators typically pursue at least three competing goals simultaneously. First, they wish to minimize the total time required to complete a workload, the makespan, to satisfy response-time or deadline commitments. Second, they wish to minimize monetary expenditure, since VM instances accrue cost every hour they run. Third, especially in corporate sustainability and data-center efficiency contexts, they wish to minimize the energy consumed by the underlying physical hardware. These three objectives are structurally in conflict: provisioning more VMs in parallel reduces makespan but increases both cost and, often, energy; consolidating tasks onto fewer VMs reduces cost but extends makespan. No single schedule simultaneously minimizes all three.

Commercial cloud schedulers, auto-scaling groups, container orchestrators, and managed workflow services typically collapse these dimensions into a single weighted utility score, chosen by the platform vendor rather than the operator. This approach is computationally convenient but theoretically unsatisfying: it discards the rich trade-off structure of the problem and forces a preference encoding that may not align with any individual user's actual priorities. Research schedulers have addressed bi-objective variants [3, 4], but few target all three dimensions simultaneously, and fewer still have been designed with public, multi-tenant deployment in mind.

Evolutionary multi-objective optimization algorithms (EMOAs) offer a principled alternative. By evolving a population of candidate schedules simultaneously, they can approximate the Pareto-optimal front, the set of schedules for which no objective can be improved without degrading at least one other in a single optimization run. The decision-maker then selects from this front rather than committing to a single aggregation weight in advance. Among EMOAs, the Non-dominated Sorting Genetic Algorithm III (NSGA-III) [5], an extension of the widely used NSGA-II [6], is theoretically well-suited to this setting: its reference-point-based diversity mechanism maintains uniform

Figure 1: Slow AI Adoption in US Companies



coverage of the Pareto front even when three or more objectives are optimized simultaneously, a property that crowding-distance-based algorithms such as NSGA-II do not guarantee.

Despite these theoretical advantages, NSGA-III has not yet been realized as a general-purpose, publicly accessible cloud scheduling service. The gap between algorithmic proposal and deployed tool is substantial and involves decisions that have received limited attention in the literature: how to encode real workflow structures, how to calibrate operators to cloud API constraints, how to meet the latency requirements of online scheduling, and how to present Pareto-front outputs to non-specialist cloud operators. This paper is a conceptual contribution aimed at closing that gap. It makes no claim to original experimental results; instead, it synthesizes the relevant literature to establish the design space, identifies the key unresolved questions, and proposes four concrete implementation pathways through which an NSGA-III-based multi-objective cloud scheduler could be made available for public use.

The remainder of the paper is organized as follows. Section 2 reviews the relevant background in cloud scheduling and multi-objective evolutionary computation. Section 3 formalizes the multi-objective cloud scheduling problem as a conceptual target. Section 4 analyses NSGA-III's suitability for this target. Section 5 proposes the architectural design of a public-facing framework. Section 6 maps four deployment pathways. Section 7 identifies the open research challenges that must be solved before such a service can operate at production scale. Section 8 concludes.

2 Background and Related Work

2.1 2.1 Cloud Computing and the Scheduling Problem

The canonical definition of cloud computing, established by the United States National Institute of Standards and Technology (NIST), characterizes it as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [1]. This definition encompasses five essential characteristics, on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service that together distinguish cloud computing from conventionally dedicated hosting.

Within this model, the resource scheduling problem takes different forms depending on the service abstraction. At the IaaS layer, a scheduler assigns tasks or jobs to VM instances, each with a type (CPU count, memory, network bandwidth) and an associated hourly cost. At the Platform-as-a-Service (PaaS) layer, the abstraction rises to functions or containers, where the scheduler additionally controls instance lifecycle (cold start, warm pool, scale-to-zero). In all cases, the fundamental trade-off between performance and frugality persists.

Early scheduling research adapted classical parallel-processing heuristics to the cloud setting. The Heterogeneous Earliest Finish Time (HEFT) algorithm [7], designed for heterogeneous multiprocessor systems, ranks tasks by an upward rank metric and greedily assigns each to the processor that yields the earliest finish time. HEFT is computationally lightweight and produces good makespan results on single-objective benchmarks, but it has no mechanism for reasoning about cost or energy. Min-Min and Max-Min heuristics follow a similar philosophy, assigning the task with the smallest or largest estimated completion time first, and suffer the same structural limitation.

