Beyond Target Placement: Unified Multi-Objective Path-Aware Offline VM Scheduling at Scale

Zihan Xu
Alibaba Cloud Computing
Beijing, China
miaotong.xzh@alibaba-inc.com

Zhidong Hu
Alibaba Cloud Computing
Beijing, China
huzhidong.hzd@alibaba-inc.com

Zhengtong Zhang
Alibaba Cloud Computing
Hangzhou, China
zhangzhengtong.zzt@alibaba-inc.com

Fei Peng
Alibaba Cloud Computing
Hangzhou, China
xinyou.pf@alibaba-inc.com

Kaiyuan Shen
Alibaba Cloud Computing
Hangzhou, China
shenkaiyuan.sky@alibaba-inc.com

Feng Wang
Alibaba Cloud Computing
Hangzhou, China
yuelin.wf@alibaba-inc.com

Xiangyun Kong
Alibaba Cloud Computing
Hangzhou, China
xiangyun.kongxy@alibaba-inc.com

Binda Ma
Alibaba Cloud Computing
Hangzhou, China
maxin.mbd@alibaba-inc.com

Jian Zhou
Alibaba Cloud Computing
Beijing, China
xuming@alibaba-inc.com

Abstract

Offline scheduling of virtual machines (VMs) in large-scale cloud clusters faces three core challenges: uniformly modeling heterogeneous business objectives, efficiently solving NP-hard problems at industrial scale, and producing feasible migration paths rather than merely computing target placements. This paper presents PAMOS (Path-Aware Multi-objective Offline Scheduling), an offline scheduling system built around a unified modeling framework that abstracts diverse constraints and objectives into a single additive cost minimization problem through several composable cost forms. A domain-specific language (DSL) enables non-expert users to define rules in fewer than ten lines of code. At the solving level, a heuristic algorithm combining active and passive migration mechanisms directly produces feasible migration-path sequences. SVD-accelerated Tag-based Cost computation and a level-cache mechanism meet the minute-scale planning-time service-level objective (SLO). Evaluation on 631 production problems shows that, after path construction, three placement-based baselines realize only 14.7–20.3% of PAMOS's optimization effectiveness and retain only 10.3–13.6% of their proposed migrations. At the scale where exact comparison is feasible, PAMOS achieves 97.0% of the path-aware mixed-integer programming (MIP) solver's optimum with four orders of magnitude speedup. PAMOS has run stably at Alibaba Cloud for over eight years, processing thousands of tasks daily, achieving substantial cost reductions and stability improvements.

CCS Concepts: • **Software and its engineering → Scheduling**.

Keywords: cloud computing, virtual machine, offline scheduling, live migration

ACM Reference Format:
Zihan Xu, Zhidong Hu, Zhengtong Zhang, Fei Peng, Kaiyuan Shen, Feng Wang, Xiangyun Kong, Binda Ma, and Jian Zhou. 2026. Beyond Target Placement: Unified Multi-Objective Path-Aware Offline VM Scheduling at Scale. In *The 2026 ACM SIGOPS Annual Technical Conference (SIGOPS ATC 2026), November 15–18, 2026, Hong Kong, China.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3828523.3857123



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

SIGOPS ATC 2026, Hong Kong, China

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2818-1/2026/11
https://doi.org/10.1145/3828523.3857123

1 Introduction

Cloud providers manage hyperscale heterogeneous computing clusters and sell virtualized slices of physical machine (PM) resources as virtual machines (VMs). VM scheduling—the process of placing VMs onto suitable PMs—is one of the central problems in cloud resource management. The scheduling process must optimize multiple business objectives such as defragmentation, load balancing, and user dispersion while satisfying hard constraints including resource capacity limits and stability isolation requirements.

Online scheduling systems prioritize response speed over global optimality [17, 25, 41], and over time the resulting suboptimal placements accumulate into what we term *placement debt*—fragmented resources, load hotspots, and dispersion violations (detailed in Section 2). Offline scheduling uses live migration to adjust VM positions in bulk, serving as the primary mechanism for repairing placement debt and enabling cloud providers to reduce failure risk, improve utilization,

and achieve significant cost savings. These benefits compound at cloud scale: for example, in a domain of thousands of PMs, a one-percentage-point gain in packing density can free dozens of PMs and yield millions of dollars in annual savings.

However, building a production-grade offline scheduling system presents three core challenges.

The first challenge is unified multi-objective modeling. Offline scheduling must serve dozens of heterogeneous business requirements spanning PM-level attributes, VM-level attributes, VM–PM relationships, and VM–VM relationships, while also accommodating the topological complexity introduced by non-uniform memory access (NUMA) architectures [27]. These requirements are often proposed independently by different teams and may conflict with or duplicate one another. The system must therefore provide a unified modeling mechanism that incorporates all requirements into a single optimization framework while remaining sufficiently easy to use.

The second challenge is efficient solving at industrial scale. As a variant of the multi-dimensional bin packing problem, the scheduling problem is NP-hard. In the production environment of Alibaba Cloud, a scheduling task may involve tens of thousands of PMs and hundreds of thousands of VMs. As online VM placement changes continuously, an excessively long planning horizon renders the result stale, and hence the system operates under a minute-scale planning-time service-level objective (SLO), meaning that solving should complete on the order of minutes. This constraint rules out the direct use of mixed-integer programming (MIP) solvers.

The third challenge is feasible migration-path planning. Live migration carries stability risks, thus VMs must be migrated individually obeying stability constraints. Consequently, the final output of offline scheduling is not merely a target VM placement but a feasible migration sequence with a well-defined temporal-topological ordering. This dimension is largely overlooked by existing work, most of which solves only for a final placement without considering whether a feasible path to that placement exists. Section 2.3 gives concrete examples of how legal target placements can be unreachable.

Existing work addresses the above challenges to varying degrees. Exact-solver approaches [38] offer strong modeling expressiveness but face scalability challenges and ignore migration-path feasibility. Heuristic solvers [36, 44] offer efficiency advantages but are typically designed for single objectives. Rebalancer [26] provides the most comprehensive framework at Meta's scale but relies on Box-centric local search that does not consider the migration-related constraints. A detailed comparison is provided in Section 8.

This paper proposes PAMOS, a large-scale offline scheduling system whose central thesis is that *end-to-end migration-path generation*—directly producing feasible migration sequences rather than computing target placements and constructing paths post hoc—is better suited to offline scheduling under production migration constraints. On 631 production problems, the three two-stage baselines realize only 14.7–20.3% of PAMOS's optimization effectiveness after path construction (Section 6.3), while PAMOS achieves 97.0% of the path-aware MIP solver's optimum at the scale where exact comparison is feasible, with four orders of magnitude faster solving time. The main contributions are:

1. A unified multi-objective modeling framework that abstracts heterogeneous business objectives into five composable cost forms with full NUMA support. A domain-specific language (DSL) allows non-expert users to define rules in fewer than ten lines of code. A rule management platform provides unified lifecycle management of modeling rules and scheduling tasks in a multi-team production environment.
2. A migration-path-oriented heuristic solver combining active/passive migration mechanisms to directly produce feasible migration paths. Several acceleration mechanisms enable industrial-scale solving within a minute-scale SLO.
3. Extensive production validation over eight years on Alibaba Cloud's hyperscale clusters, and extensive experimental results on both simulation and production environments demonstrating substantial cost reductions and stability improvements.

2 Background and Motivation

2.1 The Offline Scheduling Problem

Cloud providers place VMs onto suitable PMs through VM scheduling. VM scheduling must conform to a set of business rules: rules that must be strictly satisfied are called *constraints* (e.g., resource capacity limits), while rules that are desired but not mandatory are called *objectives* (e.g., reducing fragmented resources). Scheduling is therefore a combinatorial optimization problem that optimizes multiple objectives subject to a set of constraints.

VM scheduling proceeds in two phases. The first phase is online scheduling: when a VM is initially created, the mainstream industry approach adopts a Filter–Scorer architecture that excludes non-compliant PMs via filtering rules and then assigns a composite score to candidate PMs via heuristic scorers [17, 25, 41], trading optimality for decision speed. The second phase is offline scheduling (also known as rescheduling): over time, the accumulated sub-optimality of online decisions, the staggered release of VMs with different lifecycles, and PM (de)commissioning operations collectively degrade the cluster status, producing fragmented resources, load imbalances, and dispersion violations—referred to as

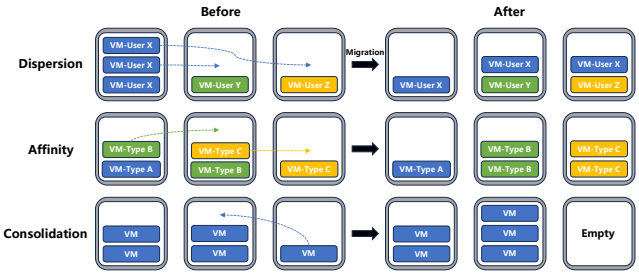


Figure 1. Illustration of three representative scenarios.

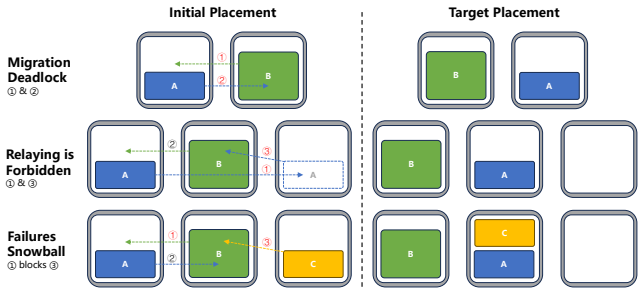


Figure 2. Three examples of infeasible migration paths under a legal target placement.

placement debt. Offline scheduling leverages live migration to adjust VM positions across PMs in bulk without disrupting VM operation, pursuing a globally optimal cluster state.

