Escaping the makespan plateau: annealed power-mean guidance for task scheduling in heterogeneous cloud environments

[1] Chandrasekhar Panda [2]Saswat Swain [3]Pravat Satpathy [4]Sonia Mohapatra
[1,2]Ph.D Scholar, [3,4]R & D Division
[1,2]KSOM - KIIT Deemed to be University ,[3,4]Tekstein Solutions,India
[1]contact@chandrasekarpanda.com [2]saswat.mck@gmail.com [3]pravat@teksteinsolutions.com [4]sonia@teksteinsolutions.com

Abstract—Metaheuristic load balancers for cloud data centres are almost always driven by the makespan of the scheduling batch. We show that this objective is degenerate as a search signal. Because the makespan is a maximum, migrating a task away from the critical virtual machine merely promotes the runner-up, so the objective does not change and the search stalls. Measured over the full migration neighbourhood, 16.43 per cent of single-task migrations leave the makespan unchanged, and steepest descent under the makespan halts after only 2.8 accepted moves. We replace the makespan during the search by the normalised power mean of the machine completion times and anneal its exponent geometrically. The substitution is safe: the power mean is proved to bracket the makespan within a factor equal to the number of machines raised to the reciprocal of the exponent, so the terminal exponent controls the worst-case loss. Under this guidance the same descent sustains 162.5 moves and reaches a 6.83 per cent lower makespan. The resulting scheduler, APEX-BC, is a memetic bee colony in which each food source is refined by an incremental power-mean descent step. On an inconsistent expected-time-to-compute model with heterogeneous virtual machines, APEX-BC reduces the makespan by 5.81 per cent relative to the strongest heuristic and halves the gap to a linear-programming lower bound from 12.08 to 5.40 per cent, while artificial bee colony, particle swarm and genetic baselines at an identical evaluation budget remain statistically indistinguishable from their seed. We also report two negative results. Under the uniform speed model used by most of the literature the problem is degenerate: the longest-processing-time rule lies 0.65 per cent from the lower bound and no search method improves on it, which questions gains previously reported in that model. In an online rolling-horizon setting the makespan is arrival-bound and non-discriminating, and no metaheuristic, including ours, matches the Sufferage heuristic on response time. All results are simulation results.

Index Terms—Cloud computing, task scheduling, load balancing, artificial bee colony, power mean, makespan, virtual machine, heterogeneous computing.

I. INTRODUCTION

Infrastructure-as-a-service platforms multiplex user workloads onto pools of virtual machines (VMs) whose capacities and resource profiles differ widely. Deciding which VM should execute which task is the load-balancing problem, and it governs completion time, tail latency, energy draw and the rate at which service-level agreements are met. The problem is a special case of scheduling independent tasks on unrelated parallel machines, which is strongly NP-hard [4], so practical systems rely on heuristics or on metaheuristic search.

A large literature applies population-based metaheuristics to this problem. Honey-bee foraging [10], artificial bee colony [15], particle swarm optimisation [19] and genetic algorithms have all been reported to improve on classical heuristics such as Min-Min, Max-Min and Sufferage [2], [6]. Almost without exception these methods use the batch makespan as the fitness function.

This paper argues, and then demonstrates, that the makespan is a poor *search* signal even when it is the right *reporting* metric. The makespan is a maximum over machine completion times. Consider a migration of one task between two machines that are not the critical one. The maximum is unchanged, so the move is fitness-neutral and the search receives no gradient. Worse, the only moves that can reduce the maximum are those that unload the single critical machine, and such a move usually promotes the runner-up to critical status, leaving the objective flat. The search therefore stalls almost immediately in a vast plateau.

Our remedy is to keep the makespan as the reported metric but to replace it during the search by the normalised power mean of the completion-time vector, whose exponent is annealed from a small value, where the landscape is smooth and globally informative, to a large value, where the surrogate is faithful to the makespan. The substitution is not heuristic: the power mean brackets the makespan within a factor that we quantify exactly, so the terminal exponent is a tunable bound on the worst-case loss.

The contributions of this paper are as follows.

(1) A diagnosis of the plateau. We quantify, on the full single-task migration neighbourhood, the fraction of fitness-neutral

moves under the makespan and under power-mean surrogates, and we measure how long steepest descent survives under each. The makespan admits 16.43 per cent neutral moves and stalls after 2.8 moves; the power mean admits none and sustains 162.5.

(2) A bounded surrogate and an annealing schedule. We prove that the normalised power mean of order p over m machines bracket the makespan within a factor m raised to the power $1/p$, and derive from this a principled geometric annealing schedule.