The recognition that cloud scheduling is inherently multi-objective led to a strand of research applying genetic algorithms and other population-based methods to bi-objective variants [3, 4]. Zhu et al. [3] applied NSGA-II to the joint minimization of makespan and cost for scientific workflows, demonstrating Pareto-front approximations that dominate single-objective solutions. Durillo and Prodan [4] extended this to Amazon EC2 spot instances, introducing cost uncertainty into the problem formulation. These contributions establish the value of multi-objective approaches but are limited to two objectives and were not designed for public deployment.

2.2 Multi-Objective Evolutionary Algorithms

Multi-objective optimization algorithms can be broadly categorized into three paradigms: scalarization approaches, which convert a multi-objective problem into one or more scalar problems via aggregation; indicator-based approaches, which guide search using quality indicators such as hypervolume; and dominance-based approaches, which maintain a population of non-dominated solutions [8]. For cloud scheduling, dominance-based EMOAs are the most practical choice, because they produce a set of trade-off solutions in a single run and impose no prior commitment to relative objective weights.

NSGA-II [6] is the most widely cited dominance-based EMOA. It uses fast non-dominated sorting to partition the

population into Pareto fronts and crowding distance to preserve diversity within each front. Its $O(MN^2)$ per-generation complexity (for M objectives and population size N) and strong empirical performance on two-objective problems made it the de facto standard for bi-objective cloud scheduling research. However, crowding distance becomes an unreliable diversity metric as the number of objectives grows beyond two: it measures distance in the objective space but does not guarantee uniform coverage of the front's geometry [5].

NSGA-III [5] was introduced specifically to address this limitation. Instead of crowding distance, it uses structured reference points, uniformly distributed on a unit hyperplane, to guide selection. Each individual in the population is associated with the nearest reference point; when the next Pareto front cannot be fully accommodated, individuals closest to the least-crowded reference points are selected preferentially. This mechanism maintains well-distributed solutions across high-dimensional Pareto fronts, making NSGA-III substantially better suited than NSGA-II for problems with three or more objectives.

MOEA/D [9] provides an alternative approach by decomposing the multi-objective problem into a set of scalar subproblems using weight vectors, then solving them simultaneously via neighborhood-based updates. MOEA/D can be computationally efficient but requires careful weight-vector design, and its performance can degrade when the Pareto front is non-convex or has irregular geometry, a characteristic commonly observed in cloud scheduling problems where cost and energy trade-offs are near-linear while makespan and cost trade-offs are convex [3].

2.3 The Gap: From Algorithm to Service

Despite an active research literature on multi-objective cloud scheduling, a publicly accessible, general-purpose scheduling service based on any EMOA does not yet exist. The absence of such a service reflects several under-studied engineering challenges. First, most research prototypes are implemented in MATLAB or Python simulation frameworks and depend on synthetic workload models rather than real cloud APIs. Second, the optimization runs described in the literature typically consume few seconds to several minutes, which is incompatible with the sub-second response times expected by cloud management planes. Third, Pareto-front outputs set of dozens of non-dominated schedules that are difficult for non-specialist operators to interpret without decision-support tooling. This paper addresses each of these gaps at the conceptual level.

3 The Multi-Objective Cloud Scheduling Problem

3.1 System Model

We conceptualize the cloud resource pool as a set V of heterogeneous VM instances, each characterized by a CPU capacity ω (in MIPS), memory μ (in GB), and a per-unit-time monetary cost c (in USD per hour). In practice, these parameters are obtained from the cloud provider's pricing API and are time-varying for spot or preemptible instances. The workload is represented as a Directed Acyclic Graph G = (T, E), where T is the set of tasks and E encodes precedence dependencies: a directed edge $(t_i, t_r) \in E$ implies that task t_i must complete before task t_r may begin execution. This DAG representation is standard for scientific workflows [3, 4] and can be derived from task-dependency graphs, Makefile rules, or data-pipeline lineage records.