Unlike online scheduling, offline scheduling has a global cluster view and no millisecond-level deadline, so it can use more sophisticated algorithms over a broader action space—but it faces a more complex mix of constraints and objectives (e.g., consolidation, dispersion, (anti-)affinity) while still producing a feasible live-migration sequence that satisfies stability requirements.

2.2 Representative Business Scenarios

To illustrate the relational complexity and objective diversity inherent in offline scheduling, Figure 1 presents three representative business scenarios.

Dispersion. To limit the blast radius of failures, cloud providers require that the number of VMs belonging to the same user on a single PM does not exceed a *dispersion threshold*. Due to sub-optimal online placements or temporal configuration changes, some PMs may violate this threshold and require repair through offline scheduling.

Affinity and anti-affinity. Several scenarios need affinity or anti-affinity: co-locating same-lifecycle VMs to reduce future fragmentation [3, 29, 43], dispersing high-load VMs to avoid hotspots, and co-locating VMs and PMs with matching resource ratios to reduce flavor mismatch. These VM–VM and VM–PM relationships are hard to express through a single optimization objective. We use lifecycle affinity as the representative affinity scenario hereafter.

Consolidation. Fragmented resources reduce cluster utilization and can be reclaimed by consolidating VMs onto fewer PMs, freeing PMs for large-flavor VMs or maintenance operations. The core of this scenario is resource recombination from a global perspective. Because a PM yields its benefit only after every VM on it has successfully moved off, consolidation is particularly sensitive to migration-path feasibility (Section 2.3).

These scenarios demonstrate that offline scheduling involves multiple heterogeneous relationships among VMs and PMs, and that the optimization objectives of different scenarios are diverse. A unified modeling framework is therefore needed to incorporate all requirements into a single optimization system for integrated planning. We report production results for these scenarios (Section 6.5), each summarized by one higher-is-better business metric: *consolidation* by packing density $\rho = \sum_{i\in\mathcal{I}} \mathrm{CPU}_i / \sum_{b\in\mathcal{B}_{occ}} \mathrm{CPU}_b$ over the occupied PMs $\mathcal{B}_{occ}$; *dispersion* by the compliance rate $\gamma = |\{i \in \mathcal{I} : \mathrm{compliant}(i)\}|/|\mathcal{I}|$, the fraction of VMs whose per-PM co-tenant count meets the dispersion threshold; and *lifecycle affinity* by lifecycle clustering density $\rho_{lc} = \sum_{i\in\mathcal{I}_{long}} \mathrm{CPU}_i / \sum_{b\in\mathcal{B}_{long}} \mathrm{CPU}_b$ over long-lived VMs $\mathcal{I}_{long}$ and their hosts $\mathcal{B}_{long}$.

2.3 Why Migration-Path Feasibility Matters

Offline scheduling must respect three kinds of rules while it moves VMs. First, live migration copies a VM's state to the destination before releasing the source [7], so the destination must have room for the VM while the VM still occupies its source. Second, production stability rules allow each VM to migrate at most once per scheduling task, cap the total number of migrations, and mark some VMs as immovable. Third, placement constraints (e.g., resource capacity, mutual exclusion) must hold in every intermediate state. Section 4.2 defines these rules formally.

The prevailing approach to rescheduling first computes a target placement and then realizes it by migrating VMs one at a time, a step called *path construction*; this separation implicitly assumes that a legal target is reachable. However, Figure 2 shows three ways this assumption fails. Each row pairs an initial placement (left) with a legal target placement (right); arrows are the migrations that path construction would need, numbered in their intended order.

Migration deadlock. In the first row, VMs A and B must swap PMs. Both placements are legal, but each migration (① and ②) needs its destination free before releasing its source, thus neither can start. Such deadlocks grow more frequent as packing tightens, which is exactly consolidation's goal.

Relaying is forbidden. In the second row, a spare PM seems to break the deadlock by moving A to it (①), releasing

the space temporarily for B. But the relay moves A twice (① and ③), which the one-move-per-VM stability rule forbids, as consecutive migrations cause jitter.

Failures snowball. In the third row, the target requires three VMs to move. If B fails to move (①), it retains the capacity needed by C (③), so C also fails. Such dependencies allow one blocked migration to propagate through the remaining path.

Implications for solver design. Accounting for feasible paths is difficult in both exact and heuristic methods. A static placement model decides only *where* each VM ends up, requiring $O(|\mathcal{I}||\mathcal{B}|)$ assignment variables. A path-aware formulation additionally represents migration order and source–destination pairs, expanding a time-indexed formulation to $O(|\mathcal{I}|^2|\mathcal{B}|^2)$; this exact formulation quickly becomes intractable (Section 6.2). Conversely, post-hoc path construction must preserve a target chosen without regard to reachability and can only discard infeasible moves; Section 6.3 shows that most planned migrations are then lost. Path feasibility therefore cannot be deferred to post-processing: migrations and their order must be generated jointly. Section 5 presents the resulting PAMOS design.

3 System Overview

PAMOS adopts a three-stage pipeline (Figure 3): modeling, solving, and evaluation.

Modeling. Users write rules in a concise DSL and manage them on a versioned rule-management platform. When a task is initiated, the system selects the required rule combination and compiles it into an abstract syntax tree (AST). A data engine then loads scheduling metadata from various online sources and combines it with the AST to produce a standardized problem file (Section 4).

Solving. A task scheduler runs multiple solvers in parallel. The default is the heuristic migration-path solver of this paper (Section 5), optionally in several configurations tuned to different problem structures specified by the user; a standardized I/O interface also lets users with special needs develop and plug in custom solvers, whose migration paths are evaluated uniformly.

Evaluation. All candidate solutions are collected and undergo unified evaluation: (1) feasibility verification: a constraint check to verify the path satisfies all constraints; noncompliant solutions are immediately discarded; (2) effectiveness evaluation: the system computes the total objective decrease along the path to judge solution quality and pick the best; and (3) metric computation: a simulated migration is performed to report business-metric changes (e.g., packing density, compliance rate), enabling users to intuitively assess scheduling effectiveness, since the abstract objective change alone cannot accurately reflect real impact. Once the results meet expectations, the chosen plan is then dispatched for execution.

4 Modeling Framework

This section describes the modeling framework of PAMOS.

4.1 Object Abstraction

PAMOS abstracts the offline scheduling problem as a multi-dimensional bin packing problem with topological structure. PMs are abstracted as Boxes with multi-dimensional resource capacities, and VMs are abstracted as Items with multi-dimensional resource demands.

Modern cloud servers widely adopt NUMA architectures [27], where a PM has multiple NUMA nodes and different resources are shared at different levels. PAMOS captures this with a two-level resource model: each Box has overall resources shared at the whole-PM level (e.g., network bandwidth, storage I/O) and node resources managed per NUMA node (e.g., CPU, memory). Formally, a Box is a tuple $\langle id, R^{overall}, R^{node}, \mathcal{N} \rangle$, where $R^{overall}$ and R^{node} are the two-level resource vectors and $\mathcal{N}$ its NUMA node sequence.

Correspondingly, each Item has both overall and node resource consumption attributes. In addition, each Item carries the number of NUMA nodes it requires n_{node}, a Boolean *migratable* indicating whether live migration is permitted, and its current placement information; formally, an Item is a tuple $\langle id, R^{overall}, R^{node}, n_{node}, migratable, box_{cur}, nodes_{cur} \rangle$.

4.2 Constraint System

PAMOS defines four classes of hard constraints. The solver never introduces new violations and preserves every constraint already satisfied in the initial state; legacy violations already present in that state (e.g., historically erroneous placements) are penalized and repaired when possible.

Resource-dimension constraints. ensure that resource usage at every level does not exceed its capacity. These constraints operate for each Box and each node:

$$\forall b \in \mathcal{B} : \sum_{i \in I_b} R_i^{overall} \le R_b^{overall}$$

$$\forall b \in \mathcal{B}, \forall n \in \mathcal{N}_b : \sum_{i \in I_{b,n}} R_i^{node} \le R_b^{node}$$

Mutual-exclusion constraints. Item–Box mutual exclusion prohibits certain Items from being placed on Boxes with which they have an exclusion relationship. Item–Item mutual exclusion prohibits two Items with an exclusion relationship from coexisting on the same Box.

Migration-control constraints. impose strict limits on live migration for stability: each Item migrates at most once per task (avoiding jitter from consecutive migrations), the total migration count must not exceed a preset bound K

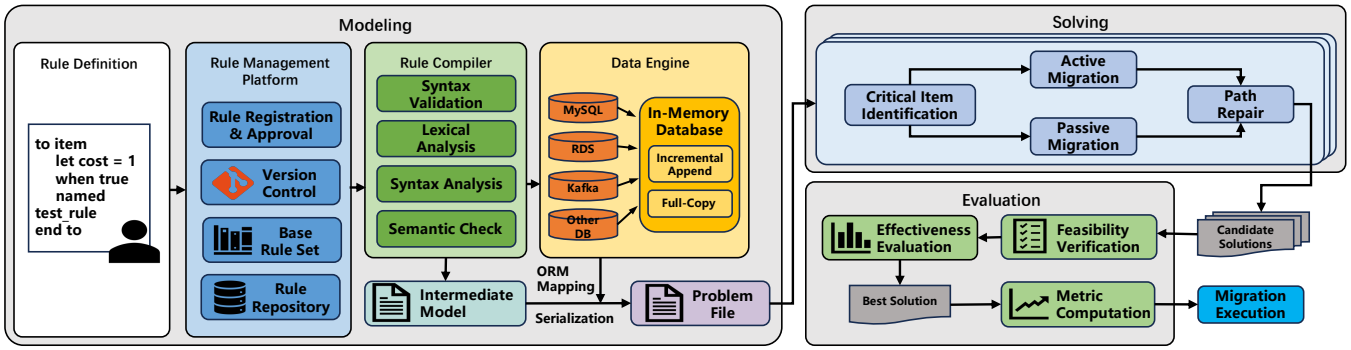


Figure 3. System architecture of PAMOS, consisting of modeling, solving, and evaluation stages.

to prevent system instability from large-scale concurrent migrations, and non-migratable Items never move.