(3) APEX-BC. A memetic bee colony in which every food source is refined by a steepest-descent step under the annealed surrogate, evaluated incrementally in constant time per candidate move and restricted to a candidate list drawn from the most loaded machines.

(4) An equal-budget, seeded, statistically controlled evaluation, including a linear-programming lower bound that lets us report absolute optimality gaps rather than only relative comparisons.

(5) Two negative results that we consider as important as the positive one: the uniform speed model widely used in this literature is degenerate, and no metaheuristic, ours included, beats simple heuristics in an online rolling-horizon regime.

II. RELATED WORK

Classical mapping heuristics for independent tasks on heterogeneous machines were consolidated by Braun et al. [2], who compared eleven static methods on expected-time-to-compute (ETC) matrices, and by Maheswaran et al. [6] for the dynamic case. Min-Min, Max-Min and Sufferage remain the reference points. On identical machines the longest-processing-time rule of Graham [5] is a 4/3-approximation, and Max-Min reduces to it when machine speeds are proportional. For unrelated machines, Lenstra, Shmoys and Tardos [4] gave a 2-approximation by rounding the linear-programming relaxation; we use that relaxation as a lower bound.

Metaheuristic schedulers for clouds are numerous. Dhinesh Babu and Krishna [10] adapted honey-bee foraging to VM load balancing; Kimpan and Kruekaew [15] combined artificial bee colony [7] with heuristic seeding; Domanal and Reddy [11] proposed a VM-assign balancer; Chen et al. [12] guided Min-Min by user priority; Kumar and Sharma [13] and Adhikari and Amgoth [14] addressed deadlines and elasticity. Madni et al. [16] compared heuristics for IaaS clouds and observed that reported improvements are often small and inconsistent across studies.

Two methodological gaps recur. First, comparisons are frequently made against unseeded metaheuristics or at unequal computational budgets, so the reported margin conflates search quality with initialisation quality; we control for both. Second, execution time is usually modelled as task length divided by VM speed. That model makes the machines *uniform* rather than *unrelated*, and we show in Section VII-B that it removes essentially all the difficulty from the problem. Production traces show pronounced resource heterogeneity and interference between co-located workloads [17], [18], which motivates the inconsistent ETC model we adopt.

Power means are classical [24] and appear in scheduling as the Lp-norm minimisation family, but to our knowledge they have not been used as an *annealed surrogate* to repair the search landscape of makespan-driven cloud schedulers. The memetic structure we use follows Moscato [27].

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. Infrastructure and workload

A data centre offers m virtual machines. VM j has compute capacity cj in millions of instructions per second (MIPS) and belongs to one of five families: general-small, general-medium, compute-optimised, memory-optimised and storage-optimised. A batch of n independent, non-preemptable tasks is to be placed. Task i has length li in millions of instructions (MI) and belongs to one of four resource classes: CPU-bound, memory-bound, I/O-bound or mixed.

Execution time is governed by an affinity coefficient between the task class and the VM family:

$$e\,i\,j \;=\; l\,i \;/\; (\,c\,j \cdot \eta\,[\,k(i),\,t(j)\,] \cdot \xi\,i\,j\,) \tag{1}$$

where $\eta \in (0,1]$ is the affinity efficiency, k(i) the class of task i, t(j) the family of VM j, and ξ a lognormal perturbation with unit median. Because η varies by more than a factor of three across class-family pairs, no VM is fastest for every task class: the resulting ETC matrix is *inconsistent* in the sense of Braun et al. [2]. Setting $\eta \equiv 1$ and $\xi \equiv 1$ recovers the *uniform* model e i j = l i / c j used by most published cloud schedulers; we study both.

B. Objective functions

Let x : {1..n} → {1..m} be an assignment and r j the residual busy time (ready time) of VM j at the dispatch instant. The completion time of VM j is

$$C\,j\,(\,x\,) \;=\; r\,j \;+\; \Sigma\,\{\,i : x(i) = j\,\}\;\; e\,i\,j \tag{2}$$

and the batch makespan, the degree of imbalance and the mean utilisation are

$$C\,max \;=\; max\,j\;\; C\,j\,, \qquad DI \;=\; (\,B\,max - B\,min\,) \,/\, B\,avg \tag{3}$$

where B j denotes the busy time accumulated by VM j. Energy follows the standard linear model [9], in which VM j draws idle power P idle, j and an additional load-proportional term up to P busy, j.