Each task t_i carries a processing demand d_i (in million instructions, MI) and, for each downstream dependency edge, a data-transfer weight representing the volume of output data that must be transmitted before the dependent task can proceed. The latter introduces an implicit dependency between scheduling decisions and network bandwidth allocation, which is a recognized complicating factor in distributed cloud scheduling.

3.2 Objectives

A schedule S maps each task to a VM and a start time, subject to precedence constraints, resource capacity, and VM availability. Three objectives are relevant to the vast majority of cloud operator contexts:

Makespan (f_1): The elapsed wall-clock time from the initiation of the first task to the completion of the last. Minimizing makespan is equivalent to meeting the tightest possible deadline and maximizing throughput under a fixed VM budget.

Monetary Cost (f_2): The total expenditure incurred by provisioning and running VMs for the duration of the schedule. For on-demand instances, this is proportional to the number of VM-hours consumed; for spot

instances, it additionally depends on the market price at each moment.

Energy Consumption (f_3): The total electrical energy consumed by the physical servers executing the scheduled tasks. This objective has grown in practical importance as data-center operators face carbon-reporting obligations and power-usage-effectiveness (PUE) targets. Energy is a function of CPU utilization, idle power draw, and the number of physical servers active at each moment.

The multi-objective scheduling problem is to find the set of Pareto-optimal schedules, those for which no objective can be reduced without worsening at least one other that together constitute the Pareto front. Rather than outputting a single schedule, a Pareto-front approximation presents the decision-maker with a menu of trade-off points from which to select based on the contextual priority of each objective.

3.3 Why Three Objectives Demand NSGA-III

Two-objective variants of the cloud scheduling problem can be handled adequately by NSGA-II, whose crowding-distance mechanism degrades gracefully in two dimensions. Adding a third objective, energy, in our formulation, introduces qualitatively different difficulties. The Pareto front becomes a two-dimensional manifold embedded in three-dimensional objective space, and its geometry can be highly irregular: convex in one subspace and near-linear in another. Crowding distance, which is computed as a sum of objective-normalized distances to neighboring solutions, cannot reliably identify isolated regions of this manifold, leading to loss of diversity and a Pareto-front approximation that is well-populated in some regions and empty in others [5].

NSGA-III's reference-point mechanism addresses this directly. Reference points, distributed uniformly on the normalized objective hyperplane, serve as explicit diversity targets. Because each reference point acts as an attractor for at least one population member, the algorithm is structurally prevented from abandoning any region of the front. Deb and Jain [5] demonstrated this advantage rigorously on the DTLZ benchmark suite with up to 15 objectives; the three-objective case that characterizes our cloud scheduling formulation sits well within the regime where NSGA-III's advantage over NSGA-II is most pronounced.

4 Suitability of NSGA-III for Cloud Scheduling

4.1 Algorithm Structure

NSGA-III follows the canonical generational EMOA loop. At each generation, the current population P (of size N) is used to generate an offspring population Q via crossover and mutation. The combined pool $R = P \cup Q$ (of size 2N) is then partitioned by fast non-dominated sorting into fronts $F_1, F_2, \ldots, F_k$. Fronts are transferred to the next generation in rank order until adding the next front would exceed N. Individuals within the last admissible front are selected by reference-point association: each individual is mapped to its nearest reference point in normalized objective space, and those closest to the least-populated reference points are selected. This selection rule simultaneously maintains elite (high-rank) solutions and distributes them uniformly across the Pareto front.

Three structural properties make this algorithm well-suited to cloud scheduling. First, NSGA-III is parameter-lean: unlike weighted-sum decomposition methods, it does not require per-run weight-vector specification. The number of reference points is determined solely by the number of objectives M and a division parameter p, following the combinatorial formula $C(M + p - 1, p)$. For $M = 3$ and $p = 12$, this yields 91 reference points sufficient to cover a three-dimensional Pareto front with high resolution. Second, the algorithm is population-based, so it naturally maintains a diverse set of candidate schedules across the trade-off space rather than committing to a single solution. Third, because non-dominated sorting and reference-point association are both deterministic given

the population, the algorithm is fully reproducible with a fixed random seed.