Legal-placement constraints. require that every Item be placed on exactly one Box and exactly n_{node} NUMA nodes satisfying its topology requirement.

4.3 Cost-Based Multi-Objective Modeling

PAMOS unifies heterogeneous business objectives into a single additive cost minimization. The key principle is composability: each requirement is expressed as one of five standardized cost forms, and the system optimizes their sum. To make the cost semantics intuitive, values use a force analogy—positive cost encourages separation (repulsion) while negative cost encourages co-location (attraction)—so non-experts define rules by reasoning about "forces" rather than abstract objective formulas. The decomposable structure also enables incremental evaluation during path search: when an Item migrates, only the local costs involving it and the affected Boxes are recomputed (Section 5.5). PAMOS defines five cost forms, uniformly called Cost.

Use Box Cost. assigns a usage cost to each Box. This Cost is added whenever a Box contains any Item, enabling objectives such as minimizing the number of active PMs.

Item Box Cost. assigns a placement cost to every Item–Box pair. This Cost is added when a particular Item is placed on a particular Box, modeling VM–PM relationships.

Item Item Cost. assigns a coexistence cost to every pair of Items. This Cost is added when two particular Items co-locate on the same Box, modeling VM–VM relationships.

Migration Cost. assigns a migration cost to each Item, representing the stability risk of live migration, thus restricting the number of migrations.

Constraint Violation Cost. assigns costs to legacy violations already present in the initial state, turning their repair into an objective without permitting new violations. Different types of constraint violations are assigned different Cost values based on their severity.

Based on the above definitions, the offline scheduling objective is uniformly formulated as:

$$\min_{\pi} \sum_{b\in\mathcal{B}} C_{use}(b)\cdot\mathbf{1}[I_b\neq\emptyset] + \sum_{i\in\mathcal{I}}\sum_{b\in\mathcal{B}} C_{ib}(i,b)\cdot\mathbf{1}[i\in I_b] + \sum_{b\in\mathcal{B}}\sum_{i,j\in I_b} C_{mix}(i,j) + \sum_{i\in\mathcal{M}} C_{mig}(i) + C_{vio}(\pi)$$

where π is a feasible migration path and $\mathcal{M}$ is the set of migrated Items. Let $\mathcal{E}(\mathcal{S}_T^{\pi})$ denote the non-migration terms above, including residual violation Cost; the total path objective is therefore $\mathcal{E}(\mathcal{S}_T^{\pi}) + \sum_{i\in\mathcal{M}} C_{mig}(i)$.

4.4 Modeling Language

To lower the modeling barrier, PAMOS provides a user-friendly domain-specific language (DSL) with a primarily declarative style: users specify a rule's object type, condition, and content, and most rules take fewer than ten lines. The basic structure is:

```
to <target_type>
    <action> when <condition> [named <name>]
end to
```

Here `target_type` is the object to which the rule applies (item, box, or item&box pair), `action` is the specific operation (adding an attribute, defining a Cost, defining mutual exclusion, declaring non-migratability, etc.), `condition` is the trigger condition, and `name` is the rule name. Three examples illustrate the language.

Figure 4a shows the attribute definition. The code adds a resource dimension named `dispersion` to both Items and Boxes, meaning each PM hosts at most three VMs of that user, enforcing the dispersion requirement.

Figure 4b shows the cost function definition. The first case assigns an `item_box_cost` of −100 to VM–PM pairs with matching resource ratios, drawing them together by an attractive force to reduce fragmentation. The second case assigns a positive `item_item_cost` proportional to CPU to pairs of high-load VMs, pushing them apart by a repulsive force to separate high-load VMs and achieve load balancing.

Figure 4c shows the mutual-exclusion definition. The first case applies mutual exclusion between exclusive VMs and all

```
to item
  add attribute {"dispersion": 1}
  when item.user_id = "123"
  named item_dispersion
end to
```

```
to box
  add attribute {"dispersion": 3}
  when true
  named box_dispersion
end to
```

(a) Attribute definition

```
to item, box
  let cost = -100
  when item.cpu * box.mem = box.cpu * item.mem
  named ratio_affinity
end to
```

```
to item, item
  let cost = first.cpu + second.cpu
  when first.load = 'high' && second.load = 'high'
  named load_anti_affinity
end to
```

(b) Cost function

```
to item, item
  mutex
  when first.exclusive
  named item_exclusive
end to
```

```
to item, box
  mutex
  when box.id = 'x'
  named empty_box
end to
```

(c) Mutual exclusion

Figure 4. Code snippet examples of the modeling DSL.

Table 1. Examples of frequently used rules.

Rule name	Action type	LOC	Description
`single_user_dispersion`	Add Attribute	10	Limit co-tenant VMs per PM
`lifecycle_affinity`	Item–Item Cost	5	Co-locate same-lifecycle VMs
`high_load_vm_dispersion`	Item–Item Cost	5	Separate high-load VMs
`empty_particular_box`	Item–Box Excl.	5	Empty target PMs
`resource_ratio_affinity`	Item–Box Cost	5	Match VM/PM resource ratios
`flavor_machine_binding`	Item–Box Excl.	8	Restrict VM flavors per PM type
`not_migrate_critical_vm`	Mig. Control	3	Prohibit migrating critical VMs
`not_use_reserved_box`	Item–Box Excl.	4	Exclude reserved PMs
`basic_use_box_cost`	Use Box Cost	3	Box usage cost (CPU+Mem)
`basic_migrate_item_cost`	Mig. Cost	3	Migration cost (CPU+Mem)

other VMs, enforcing exclusive occupancy. The second case aims to empty a specific PM by applying mutual exclusion between that PM and all VMs.

For advanced scenarios, the language also supports conditional branches (`if/else`), loops (`for`), function declarations, and basic string and set operations, covering the modeling needs of the vast majority of requirements. Table 1 shows frequently used production rules, most of which require fewer than ten lines of DSL code.

4.5 Rule Management and Compilation

In a multi-team production environment, unrestricted rule definition easily causes logical conflicts or redundancies. PAMOS therefore provides an online rule-management platform for unified lifecycle management; it currently maintains over 200 active rules from 10 independent teams.

Rule registration and management. Users submit DSL rules to the platform for unified review and registration. The platform maintains version history and supports canary release and rollback. When initiating a scheduling task, users select rules from the repository by reusing existing ones or creating new. A base rule set encapsulates the system's fundamental constraints (region restrictions, VM migratability, etc.) for easy reuse.

Rule compilation. Submitted rules are assembled into one script and compiled: (1) syntax validation; (2) lexical analysis; (3) recursive-descent parsing into an AST with semantic checks; and (4) the artifact is cached as an intermediate model in a model repository for reuse, achieving a compile-once-run-anywhere paradigm.

Problem generation. A unified data engine then aggregates scheduling metadata scattered across various data sources, held in an in-memory database with continuous incremental appending and periodic full copying to ensure both freshness and accuracy, loading hundreds of millions of records within seconds. The data is bound to problem objects via object-relational mapping (ORM) and serialized into a JSON-format problem file for downstream solvers.

4.6 Tag-Based Compressed Representation

Storing Cost values for every Item–Box and Item–Item pair would require $O(|\mathcal{I}| \times |\mathcal{B}| + |\mathcal{I}|^2)$ space, which is prohibitive at scale. Because Cost values are assigned uniformly to batches of objects satisfying the same rule conditions, PAMOS instead annotates each object with Tags derived from rule conditions and stores only a Tag-to-Tag relationship matrix. The Cost between any two objects is computed as the sum over their Tag pairs: $Cost(x, y) = \sum_{i \in \mathcal{T}(x)} \sum_{j \in \mathcal{T}(y)} M_{ij}$, where $\mathcal{T}(\cdot)$ is the Tag set. Let t_I and t_B denote Item and Box Tag counts. Storage reduces to $O(t_I \times t_B + t_I^2) + O(|\mathcal{I}| \cdot t_I + |\mathcal{B}| \cdot t_B)$. Because $t_I, t_B \ll |\mathcal{I}|, |\mathcal{B}|$, this yields orders-of-magnitude storage savings.

5 Solving Algorithm

This section describes the core solving algorithm of PAMOS, starting with the intuition behind it. Unlike existing work, the algorithm avoids committing to a target placement and builds a path incrementally. *Active migration* appends a move legal in the current state; when a destination is blocked, *Passive migration* recursively evicts bystander VMs from that destination rather than relay the blocked VM twice, rolling back if any eviction fails. Every raw move is locally legal, whereas Path Repair enforces the global one-move and migration-budget constraints. The resulting feasible path is $\pi = \langle m_1, \ldots, m_T \rangle$, where $m_t = (i, b_s, b_d)$ migrates Item i from b_s to b_d, which can be directly executed as live migrations in production.

Problem restatement. Given the current cluster state $\mathcal{S}_0 = (\mathcal{I}, \mathcal{B}, \phi_0)$, the goal is to find a feasible migration path $\pi^* = \arg\min_{\pi \in \Pi} \mathcal{E}(\mathcal{S}_T^\pi) + \sum_{t=1}^{T} C_{mig}(i_t)$ that minimizes the

Algorithm 1 PAMOS Main Loop

Require: Cluster state $\mathcal{S}_0$, migration budget K, search depth l, early-stop ratio α
Ensure: Feasible migration path π

$\mathcal{I}_{key} \leftarrow$ MultiWayRecall($\mathcal{S}_0$) ▷ Section 5.1
Sort $\mathcal{I}_{key}$ by $\Phi(i)$ descending ▷ Section 5.1.2
$\pi \leftarrow \langle\rangle$; $fails \leftarrow 0$
for each $i \in \mathcal{I}_{key}$ **do**
  if Migrate($i, \mathcal{S}, \pi, l$) **then** $fails \leftarrow 0$
  else $fails \leftarrow fails + 1$
  end if
  if $fails \geq \lceil \alpha \cdot |\mathcal{I}_{key}| \rceil$ **then break**
  end if
end for
$\pi \leftarrow$ PathRepair(π, K) ▷ Section 5.4
return π

total Cost at the terminal state, where Π is the set of feasible paths satisfying all hard constraints, $\mathcal{E}(\mathcal{S})$ is the total system Cost at state $\mathcal{S}$, and $T = |\pi|$ is the number of migration steps. Feasibility requires that no intermediate state introduces a new hard-constraint violation or breaks a constraint satisfied in $\mathcal{S}_0$, that $T \leq K$, and that each Item migrates at most once.

This problem is NP-hard, and the feasible-path constraint further increases the solving difficulty. This paper adopts a heuristic approach with a three-phase pipeline of critical Item identification, active/passive migration, and path repair, supplemented by acceleration mechanisms for efficient solving at industrial scale. Algorithm 1 shows the main loop.

5.1 Critical Item Identification and Priority Ordering

To efficiently identify the Items that contribute most to the total Cost, the algorithm performs independent recall on six dimensions—three constraint-violation dimensions and three placement-dependent Cost dimensions—and then merges the results to form a unified candidate set $\mathcal{I}_{key}$.

5.1.1 Multi-Way Recall Strategy. The critical Item identification for each dimension follows a unified principle: on each Box, identify the Items that contribute the most to that dimension of Cost. The recall channels are:

- Resource-constraint dim.: all Items on Boxes with resource violations, i.e., $\mathcal{I}_{res} = \bigcup_{b \in \mathcal{B}_{vio}} \{i \in I_b \mid \exists r : R_i^r > 0\}$.
- Item–Box mutual-exclusion dim.: Items placed on Boxes with which they have a mutual-exclusion relationship.
- Item–Item mutual-exclusion dim.: Items on the same Box that violate pairwise mutual-exclusion relationships.
- Box usage cost dim.: all Items on Boxes with $C_{use}(b) > 0$, supporting consolidation.
- Item–Box cost dim.: Items whose current C_{ib} is significantly higher than the minimum feasible C_{ib}.
- Item–Item cost dim.: Items whose accumulated C_{mix} on their current Box significantly exceeds the minimum feasible level.

The final critical Item set is $\mathcal{I}_{key} = \mathcal{I}_{res} \cup \mathcal{I}_{ib} \cup \mathcal{I}_{ii} \cup \mathcal{I}_{use} \cup \mathcal{I}_{ib\text{-}cost} \cup \mathcal{I}_{mix}$.

5.1.2 Potential Score and Priority Ordering. After obtaining $\mathcal{I}_{key}$, a potential score $\Phi(i)$ is computed for each Item as a composite priority metric measuring the expected per-unit-resource benefit of migration:

$$\Phi(i) = \frac{1}{S_{res}(i)} \big(\Delta C_{ib}(i) + \Delta C_{mix}(i) + \Delta C_{res}(i) + \Delta C_{use}(i) - C_{mig}(i)\big) \quad (1)$$

where $S_{res}(i) = \sum_{r \in \mathcal{R}_{tight}} R_i^r / \sum_{j \in \mathcal{I}} R_j^r$ is the composite resource score over tight resource dimensions, $C_{mig}(i)$ the migration cost, and each ΔC represents the marginal Cost improvement from moving i to its best feasible Box. Dividing by $S_{res}(i)$ follows a benefit-density principle—migrating small, high-benefit Items first raises the success rate and Cost reduction. All Items are sorted by descending $\Phi(i)$.

5.2 Active Migration

For each Item i in the priority sequence, the algorithm first attempts active migration, which seeks to move i directly to a target Box with sufficient resources. The target Box is selected through a Filter–Scorer strategy.

5.2.1 Filtering Stage. The candidate Box set $\mathcal{B}_{cand}(i)$ is constructed by applying three conditions: (1) no mutual exclusion relationship exists (both Item–Box and Item–Item); (2) Box-level resource capacity is sufficient; (3) at least one NUMA node combination satisfying the topology and resource capacity requirement of the Item exists.

5.2.2 Scoring Stage. For each candidate Box $b \in \mathcal{B}_{cand}(i)$, the algorithm computes a composite matching score:

$$S_{active}(i, b) = \hat{S}_{cos}(i, b) + \hat{S}_{gap}(i, b) + \hat{S}_{cost}(i, b) \quad (2)$$

where $\hat{S}(i, b) = (S(i, b) - \mu_S)/\sigma_S$ is the z-score normalization of sub-score S over $\mathcal{B}_{cand}(i)$ (μ_S, σ_S being the mean and standard deviation), standardizing the sub-scores to a common scale and removing tunable weight hyperparameters.

The three sub-scores capture complementary aspects of placement quality. Resource-ratio matching S_{cos} measures the cosine similarity of the Item and Box resource vectors, reducing fragmentation. Resource gap $S_{gap} = -\|\mathbf{a}_b - \mathbf{R}_i\|_2$ measures tightness of fit. Cost decrease $S_{cost} = \mathcal{E}_i(\mathcal{S}) - \mathcal{E}_i(\mathcal{S}|_{\phi(i) \to b})$ estimates the Cost reduction.

For Boxes with NUMA architecture, the algorithm additionally computes a node-level score for each legal node combination and selects the optimal one. The final score combines the Box-level and NUMA-level scores. Item i is

Algorithm 2 MIGRATE$(i, \mathcal{S}, \pi, l)$

Require: Item i, current state $\mathcal{S}$, migration path π, remaining depth l
Ensure: Whether migration succeeded (Boolean)

$\mathcal{B}_{cand} \leftarrow$ FILTER$(i, \mathcal{S})$
if $\mathcal{B}_{cand} \neq \emptyset$ **then**
  $b^* \leftarrow \arg\max_{b \in \mathcal{B}_{cand}} S_{active}(i, b)$
  Execute migration $(i, \phi(i), b^*)$; update $\mathcal{S}$; append to π
  return true
end if
if $l > 0$ **then**
  return PASSIVEMIGRATE$(i, \mathcal{S}, \pi, l)$
end if
return false

migrated to the Box with the highest score. If no feasible Box exists, the algorithm proceeds to passive migration. Algorithm 2 summarizes this active-first control flow.

5.3 Passive Migration

When active migration fails and the remaining depth $l > 0$, the algorithm triggers passive migration, evicting Items from a target Box to free space for the incoming Item—the multi-step cascading mechanism that distinguishes it from simple greedy strategies and lets it escape local optima.

5.3.1 Target Box Selection. The candidate set uses relaxed feasibility conditions: the Box's maximum potential resources (total Box resources minus the consumption of non-migratable Items) must fit the incoming Item. Candidates are scored with the same z-score normalization as active migration:

$$S_{passive}(i, b) = \hat{S}'_{gap}(i, b) + \hat{S}_{cost}(i, b) + \hat{S}_{yield}(b) \quad (3)$$

where each $\hat{S}$ is z-score-normalized over $\mathcal{B}_{passive}(i)$: S'_{gap} is the resource gap (a smaller gap means fewer evictions are needed), S_{cost} is the Cost-decrease benefit, and $S_{yield}(b) = \frac{1}{|I_b^{mov}|} \sum_{j \in I_b^{mov}} \Phi(j)$ is the mean potential score of movable Items on b, reflecting the expected incidental benefit of eviction—Boxes whose movable Items themselves have high migration value are preferred, as their eviction produces beneficial side effects.

5.3.2 Yield Item Selection and Recursive Migration. Once a target Box b is determined, a minimal yield set of yield Items $\mathcal{I}_{yield} \subseteq I_b^{mov}$ is then chosen by an expand–contract strategy: In the expand phase, Items are added in order of $(S_{res}(j), -\Phi(j))$ until space suffices. In the contract phase, redundant ones are removed while maintaining sufficiency, giving a locally minimal set.

Each yield Item is migrated recursively by a MIGRATE call at depth $l-1$; if all succeed the original Item moves into the freed space, otherwise the algorithm rolls back all evictions and tries the next candidate Box, guaranteeing consistency. Algorithm 3 formalizes the recursive eviction and rollback procedure.

Algorithm 3 PASSIVEMIGRATE$(i, \mathcal{S}, \pi, l)$

Require: Item i, current state $\mathcal{S}$, migration path π, remaining depth l
Ensure: Whether migration succeeded (Boolean)

$\mathcal{B}_{passive} \leftarrow$ FILTERPASSIVE$(i, \mathcal{S})$
Sort $\mathcal{B}_{passive}$ by $S_{passive}(i, b)$ descending
for each $b \in \mathcal{B}_{passive}$ **do**
  $\mathcal{I}_{yield} \leftarrow$ EXPANDCONTRACT$(i, b, \mathcal{S})$
  $\mathcal{S}_{snap} \leftarrow$ SNAPSHOT$(\mathcal{S})$, $\pi_{snap} \leftarrow |\pi|$
  $success \leftarrow$ **true**
  for each $j \in \mathcal{I}_{yield}$ **do**
    if $\neg$MIGRATE$(j, \mathcal{S}, \pi, l-1)$ **then**
      $success \leftarrow$ **false**; **break**
    end if
  end for
  if *success* **then**
    Execute $(i, \phi(i), b)$; update $\mathcal{S}$; append to π
    return true
  else
    ROLLBACK$(\mathcal{S}, \mathcal{S}_{snap})$; truncate π to π_{snap}
  end if
end for
return false

5.4 Path Repair

Although every raw move is legal in the state where it is generated, recursive passive migration may make π_{raw} violate task-level constraints by moving an Item multiple times or exceeding K. Path Repair converts it into a globally feasible final path π.