4.2 Encoding Challenges

Applying NSGA-III to cloud scheduling requires resolving several encoding decisions that are absent from benchmark studies. A schedule chromosome must represent both a task ordering consistent with DAG precedence constraints and a mapping from tasks to VM instances. Two broad encoding families are available.

Direct encodings represent start times and VM assignments explicitly for each task. They are straightforward to decode into concrete schedules but require expensive repair operators to restore feasibility after crossover, since randomly recombining two feasible start-time vectors typically produces a vector that violates precedence. Indirect encodings represent a topological ordering of the tasks and a VM assignment, with actual start times computed deterministically from this representation by a list-scheduling heuristic. Indirect encodings guarantee feasibility by construction, any topological ordering is a valid schedule skeleton, at the cost of a decoding step that adds computational overhead.

For DAG-structured cloud workloads, indirect encodings are generally preferred in the research literature [3, 4], and we adopt this recommendation as a design principle for a public framework. The trade-off is explicit: decoding cost grows with workflow size, and the decoding heuristic introduces a deterministic bias that the genetic operators cannot directly counteract. Managing this bias for example, by including multiple decoding strategies as selectable options is an open design question discussed further in Section 7.

4.3 Operator Design

Genetic operators, crossover and mutation must be tailored to the chromosome representation to preserve feasibility and explore the search space effectively. For the indirect encoding described above, order-based crossover operators (such as order crossover, OX, or partially matched crossover, PMX) are well established for permutation representations and can be adapted to respect DAG precedence by post-processing any precedence violations after crossover. Mutation can take the form of a random pairwise task swap (with a precedence check) or a VM reassignment for a selected task.

A cloud-specific consideration is VM heterogeneity. Unlike classical parallel-machine scheduling benchmarks, where processors differ only in speed, cloud VM types differ across multiple dimensions simultaneously: CPU count, memory, network bandwidth, and price. An effective VM-reassignment mutation should be aware of this heterogeneity, favoring reassignments that improve utilization balance or reduce cost rather than making random swaps. This requires access to the provider's VM catalogue, which is available via pricing APIs from all major cloud providers.

4.4 Scalability Considerations

The computational cost of NSGA-III's non-dominated sorting step grows as $O(MN^2)$ per generation, where M is the number of objectives and N is the population size. For M = 3 and N = 200, this is manageable with modern hardware. However, real cloud workloads can comprise thousands of tasks, and the fitness evaluation, which requires decoding the chromosome and computing the schedule's makespan, cost, and energy. It can be expensive if involves API calls to the cloud provider's pricing service. A practical framework must implement at least the following efficiency measures: (i) caching of VM pricing data to avoid repeated API calls within a single optimization run; (ii) incremental fitness evaluation that recomputes only the tasks affected by a genetic operator; and (iii) parallel fitness evaluation across the population, exploiting the embarrassing parallelism of population-based methods.

5 A Conceptual Framework Architecture

5.1 Layered Design

A publicly usable NSGA-III-based cloud scheduling framework must bridge three conceptual layers: an input layer that ingests workload descriptions and cloud pricing data; a core optimization layer that executes the NSGA-III search; and an output layer that presents Pareto-front results to the decision-maker and translates the selected schedule into executable API calls. Table 1 maps the principal components within each layer, their responsibilities, and the most consequential design decision each component must resolve. This layered architecture deliberately separates concerns. The input layer is cloud-provider-specific, a different pricing adapter is required for AWS, Azure, and Google Cloud. While the core optimization layer is provider-agnostic and can be developed and tested independently of any production cloud account. The output layer is user-interface-specific and can support multiple front-ends (web dashboard, REST API, command-line tool) without modifying the core.