Path merging. For each Item appearing multiple times, its chain $b_0 \to b_1 \to \cdots \to b_k$ is merged into a single direct migration $b_0 \to b_k$. Cyclic dependencies (rare in practice, <1% of cases) are detected via topological sorting and broken by redirecting the lowest-cost Item in the cycle to a third-party Box as its final destination, so that every Item still migrates at most once.

Benefit verification and budget control. Merged migrations are grouped by originating Cost dimension and sorted by expected Cost decrease. Each group undergoes benefit verification: groups with $\Delta\mathcal{E}_g \leq 0$ are reverted. The algorithm maintains a running migration count and stops once the upper bound K is reached.

5.5 Algorithm Acceleration

To meet the minute-scale SLO, three complementary acceleration mechanisms are proposed, operating at different levels: early stopping reduces search width, vectorized computation

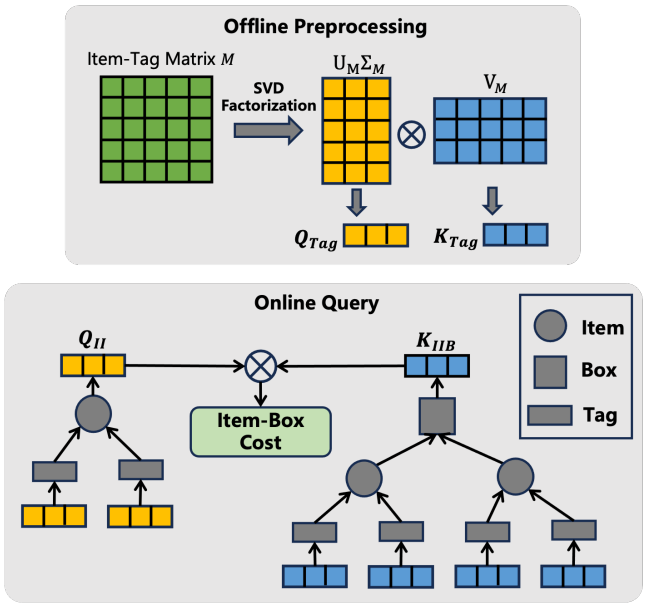


Figure 5. Illustration of vectorized Cost computation: offline, the Item–Tag matrix M is factorized by truncated SVD into d-dimensional Tag embeddings (Q_{Tag}/K_{Tag}, schematic names); online, the Tag embeddings are aggregated into Item and Box embeddings, and each query reduces to two inner products between the Item and Box embeddings.

accelerates Cost evaluations, and caching reduces redundant search depth.

5.5.1 Early Stopping. Because critical Items are sorted by decreasing $\Phi(i)$, expected benefit diminishes toward the tail. A counter tracks consecutive migration failures, resetting on each success; when the count reaches $\lceil \alpha \cdot |\mathcal{I}_{key}| \rceil$ ($\alpha \in (0, 1]$ tunable), the search stops early. This approximate mechanism trades a small amount of optimization effectiveness for lower solving time, as quantified in Section 6.4.

5.5.2 SVD-Based Vectorized Cost Computation. Repeated Cost computation during Item placement can be time-consuming. PAMOS reformulates Tag-based Cost as matrix operations and applies truncated SVD to obtain d-dimensional Item and Box embeddings. Each query then becomes two d-dimensional inner products, and a migration updates the placement-dependent embedding through two $O(d)$ vector additions. As Table 2 shows, this reduces per-query complexity from $O(t_I t_B + t_I^2)$ to $O(d)$ while avoiding the quadratic storage of full precomputation. We use d=50; more than 99% of production tasks have Tag matrices smaller than this dimension, making the decomposition exact for them. The complete derivation is as follows.

Let $M \in \mathbb{R}^{t_I \times t_I}$ be the Item-Tag relationship matrix, $N \in \mathbb{R}^{t_I \times t_B}$ the Item-Tag to Box-Tag matrix, $B \in \{0,1\}^{|\mathcal{I}| \times t_I}$ the Item-to-Tag membership matrix, $D \in \{0,1\}^{|\mathcal{B}| \times t_B}$ the Box-to-Tag membership matrix, and $H \in \{0,1\}^{|\mathcal{B}| \times |\mathcal{I}|}$ the placement

Table 2. Comparison of the Per-query and Space complexity of three Cost computation approaches.

Approach	Per-query	Space
Naive (per-Tag-pair)	$O(t_I t_B + t_I^2)$	$O(t_I t_B + t_I^2)$
Full precomputation	$O(1)$	$O(\lvert\mathcal{I}\rvert\lvert\mathcal{B}\rvert + \lvert\mathcal{I}\rvert^2)$
Proposed (SVD)	$O(d)$	$O((\lvert\mathcal{I}\rvert+\lvert\mathcal{B}\rvert)d)$

matrix, where t_I and t_B are the numbers of Item and Box Tags (Section 4). Materializing all Cost values yields the Item–Item matrix $P \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{I}|}$ and the Item–Box matrix $C \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{B}|}$, whose (x, y) entry is the total Cost of placing Item x on Box y:

$$P = BMB^T \tag{4}$$

$$C = BND^T + BMB^T H^T \tag{5}$$

Applying truncated SVD ($M \approx U_M \Sigma_M V_M^T$, retaining the top d singular values), embedding matrices analogous to Query–Key attention are defined:

$$Q_{II} = BU_M\Sigma_M,\ K_{II} = BV_M \tag{6}$$

$$Q_{IB} = BU_N\Sigma_N,\ K_{IB} = DV_N \tag{7}$$

$$K_{IIB} = HK_{II} \tag{8}$$

The Cost between Item x and Box y reduces to two d-dimensional inner products:

$$C_{xy} = \langle Q_{IB}^{(x)}, K_{IB}^{(y)} \rangle + \langle Q_{II}^{(x)}, K_{IIB}^{(y)} \rangle \tag{9}$$

Static matrices are precomputed once in $O(|\mathcal{I}|\, t_I\, d + |\mathcal{B}|\, t_B\, d)$ time. When Item x migrates from Box y_s to y_d, only K_{IIB} requires updating: subtracting $K_{II}^{(x)}$ from $K_{IIB}^{(y_s)}$ and adding it to $K_{IIB}^{(y_d)}$, two $O(d)$ vector additions independent of problem scale. Figure 5 illustrates the calculation procedure.

5.5.3 Level Cache Mechanism. The recursive search in passive migration causes computation to grow exponentially with depth: unrolling the recurrence yields $T(l) = O(|\mathcal{I}|^l \cdot |\mathcal{B}| \log |\mathcal{I}|)$, which becomes unacceptable at $l \geq 2$ for industrial-scale problems.

The level cache records the minimum successful migration depth $l_{min}(i)$ for each Item. When a search at depth l fails, the cache is updated as $l_{min}(i) \leftarrow l + 1$. In subsequent searches, if the remaining depth $l' < l_{min}(i)$, Item i is skipped without executing an actual search.

Proposition 5.1 (Amortized complexity). *With the level cache, each Item is searched at most once at each depth level, reducing the amortized complexity from* $O(|\mathcal{I}|^l \cdot |\mathcal{B}| \log |\mathcal{I}|)$ *to* $O(l \cdot |\mathcal{I}| \cdot |\mathcal{B}| \log |\mathcal{I}|)$.

The cache assumes that a failure at one state remains informative after the state evolves. Although this need not always hold, fewer than 5% of Items typically move in one run, homogeneous Items can substitute for an incorrectly skipped

peer, and the small production depth ($l \leq 2$) limits the affected search branches. Experiments show that the cache trades only 2.5% of Cost decrease for about 20× speedup (Section 6.4).

6 Evaluation

This section validates the effectiveness of the solving algorithm of PAMOS through multi-dimensional experiments. The evaluation spans four levels: gap analysis relative to MIP optimal solutions, comparison with heuristic baselines, ablation and sensitivity analysis of key components, and production deployment results.

6.1 Experimental Setup

Dataset. The benchmark consists of 631 real scheduling problems from Alibaba Cloud's public-cloud production environment, covering most scheduling domains and dozens of rule combinations. Problem sizes range from 20 to 9,000 PMs and from 50 to 80,000 VMs (Figure 6). The workload comprises third-party tenant VMs with heavily skewed lifetimes (minutes to years, median about 1 hour) and flavors spanning 1 to hundreds of vCPUs (median about 4) and 1 GB to several TB of memory (median about 16 GB). Solving time grows with problem scale: most problems finish within 100 seconds and all within 500 seconds, within the minute-scale SLO.

Evaluation metrics. Three metrics are evaluated: (1) solving time (s), measuring algorithm efficiency; (2) Cost decrease, measuring optimization effectiveness; (3) migration count, measuring stability risk and cost-decreasing efficiency. Unless noted, Cost decrease and migration count are relative to PAMOS (=100%); Table 3 instead uses Standard MIP's final-state Cost decrease as 100%. For two-stage methods we report both the *final-state* Cost decrease and the *path-aware* Cost decrease after path construction; for end-to-end methods the two coincide.

Experimental environment. All experiments are executed on servers equipped with Intel Xeon processors and 64 GB of memory. PAMOS uses its default configuration: search depth $l = 1$, early-stopping parameter $\alpha = 3\%$, with caching and vectorized acceleration enabled.

6.2 Comparison with MIP Optimal Solutions

This experiment quantifies the gap between the algorithm of PAMOS and the theoretical optimum by comparing against MIP models solved with MindOpt solver [31].

Two MIP baselines are established [42]. Standard MIP solves for the final placement. It decides only *where* each VM ends up, which yields the optimal final allocation without considering migration-path feasibility. Path-aware MIP additionally represents migration order and source–destination pairs, yielding a path-aware optimal solution. The variable

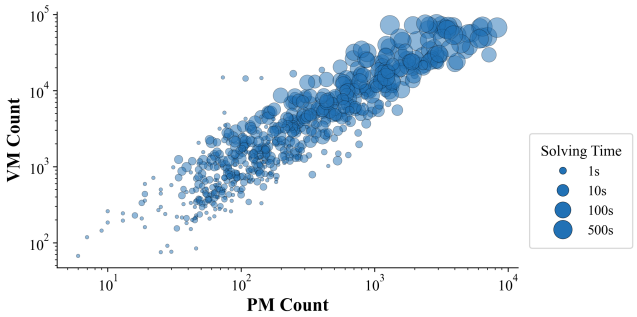


Figure 6. Dataset problem statistics. Each bubble represents one problem, with axes showing PM and VM counts (log scale) and bubble size encoding solving time.

Table 3. Comparison with MIP solutions. Cost decrease values are relative to the Standard MIP final-state cost decrease. Timeout indicates the solver times out after 6 hrs.

Scale (PM×VM)	Algorithm	Solve time	Final-State Cost decr.	Path-Aware Cost decr.
10×70	Standard MIP	~30 min	100%	25.3%
	Path-aware MIP	~3 h	80.6%	80.6%
	PAMOS	0.31 s	78.2%	78.2%
30×400	Standard MIP	~2 h	100%	21.4%
	Path-aware MIP	Timeout	N/A	N/A
	PAMOS	0.52 s	73.1%	73.1%
50×600	Standard MIP	Timeout	N/A	N/A
	Path-aware MIP	Timeout	N/A	N/A
	PAMOS	0.77 s	N/A	N/A

count grows from $O(|\mathcal{I}| \cdot |\mathcal{B}|)$ to $O(|\mathcal{I}|^2 \cdot |\mathcal{B}|^2)$, making it feasible only at very small scales.

In order to fill the gap between final placement and feasible migration paths, a path construction procedure is applied to the standard MIP placement solutions to convert the target placement into an executable migration sequence while enforcing migration-path feasibility.

Path construction algorithm. The path construction procedure iteratively attempts direct migrations $i \to \phi^*(i)$ for each Item whose target differs from its initial placement, repeating until convergence (no further feasible migrations exist). Under the single-migration constraint this is the strongest possible target-preserving construction: the only target-preserving move is the direct one, and any non-target intermediate move irreversibly consumes an Item's single migration opportunity and deviates the final state from the first-stage optimum. Any effectiveness loss after path construction is therefore attributable to the two-stage paradigm rather than to a weak path builder.

Due to the limited scalability of the MIP baselines, we evaluate PAMOS against the MIP models only on small-scale problems, using two metrics: cost decrease after final placement solving (final-state cost decrease) and cost decrease after path construction.

Table 3 reveals two findings. First, Standard MIP's theoretical optimum is largely unreachable: enforcing migration-path feasibility via path construction collapses its realized

effectiveness to only 25.3% and 21.4% at the two scales (consistent with Section 6.3). Second, the path-aware MIP is the appropriate reference point but is tractable only at 10×70, where PAMOS reaches 97.0% of its Cost decrease (78.2% vs. 80.6%) with four orders of magnitude faster solving, showing that PAMOS operates close to the path-aware optimum, at least at the scales where exact verification is possible. This indicates that most of PAMOS's gap to Standard MIP comes from enforcing migration-path feasibility rather than solely from heuristic suboptimality. The path-aware MIP runtime (about 3 hours at 10×70, timeout at 30×400) matches the model-size growth of Section 2.3 and cannot meet the minute-scale SLO, whereas PAMOS achieves sub-second solving time.

6.3 Comparison with Heuristic Baselines

As discussed in Section 8, no existing system simultaneously handles the multi-objective, multi-relationship problem definition and migration-path planning addressed in this paper. We therefore construct four representative baselines following established practice [28], each embodying a distinct algorithmic paradigm. The first three follow a common two-stage framework: *final-state solving* computes a target VM placement, then *path construction* attempts to convert the target placement into an executable migration sequence. This framework reflects the predominant approach in existing work, which computes a target allocation without considering migration-path feasibility.

BFD + Path Construction. All migratable Items are removed from their Boxes and repacked using the Best Fit Decreasing (BFD) algorithm—Items are sorted by resource consumption in descending order and each is placed on the feasible Box with the lowest accumulated Cost. This baseline represents the classical bin packing approach.

Partitioned MIP + Path Construction. The problem is partitioned into subproblems of manageable size [34], and each subproblem is solved independently using MIP to obtain the final placement. This baseline directly instantiates the exact-optimization rescheduling paradigm employed by DCM [38], with partitioning applied to achieve tractable solving times at production scale.

Box-centric Greedy + Path Construction. Inspired by the Box-centric optimization paradigm [36], this baseline iterates over Boxes and greedily packs each Box with the Items that minimize the accumulated Cost. This baseline represents the box-centric greedy packing approach.

In addition to the three two-stage baselines, an end-to-end migration-based heuristic is included for comparison:

Greedy Migration. All migratable Items are sorted in descending order of their Cost contribution, and each Item is greedily migrated to the feasible Box with the lowest Cost. This baseline shares the migration-action-based approach of PAMOS but lacks potential-score-guided priority ordering and the deep-search capability of passive migration.

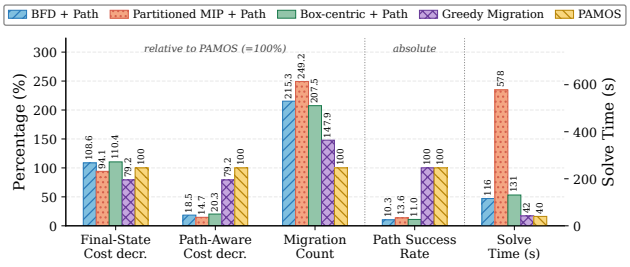


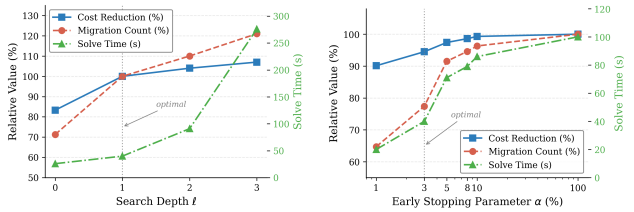
Figure 7. Comparison with heuristic baselines across 631 problems. Cost decrease (final-state and path-aware) and migration count are relative to PAMOS (=100%); the planning-time path success rate is an absolute percentage; solve time is in seconds (right axis).

Figure 7 reveals several important findings.

Path construction causes substantial loss. The three two-stage baselines all achieve competitive or even superior final-state Cost decrease (up to 110.4% of PAMOS), yet their path-aware Cost decrease after path construction collapses to merely 14.7–20.3%. The root cause is that these methods compute target placements without considering migration-path feasibility, so most planned migrations cannot be realized and must be abandoned. This dramatic gap directly validates the design decision in this paper to treat migration-path feasibility as a first-class constraint throughout the solving process.

Why two-stage plans lose so much. To quantify this gap, we define the *planning-time path success rate* as the fraction of planned migrations retained in the executable path after path construction. Figure 7 reports 10.3% for BFD, 13.6% for Partitioned MIP, and 11.0% for Box-centric Greedy; Greedy Migration and PAMOS are 100% by construction. The mechanism is structural: a two-stage method optimizes the target placement without considering its reachability from the initial placement, effectively repacking the cluster from scratch and producing many mutually dependent migrations. The resulting dense migration dependencies amplify the infeasibility mechanisms in Section 2.3, leaving only a small fraction of the planned migrations executable and sharply reducing the realized Cost decrease. However, path success rate and realized Cost decrease are not linearly related: migrations provide unequal Cost reductions, and some benefits require coordinated groups of moves, so the Cost loss depends on which migrations fail. We therefore report both metrics separately.

Partitioned MIP performs worst despite exact solving. Partitioned MIP achieves only 94.1% final-state Cost decrease because the fine-grained partitioning severs the global relationship among VM–VM and VM–PM, severely



(a) Search depth. **(b)** Early-stopping parameter.

Figure 8. Sensitivity of PAMOS to (a) the passive-migration search depth l ($l = 0$ performs only active migration) and (b) the early-stopping parameter α. Cost decrease and migration count (left axis) are relative to $l = 1$ in (a) and to $\alpha = 100\%$ in (b); solve time (right axis) is in seconds. The dotted line marks the default ($l = 1$, $\alpha = 3\%$), which trades a small fraction of Cost decrease for a large reduction in solve time.

compromising global optimization quality. After path construction, its path-aware result drops to the lowest among all baselines, while consuming the longest solving time and the most migrations. This result demonstrates that partitioned exact solving is poorly suited to our problems.

Greedy Migration validates the migration-based paradigm. The Greedy Migration baseline shares PAMOS's end-to-end approach and avoids the catastrophic path-construction loss, reaching 79.2% of PAMOS's effectiveness. Even this simple end-to-end method significantly outperforms every evaluated two-stage baseline on the path-aware result, providing direct evidence that end-to-end generation is better suited to our setting.

6.4 Ablation and Sensitivity Experiments

This section performs ablation and sensitivity analysis on key components and configuration parameters to validate the contribution of each design decision.

Search depth. The search depth l controls the recursion depth of passive migration. $l = 0$ means only active migration is performed.

Cost decrease improves with depth but with diminishing marginal returns (Figure 8a): from l=0 to 1 it gains 16.7%, from 1 to 2 only 4%, while solving time and migration count grow sharply. Deeper search covers more opportunities through longer cascades but adds stability risk, so $l = 1$ is the default.

Early-stopping parameter. The early-stopping parameter α controls how far the search proceeds down the candidate sequence before stopping early (Section 5.5.1).

As shown in Figure 8b, at $\alpha = 3\%$, the algorithm achieves 94.5% of the optimization effectiveness with only 40% of the solving time, representing the Pareto-optimal point in the efficiency–effectiveness trade-off. This is the default setting in production.