Table 1. Layered component model for a public NSGA-III cloud scheduling framework

Layer	Component	Responsibility	Key Design Decision
Input	Workload Ingester	Parse DAG from YAML/JSON/GraphML; validate precedence	Schema standardization across workflow formats
Input	Pricing Adapter	Fetch VM catalogue and pricing from cloud provider APIs	Caching strategy; spot-price uncertainty handling
Core	Chromosome Engine	Encode/decode schedule chromosomes; evaluate fitness	Indirect vs. direct encoding; decoder bias management
Core	NSGA-III Solver	Evolve population; manage reference points; apply operators	Reference-point density; operator probability tuning
Core	Constraint Manager	Enforce precedence, capacity, and availability constraints	Repair vs. penalty vs. feasibility-preserving operators
Output	Pareto Front Store	Persist non-dominated front; version by run ID	Archive strategy; convergence stopping criterion
Output	Decision Interface	Visualize trade-off surface; let operator select a schedule	3-D scatter plot; radar chart; preference elicitation
Output	Execution Bridge	Translate selected schedule to cloud provider API calls	Idempotency; rollback on partial failure; audit log

5.2 Workflow Input Specification

A framework intended for public use must accept workload descriptions in formats that practitioners use. The most widely adopted formats for workflow specification in scientific and data-engineering communities are YAML-based DAG files (as used by Apache Airflow), JSON-LD workflow descriptions, and the XML-based BPEL standard for business-process workflows. A standards-compliant ingester should support at least YAML and JSON, with a plugin interface for additional formats. The internal representation should be a typed DAG object with explicit fields for task processing demand, inter-task data transfer volume, and task-level resource requirements (minimum CPU, minimum memory).

A critical practical issue is that task processing demand typically expressed in million instructions or estimated wall-clock time on a reference machine is rarely known precisely in advance. Users often provide estimates derived from historical execution logs or profiling runs. The input layer should accept demand distributions (mean and standard deviation) rather than point estimates, enabling the core optimization layer to reason about schedule robustness. This is an area where the framework design can directly improve upon the deterministic formulations that dominate the research literature.

5.3 Decision-Support Interface

Perhaps the most under-studied component in the research literature is the interface through which a non-specialist cloud operator selects a schedule from a Pareto-front approximation. A Pareto front for a three-objective problem is a two-dimensional manifold, which must be projected and rendered in a form that is simultaneously informative and navigable. Three visualization strategies are worth considering.

A three-dimensional scatter plot, rendered in an interactive web-based environment (for example, using Plotly or Three.js), allows the operator to rotate the front and identify its structure. Color can encode a fourth variable (for instance, the number of VMs provisioned), adding a dimension of information without increasing the geometric complexity. A parallel-coordinates plot, in which each objective occupies a vertical axis and each schedule is a polyline connecting its objective values, allows the operator to filter the front by drawing windows on individual axes. A radar (spider) chart overlays a small number of selected schedules, facilitating direct comparison of their objective profiles.

Beyond static visualization, the interface should support preference elicitation: prompting the operator to provide ordinal priorities (e.g., “cost is more important than energy”) or bound constraints (e.g., “makespan must not exceed 30 minutes”) and using these to highlight or filter the relevant subset of the Pareto front. This bridges the gap between the mathematical output of NSGA-III and the practical decision-making needs of cloud operators who may not be familiar with Pareto optimality as a concept.

6 Pathways to Public Deployment

A single framework design can be deployed in qualitatively different ways depending on the target user base, latency requirements, and governance context. We identify four distinct pathways, each addressing a different segment of the cloud-scheduling user community. Table 2 provides a comparative overview; the following subsections elaborate on each

6.1 Pathway A: Open-Source Library

The lowest-friction path to public availability is the release of a well-documented, pip-installable .Net or Python library implementing the Cloud NSGA-III framework. This pathway targets researchers who wish to replicate, extend, or compare against the framework, and DevOps engineers in organizations with existing orchestration tooling.