Table 4. Ablation of acceleration mechanisms and scoring strategies. Each row disables one component of full PAMOS (last row). Solve time is in seconds; Cost decrease and migration count are relative to full PAMOS (=100%).

Configuration	Solve time	Cost decr.	Mig. count
w/o Level Cache	814 s	102.5%	103.4%
w/o Vectorized Cost Comp.	120 s	99.8%	99.9%
w/o Item potential scoring	164 s	78.5%	170.0%
w/o Active-mig. Box scoring	63 s	52.5%	104.8%
w/o Passive-mig. Box scoring	57 s	88.9%	129.8%
PAMOS (full)	40 s	100%	100%

Acceleration mechanisms and scoring strategies. Table 4 shows the impact of disabling the acceleration mechanisms and replacing each scoring strategy with random shuffle to validate their contribution.

The cache gives about 20× speedup for only a 2.5% Cost-decrease difference. Vectorization provides about 3× speedup; since it only changes how Cost is computed without affecting the solving strategy, its tiny effectiveness fluctuation comes from randomness in sorting.

All three scoring strategies contribute significantly. Item potential scoring primarily affects search efficiency: without it, the algorithm requires 4× the time and 70% more migrations to achieve comparable effectiveness. Active-migration Box scoring has the greatest impact on optimization effectiveness because it directly determines the migration destination of each Item. The primary contribution of passive-migration Box scoring lies in identifying paths that achieve objectives through shorter migration chains; without it, the migration count increases significantly.

6.5 Production Deployment Results

PAMOS has run within Alibaba Cloud for over eight years, processing thousands of tasks daily across hundreds of scheduling domains. We report production metrics for the three scenarios of Section 2, drawn from all daily tasks over the most recent month (per-task change, before vs. after each run). The stakes differ by scenario: a one-point packing-density gain frees dozens of PMs; each dispersion non-compliant instance is a blast-radius risk, so even a 0.1-point gain removes hundreds of at-risk instances across a large cluster; and co-locating long-lived VMs curbs future fragmentation as short-lived VMs are released [3, 29, 43].

As shown in Figure 9, tuning improves all three metrics by mean values of 1.41, 0.12, and 0.97 percentage points (pp). These gains are sustained daily: online scheduling continuously introduces new placement debt, so each daily run encounters fresh optimization opportunities and produces

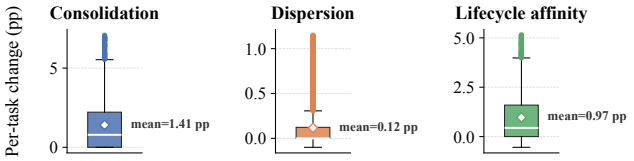


Figure 9. Production deployment results: per-task metric improvements over one month of daily tasks. Boxes span the interquartile range (IQR), the white line marks the median, the diamond the mean; whiskers follow the Tukey convention (1.5×IQR) and circles mark tasks beyond them. The y-axis reports per-task changes in percentage points (pp).

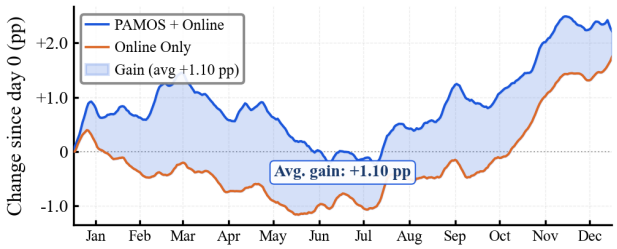


Figure 10. Longitudinal simulation over one year of replayed production traces: change in packing density since day 0 (pp), with and without daily PAMOS tuning; the shaded area is PAMOS's gain over online-only scheduling (average +1.10 pp). Only relative changes are shown.

consistent gains. Notably, the optimization objectives of dispersion and the other two scenarios are inherently opposing—it spreads VMs apart while consolidation and affinity cluster them—yet all three competing metrics improve at once, showing that the unified cost framework balances conflicting objectives in sustained production operation.

Longitudinal simulation analysis. To quantify the sustained value of offline scheduling over a longer time horizon, we replay nearly one year of real request traces from a production cluster through a high-fidelity simulator that faithfully models the online scheduling events. Two configurations are compared: (1) PAMOS + online scheduling, where PAMOS performs daily tuning in addition to the online scheduler, and (2) online scheduling only, where no offline scheduling is performed.

Figure 10 plots the change in packing density since day 0. The PAMOS-augmented curve stays above the online-only curve on every day of the year, with an average gap of +1.10 pp. This demonstrates that placement debt is not a one-time phenomenon but accumulates continuously through sub-optimal online placements and VM lifecycle events, and that daily offline scheduling provides a persistent corrective effect rather than a diminishing one-time improvement. The other two scenarios show similar trends and are omitted for brevity.

7 Operational Experience

This section distills three lessons from eight years of production operation, reviews the alternative solving approaches we explored, and discusses limitations.

7.1 Lessons from Deployment

Lesson 1: path feasibility is learned the hard way. PAMOS initially followed exactly the two-stage paradigm of Section 6.3: a partitioned MIP computed the target placement, and path construction turned it into migrations. Those "optimal" placements routinely lost most of their benefit, because many planned migrations proved infeasible and were abandoned. Consolidation suffered most: a PM pays off only when every VM leaves it, so one stranded VM voids the migrations spent on its neighbors—many VMs moved, no PM freed, stability cost for zero gain. This drove the rebuild around end-to-end path generation; the evaluation mirrors it, with Partitioned MIP realizing only 14.7% of PAMOS's Cost decrease at 14× the solving time (578 s versus 40 s, Figure 7).

Lesson 2: plans must be fast to stay valid. While a plan is computed and executed, online scheduling keeps changing the cluster, so it drifts from the snapshot the plan was built on and migrations start failing. We therefore bound planning by the minute-scale SLO (Section 1) and tag the PMs in an active plan so the online scheduler lowers their priority while it executes; the production execution success rate then exceeds 90%.

Lesson 3: weight governance takes real work. The additive Cost model makes rules easy to write but shifts effort to governing their weights, since rules from different teams interact through one objective. We tier objectives by priority and separate adjacent tiers by orders of magnitude so lower tiers cannot override higher ones; when a task produces unexpected migrations, we trace the executed path to attribute them to the responsible rules; and common weights are tuned by black-box optimization on simulated tasks, with new rules gated by canary release and rollback (Section 4.5). This governance is the price of the DSL's ease of use and remains a continuing investment.

7.2 Alternative Approaches

Other than the current design, we explored several alternative solving paradigms; none of them became the production solver.

Simulated annealing. Simulated annealing (SA) is a natural candidate for escaping local optima, but it proved impractical under migration constraints for two reasons. First, SA randomly accepts moves that worsen the objective; in our setting, each such "bad" move irreversibly consumes one migration from the budget K, leaving fewer moves for genuinely beneficial migrations. Second, uniform random exploration of the vast $O(|\mathcal{I}| \times |\mathcal{B}|)$ search space fails to

prioritize high-value migrations, so convergence far exceeds the minute-scale SLO at production scale. This is consistent with Rebalancer's findings [26], where SA also failed to outperform structured local search in both quality and runtime.

Reinforcement learning. Reinforcement learning (RL) has been applied to VM rescheduling [11], but three factors made it impractical for us: (1) the state space involves variable resource dimensions, NUMA topologies, and dynamic relationships that resist fixed-dimension encoding; (2) different tasks define different Cost types and ranges, making reward design non-transferable; and (3) clusters vary enormously in scale, precluding a single model across thousands of scheduling domains.

7.3 Limitations

PAMOS has three limitations. First, its DSL cannot express time-varying state within one run; we retain this restriction as operators predominantly use time-invariant rules with a small grammar. Second, PAMOS is heuristic: the 97.0% result holds only at the 10×70 scale where exact comparison is tractable, and we make no optimality claim at production scale. Finally, concurrent cluster changes may invalidate otherwise feasible moves; the priority-tagging mechanism (Lesson 2) mitigates but cannot eliminate this risk.

8 Related Work

VM placement. The VM placement problem is a variant of the vector bin packing problem, and classical algorithms such as First Fit and Best Fit have been widely applied. In industrial practice, the Borg system [41] and the Twine system [39] demonstrate large-scale cluster scheduling implementations but primarily focus on the online scheduling phase rather than offline global re-optimization. Recent ML-based work [3, 29] targets single-placement decisions during online scheduling and differs fundamentally from the offline batch rescheduling problem addressed here.

Cluster management. The most closely related work falls within the cluster management domain [2, 5, 6, 8–10, 12, 13, 15–17, 20, 21, 23, 24, 32, 33, 35, 37–41]. DCM [38] models rescheduling as an optimization problem and solves it with an MIP solver, offering strong expressiveness but facing scalability challenges and ignoring migration-path feasibility. Wrasse [36] leverages GPU parallelism to accelerate VM allocation; however, its optimization objective is limited and it does not consider migration paths. Rebalancer [26] provides the most comprehensive modeling and solving framework for generalized bin packing, supporting a variety of Box-centric constraints and objectives. It takes migrations into account, but its `MinimizeMovementSpec` only bounds total moves rather than per-VM migration counts, and its Box-centric paradigm and expression-graph construction cannot handle the fully connected Item–Item relationships and NUMA topology adaptation required by our problem definition.

Migration planning and reconfiguration. Entropy and BtrPlace compute target reconfigurations under placement constraints [18, 19], whereas CQNCR optimizes migration order for a given initial/target configuration and migration set [4]. PAMOS instead jointly chooses migration actions and their order while optimizing a multi-objective Cost model under one-move-per-VM and intermediate-state constraints.

Heuristic methods. The 2012 ROADEF/EURO Challenge [1] addressed a machine reassignment problem involving heuristic algorithms for rescheduling processes across machines. The multi-objective problem definition shares some similarities with the present work, and some contestants employed heuristic solving algorithms [14, 22, 30]. However, the competition involved smaller problem scales, and the optimization objectives differ substantially from those defined in this system, making those algorithms difficult to apply to the problems defined herein.

9 Conclusion

This paper presents PAMOS, an offline VM scheduling system that directly generates migration paths under production stability constraints. Across 631 production problems, after path construction the three two-stage baselines retain only 10.3–13.6% of their proposed migrations and realize only 14.7–20.3% of PAMOS's Cost decrease. PAMOS avoids this loss through a unified cost-based modeling framework and an Item-centric heuristic with active and passive migration. At the scale where exact comparison is tractable, PAMOS achieves 97.0% of the path-aware MIP solver's Cost decrease with four orders of magnitude speedup. Eight years of deployment at Alibaba Cloud, processing thousands of daily tasks, demonstrates its practical effectiveness.

Acknowledgments

We sincerely thank our shepherd and the anonymous reviewers for their valuable feedback, and all teams within Alibaba Cloud that contributed to the deployment of PAMOS.

References

[1] H Murat Afsar, Christian Artigues, Eric Bourreau, and Safia Kedad-Sidhoum. 2016. *Machine reassignment problem: the ROADEF/EURO Challenge 2012.* Technical Report.

[2] Faraz Ahmad, Srimat T Chakradhar, Anand Raghunathan, and T N Vijaykumar. 2012. Tarazu: optimizing mapreduce on heterogeneous clusters. *ACM SIGARCH Computer Architecture News* 40, 1 (2012), 61–74.

[3] Hugo Barbalho, Patricia Kovaleski, Beibin Li, Luke Marshall, Marco Molinaro, Abhisek Pan, Eli Cortez, Matheus Leao, Harsh Patwari, Zuzu Tang, Tamires Santos, Larissa Rozales Gonçalves, David Dion, Thomas Moscibroda, and Ishai Menache. 2023. Virtual Machine Allocation with Lifetime Predictions. In *MLSys*.

[4] Md. Faizul Bari, Mohamed Faten Zhani, Qi Zhang, Reaz Ahmed, and Raouf Boutaba. 2014. CQNCR: Optimal VM Migration Planning in Cloud Data Centers. In *2014 IFIP Networking Conference*. 1–9. doi:10.1109/IFIPNetworking.2014.6857120

[5] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: scalable and coordinated scheduling for cloud-scale computing. In *OSDI*, Vol. 14. 285–300.

[6] Zuzhi Chen, Fuxin Jiang, Binbin Chen, Yu Li, Yunkai Zhang, Chao Huang, Rui Yang, Fan Jiang, Jianjun Chen, Wu Xiang, Guozhu Cheng, Rui Shi, Ning Ma, Wei Zhang, and Tieying Zhang. 2024. Resource Allocation with Service Affinity in Large-Scale Cloud Environments. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 5280–5293. doi:10.1109/ICDE60146.2024.00397

[7] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live Migration of Virtual Machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, USA, 273–286.

[8] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 153–167.

[9] Carlo Curino, Subru Krishnan, Konstantinos Karanasos, Sriram Rao, Giovanni Matteo Fumarola, Botong Huang, Kishore Chaliparambil, Arun Suresh, Young Chen, Solom Heddaya, et al. 2019. Hydra: a federated resource manager for data-center scale analytics. In *NSDI*. 177–192.

[10] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices* 49, 4 (2014), 127–144.

[11] Xianzhong Ding, Yunkai Zhang, Binbin Chen, Donghao Ying, Tieying Zhang, Jianjun Chen, Lei Zhang, Alberto Cerpa, and Wan Du. 2025. Towards VM Rescheduling Optimization Through Deep Reinforcement Learning. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*. Association for Computing Machinery, 686–701. doi:10.1145/3689031.3717476

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. 2019. The design and operation of cloudlab. In *USENIX Annual Technical Conference*. 1–14.

[13] Panagiotis Garefalakis, Konstantinos Karanasos, Peter Pietzuch, Arun Suresh, and Sriram Rao. 2018. Medea: scheduling of long running applications in shared production clusters. In *Proceedings of the Thirteenth EuroSys Conference*. 1–13.

[14] Haris Gavranovic, Mirsad Buljubasic, and Emir Demirovic. 2012. Variable neighborhood search for google machine reassignment problem. *Electronic Notes in Discrete Mathematics* 39 (2012), 209–216.

[15] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. 2013. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*. 365–378.

[16] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multiresource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 455–466.

[17] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. 2020. Protean: Vm allocation service at scale. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 845–861.

[18] Fabien Hermenier, Julia Lawall, and Gilles Muller. 2013. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. *IEEE Transactions on Dependable and Secure Computing* 10, 5 (2013), 273–286. doi:10.1109/TDSC.2013.5

[19] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. 2009. Entropy: A Consolidation Manager for Clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)*. Association for Computing Machinery, New York, NY, USA, 41–50. doi:10.1145/1508293.1508300

[20] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, Vol. 11. 22–22.

[21] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. 2009. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. 261–276.

[22] W Jaskowski, Marcin Szubert, and Piotr Gawron. 2016. A hybrid mip-based large neighborhood search heuristic for solving the machine reassignment problem. *Annals of Operations Research* 242 (2016), 33–62.

[23] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. 2016. Morpheus: Towards automated slos for enterprise clusters. In *OSDI*. 117–134.

[24] Karthik Kambatla, Vamsee Yarlagadda, Íñigo Goiri, and Ananth Grama. 2018. Ubis: Utilization-aware cluster scheduling. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 358–367.

[25] Xiaodi Ke, Cong Guo, Siqi Ji, Shane Bergsma, Zhenhua Hu, and Lei Guo. 2021. Fundy: A Scalable and Extensible Resource Manager for Cloud Resources. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. 540–550. doi:10.1109/CLOUD53861.2021.00070

[26] Neeraj Kumar, Pol Mauri Ruiz, Vijay Menon, Igor Kabiljo, Mayank Pundir, Andrew Newell, Daniel Lee, Liyuan Wang, and Chunqiang Tang. 2024. Optimizing Resource Allocation in Hyperscale Datacenters: Scalability, Usability, and Experiences. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 507–528.

[27] Christoph Lameter. 2013. NUMA (Non-Uniform Memory Access): An Overview. *Queue* 11, 7 (2013), 40–51. doi:10.1145/2508834.2513149

[28] Sangmin Lee, Rina Panigrahy, Vijayan Prabhakaran, Venugopalan Ramasubramanian, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. 2011. Validating Heuristics for Virtual Machines Consolidation. (2011).

[29] Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, YING LIU, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, and Martin Maas. 2025. LAVA: Lifetime-Aware VM Allocation with Learned Distributions and Adaptation to Mispredictions. In *Proceedings of Machine Learning and Systems*, Vol. 7.

[30] Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. 2012. Comparing solution methods for the machine reassignment problem. In *Principles and Practice of Constraint Programming: 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*. Springer, 782–797.

[31] MindOpt Inc. 2024. MindOpt Optimizer. https://mindopt.tech/portal. [Online].

[32] Pulkit A Misra, Íñigo Goiri, Jason Kace, and Ricardo Bianchini. 2017. Scaling distributed file systems in resource-harvesting datacenters. In *USENIX Annual Technical Conference*. 799–811.

[33] Aravind Narayanan, Elisa Shibley, and Mayank Pundir. 2020. Fault tolerance through optimal workload placement. https://engineering.fb.com/2020/09/08/data-center-engineering/fault-tolerance-through-optimal-workload-placement/.

[34] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 521–537. doi:10.1145/3477132.3483588

[35] Andrew Newell, Dimitrios Skarlatos, Jingyuan Fan, Pavan Kumar, Maxim Khutornenko, Mayank Pundir, Yirui Zhang, Mingjun Zhang, Yuanlai Liu, Linh Le, et al. 2021. RAS: continuously optimized region-wide datacenter resource allocation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*. 505–520.

[36] Anshul Rai, Ranjita Bhagwan, and Saikat Guha. 2012. Generalized resource allocation for the cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing*. 1–12.

[37] Bikash Sharma, Victor Chudnovsky, Joseph L Hellerstein, Rasekh Rifaat, and Chita R Das. 2011. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–14.

[38] Lalith Suresh, João Loff, Faria Kalim, Sangeetha Abdu Jyothi, Nina Narodytska, Leonid Ryzhyk, Sahan Gamage, Brian Oki, Pranshu Jain, and Michael Gasch. 2020. Building scalable and flexible cluster managers using declarative programming. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 827–844.

[39] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matt Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: A unified cluster management system for shared infrastructure. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 787–803.

[40] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. 2016. TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*. 1–16.

[41] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–17.

[42] Juan Pablo Vielma. 2015. Mixed integer linear programming formulation techniques. *SIAM Rev.* 57, 1 (2015), 3–57. doi:10.1137/130915303

[43] Zhengtong Zhang, Zihan Xu, Zhidong Hu, Yanbo Shan, Fei Peng, Suhong Chen, Kaiyuan Shen, Xiangyun Kong, Handu Ding, Bing He, and Binda Ma. 2026. DVLA: Dynamic VM Lifetime Aware Scheduling for Drifting Lifetime Distributions and Long-Lived VM Placement Debt (Operational Systems). In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. USENIX Association, Seattle, WA, 2149–2168. https://www.usenix.org/conference/osdi26/presentation/zhang-zhengtong

[44] Jiazhen Zhu, Wenda Tang, Xianglong Meng, Nan Gong, Tianxiang Ai, Guanghui Li, Bin Yu, and Xin Yang. 2024. PheCon: Fine-Grained VM Consolidation with Nimble Resource Defragmentation in Public Cloud Platforms. In *Proceedings of the 53rd International Conference on Parallel Processing (ICPP '24)*. 712–721. doi:10.1145/3673038.3673139