A library release requires: a stable public API with semantic versioning; comprehensive unit and integration tests; example notebooks covering the three major cloud providers and at least one public workload trace; and continuous-integration (CI) pipelines that validate the library against new dependency releases. The library should be deposited on the Python Package

Table 2. Comparative analysis of four deployment pathways for a public NSGA-III scheduling service

Pathway	Primary User	Latency Profile	Governance Model	Reproducibility
Open-Source Library	Researchers, DevOps engineers	Offline (minutes)	Community / Apache / MIT	Fully reproducible with fixed seed
Cloud Provider Plugin	Cloud platform tenants	Near-real-time (seconds)	Provider ToS; vendor lock-in risk	Reproducible within provider sandbox
SaaS Scheduling API	Any cloud user (multi-cloud)	Real-time (< 1 s target)	Commercial SLA; data-privacy policy required	Partial — depends on API versioning
Embedded DC Advisor	Data-center operators	Batch / advisory (hours)	Internal IT governance	High — full control of environment

Index (PyPI) and mirrored on a public code repository (GitHub or GitLab) under an open-source license, MIT or Apache 2.0 are appropriate choices that maximize uptake while imposing minimal constraints on commercial use.

6.2 Pathway B: Cloud Provider Plugin

Major cloud providers offer extension points for custom scheduling logic. AWS Step Functions supports custom state machine interpreters; Azure Batch supports custom job schedulers via the Azure Batch service API; Google Cloud Workflows supports custom retry and branching logic. A provider-specific plugin would embed the NSGA-III optimizer within the provider's own scheduling infrastructure, giving it direct access to real-time VM availability and pricing data without requiring the user to manage a separate optimization service.

This pathway reduces latency substantially: because the plugin runs within the provider's own control plane, it can retrieve VM state without cross-network API calls and can cache pricing data in memory close to the scheduling decision. The trade-off is vendor lock-in: a plugin developed for AWS Batch cannot be ported to Azure without a significant re-implementation effort. An abstraction layer that separates the provider-specific cloud adapter from the NSGA-III core mitigates this risk and supports multi-cloud users.

6.3 Pathway C: SaaS Scheduling API

A Software-as-a-Service (SaaS) deployment exposes the framework as a REST or gRPC API accessible to any cloud user regardless of their provider. The user submits a workload DAG and a set of VM options (with prices), and the API returns a Pareto-front approximation as a JSON document from which the user selects a schedule. This pathway has the broadest potential reach, as it is provider-agnostic and requires no software installation on the client side.

A SaaS deployment introduces a set of engineering and governance challenges that are absent from the library pathway. Latency: the API must respond within a time budget compatible with deployment pipelines (ideally under one second for small workloads, under 30 seconds for large ones), which requires either time-bounded optimization runs or a tiered service model in which a fast heuristic provides an immediate response while the full NSGA-III run proceeds asynchronously. Data privacy: workload DAGs may encode commercially sensitive information (task names, data volumes, inter-task dependencies), which must be handled under a published data-processing agreement. Multi-tenancy:

concurrent users must be isolated to prevent interference between optimization runs.

A practical implementation might use a serverless compute layer (AWS Lambda, Google Cloud Run) for small workloads and a provisioned GPU or CPU cluster for large-workload runs, with a queue-based architecture mediating between incoming requests and optimization workers. The Pareto-front results should be cached per workload fingerprint so that repeated submissions of the same or similar workloads benefit from previous optimization runs.

6.4 Pathway D: Embedded Data-Centre Advisor

For organizations that operate their own data centers or private clouds — including large enterprises, national research computing facilities, and telecommunications providers — an embedded advisory system represents the highest-value deployment model. In this model, the NSGA-III framework runs as a long-lived service within the data center's management plane, continuously ingesting workload forecasts, VM utilization telemetry, and energy metering data, and producing periodic Pareto-front recommendations for the next scheduling window.

This pathway allows the framework to be trained on organization-specific workload patterns and hardware characteristics, producing recommendations that are calibrated to the actual energy model of the data center's servers (measured via IPMI or iDRAC power sensors) rather than generic SPECpower approximations. The governance model is entirely internal: the organization controls the data, the model, and the decision. The limitation is the engineering investment required for integration with existing data-center management software (OpenStack, VMware vSphere, Kubernetes), which is substantial.

7 Open Research Challenges

The framework design and deployment pathways described in Sections 5 and 6 rest on a number of assumptions that the current literature has not fully validated. We identify seven open research challenges that constitute the most pressing gaps between the conceptual framework proposed here and a production-ready service.

7.1 Dynamic Workload Arrival and Rescheduling

The formulation in Section 3 assumes a static workload, a single DAG submitted in its entirety before scheduling begins. Real cloud workloads arrive as a stream of jobs, each with its own DAG, deadline, and priority. When a new job arrives, the optimal schedule for the existing workload may no longer be optimal; rescheduling is required. However, full re-optimization from scratch for every new arrival is computationally prohibitive. A promising direction is warm-starting NSGA-III from the previous generation's population when a new job arrives, discarding any solutions that violate the new constraints. Whether this approach converges to a near-optimal front faster than cold-start optimization is an open empirical question.

7.2 Handling Task Duration Uncertainty

Task duration estimates are notoriously imprecise in practice. A workflow that historically completes in 20 minutes may take 45 minutes due to data skew or co-tenant interference. Schedules computed under deterministic duration assumptions may become infeasible or severely suboptimal when actual durations deviate. Robust multi-objective optimization in which the objectives are evaluated under multiple duration scenarios and the schedule is selected to minimize worst-case or expected objective values is theoretically appealing but significantly increases the computational cost of fitness evaluation. Efficient approximations, such as evaluating each schedule under a small sample of scenarios rather than the full distribution, warrant investigation.

7.3 Energy Modelling Accuracy

The energy objective in Section 3.2 depends on an energy model that relates CPU utilization to physical power draw. Published models, derived from SPECpower benchmark data, provide reasonable approximations for standardized server hardware but may be significantly inaccurate for the heterogeneous and proprietary hardware found in major cloud data centers. A framework that targets energy minimization

must either access real power metering data — available in some data-center advisory deployments but not in shared public cloud environments — or use conservative model bounds that acknowledge the uncertainty. The impact of modelling inaccuracy on the quality of Pareto-front recommendations has not been quantified in the literature.

7.4 Carbon-Aware Scheduling as a Fourth Objective

Carbon intensity of electricity generation varies substantially by geography and time of day. Scheduling a workload to run in a region with high renewable energy penetration at an off-peak time can reduce its carbon footprint significantly without altering its makespan, cost, or energy consumption. Adding carbon intensity as a fourth scheduling objective transforms the problem from three-objective to four-objective, further motivating the use of NSGA-III (which handles high-dimensional fronts better than NSGA-II) and introduces a new data dependency: real-time marginal carbon intensity data from electricity grid operators. APIs for this data exist (for example, Electricity Maps and the UK National Grid ESO API), but their integration into a scheduling framework raises questions of data availability, latency, and geographic coverage that have not been addressed in the scheduling literature.

7.5 Fairness and Multi-Tenant Equity

A SaaS scheduling service handles workloads from many different users simultaneously. Optimizing aggregate system-wide objectives (total energy, total cost) may systematically disadvantage specific user classes for example, consistently delaying the workloads of users with flexible deadlines in favor of those with tight ones, or favoring large workloads that yield better parallelism efficiency over small ones. The multi-objective optimization literature has begun to incorporate fairness as an explicit objective [8], but its application to multi-tenant cloud scheduling has not been explored. A public scheduling service has an ethical obligation to be transparent about any such trade-offs.

7.6 Explainability of Pareto-Front Recommendations

When a scheduling service recommends a set of Pareto-optimal schedules, the cloud operator needs to understand not just which schedule to select but why that schedule achieves its objective values. An operator asking "why does schedule A cost $2.40 more than schedule B?" needs an answer in terms they can act on (for example, "because schedule A provisions two additional large-memory VMs to reduce makespan by 18 minutes"). Generating such explanations from the chromosome representation and fitness values is a form of post-hoc explainability that has received little attention in the evolutionary computation community. It is, however, essential for user trust in a public service.

7.7 Benchmarking Infrastructure

Progress in multi-objective cloud scheduling research is impeded by the absence of a shared, standardized benchmarking infrastructure. Existing public workload traces (Google Cluster Trace, Azure Public Dataset, Bitbrains) are valuable but were collected for different purposes and require non-trivial pre-processing before they can be used as scheduling benchmarks. A community-maintained repository of workload instances, with standardized formats, known optimal or best-known solutions, and automated performance reporting, would accelerate progress in the field in the same way that the DIMACS and TSPLIB repositories accelerated combinatorial optimization research. Establishing such a repository and the community governance structure to maintain it is a social-scientific challenge as much as a technical one.

8 Conclusion

This paper has presented a conceptual framework for applying NSGA-III to the multi-objective cloud resource scheduling problem, with the explicit goal of informing the design of a publicly deployable optimization service. We have argued that the three-objective formulation, makespan, monetary cost, and energy is qualitatively different from the two-objective variants that dominate the existing literature, and that

this difference motivates the use of NSGA-III's reference-point diversity mechanism over the crowding-distance approach of NSGA-II. We have identified the key design decisions that a practical realization must resolve chromosome encoding for DAG-structured workloads, cloud-aware genetic operators, scalability management, and decision-support interfaces for non-specialist operators.

Four deployment pathways have been proposed and analyzed: an open-source Python library for researchers and DevOps practitioners; a cloud provider plugin for tenants seeking low-latency scheduling within a single provider's ecosystem; a multi-cloud SaaS API for the broadest possible user base; and an embedded data-center advisor for organizations with private cloud infrastructure. Each pathway involves distinct trade-offs between latency, governance, reproducibility, and engineering investment.

Seven open research challenges have been identified that stand between this conceptual framework and a production-ready service. These challenges, dynamic rescheduling, task duration uncertainty, energy modelling accuracy, carbon-aware scheduling, multi-tenant fairness, explainability, and benchmarking infrastructure; represent a research agenda that the cloud computing and evolutionary computation communities must address together if the theoretical promise of multi-objective evolutionary scheduling is to be realized in practice.

The authors hope that this conceptual synthesis will serve as a foundation for future empirical work. The framework design proposed here is explicitly intended to be testable: each component raises specific, falsifiable research questions, and the deployment pathways provide concrete engineering targets against which implemented systems can be evaluated. We invite the community to engage with these questions, to challenge the design choices made here, and to contribute toward a public-facing multi-objective cloud scheduling service that can be used, studied, and improved by the broader research community.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication 800-145, Sep. 2011. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-145

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010. https://doi.org/10.1145/1721654.1721672

[3] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 5, pp. 1344–1357, May 2016. https://doi.org/10.1109/TPDS.2015.2446459

[4] J. J. Durillo and R. Prodan, "Multi-objective workflow scheduling in Amazon EC2," Cluster Computing, vol. 17, no. 2, pp. 169–189, Jun. 2014. https://doi.org/10.1007/s10586-013-0325-0

[5] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints," IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577–601, Aug. 2014. https://doi.org/10.1109/TEVC.2013.2281535

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, Apr. 2002. https://doi.org/10.1109/4235.996017

[7] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274, Mar. 2002. https://doi.org/10.1109/71.993206

[8] K. Deb, "Multi-Objective Optimization Using Evolutionary Algorithms. Chichester, UK: Wiley, 2001. ISBN: 978-0-471-87339-6

[9] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," IEEE Transactions on Evolutionary Computation, vol. 11, no. 6, pp.

712–731, Dec. 2007. https://doi.org/10.1109/TEVC.2007.892759

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, pp. 23–50, Jan. 2011. https://doi.org/10.1002/spe.995

[11] Google LLC, "Google Cluster Data 2019," 2019. [Online]. Available: https://github.com/google/cluster-data [Accessed: 07 May 2026]

[12] M. Cortez, M. Baughman, J. Gittens, T. Chan, and C. Bhagwan, "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP '17), Shanghai, China, 2017, pp. 153–167. https://doi.org/10.1145/3132747.3132772

[13] S. Shen, V. van Beek, and A. Iosup, "Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters," in Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Shenzhen, China, 2015, pp. 465–474. https://doi.org/10.1109/CCGrid.2015.60