Two evaluation protocols are used. In the *static* protocol all n tasks are available at time zero and r j = 0; this is the setting in which almost all published comparisons are made. In the *online rolling-horizon* protocol tasks arrive over time, the dispatcher accumulates arrivals for a window Δ and schedules the accumulated batch against the residual ready times, which are carried into the next epoch. Within a VM, the tasks of one epoch are executed shortest-processing-time first; this rule is identical for every scheduler, so all measured differences are attributable to the assignment.

IV. THE MAKESPAN PLATEAU

A. Why the maximum is a poor search signal

Fix an assignment x and consider migrating task i from VM u to VM v. Only the coordinates C u and C v of the completion

vector change. If both machines remain strictly below the critical machine after the move, C max is unchanged and the move is *fitness-neutral*. The set of such moves is typically large, and a search that accepts only strict improvements cannot traverse it.

Moreover, when the move does unload the critical machine, the new maximum is the larger of the reduced value and the previous runner-up. In a well-balanced solution the runner-up is close to the maximum, so the achievable decrease per move is bounded by a small gap. Sequences of individually neutral moves are usually required to make progress, and a makespan-guided search cannot navigate them because the intermediate states are not improving.

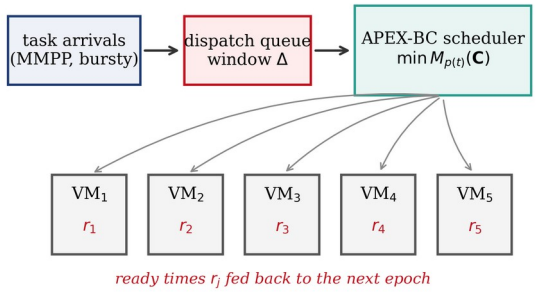

Fig. 1. Rolling-horizon scheduling model. Ready times produced by one dispatch epoch become the initial conditions of the next.

B. A bounded smooth surrogate

For a completion vector C with m strictly positive entries and p ≥ 1, define the normalised power mean

$$M_p(C) = \left((1/m)\cdot \Sigma_j\ C_j^{\wedge} p\right)^{\wedge}(1/p) \quad (4)$$

so that M 1 is the arithmetic mean and M p → C max as p → ∞ [24].

Proposition 1 (bracketing). For every C and every p ≥ 1,

$$M_p(C) \le C_{max}(C) \le m^{\wedge}(1/p)\cdot M_p(C). \quad (5)$$

Proof. Replacing every entry by the largest gives M p ≤ ((1/m) · m · C max^p)^(1/p) = C max. Retaining only the largest entry in the sum gives M p ≥ ((1/m) · C max^p)^(1/p) = m^(−1/p) · C max, which rearranges to the right-hand inequality. ∎

Proposition 2 (approximation transfer). Let x*p minimise M p and x* minimise C max. Then C max(x*p*) ≤ *m^(1/p) · C max(x*).

Proof. By Proposition 1, C max(x*p*) ≤ *m^(1/p) M p(x*p) ≤ m^(1/p) M p(x*) ≤ *m^(1/p) C max(x*), where the middle step uses optimality of x*p for M p. ∎

Proposition 2 turns the exponent into an explicit accuracy dial. For m = 20, the worst-case ratio is 1.206 at p = 16, 1.098 at p = 32 and 1.024 at p = 128. Minimising a smooth surrogate therefore costs at most a few per cent of makespan optimality, and in practice much less.

The surrogate also removes the plateau. Every partial derivative

$$\partial M_p / \partial C_j = (1/m)\cdot (C_j / M_p)^{\wedge}(p-1) \quad (6)$$

is strictly positive, so no machine is invisible to the objective and, generically, no move is fitness-neutral.

C. Annealing the exponent

A small exponent yields a smooth landscape that rewards flattening the whole completion profile but is a loose proxy for the makespan; a large exponent is faithful but nearly as flat as the maximum. We therefore anneal *p* geometrically in S stages as the evaluation budget E is consumed:

$$p(s) = p0\cdot (pT/p0)^{\wedge}(s/(S-1)), \quad s = 0\,..\,S-1 \quad (7)$$

with p0 = 2, pT = 128 and S = 8 throughout. Table I reports the diagnostics that motivate this choice, measured on the full migration neighbourhood from a Sufferage start.

Table I. Search-landscape diagnostics (n = 200, m = 20, 20 seeds, mean ± s.d.)

Guidance	Neutral moves (%)	Descent length	Makespan gain (%)
makespan C max	16.43 ± 11.94	2.8 ± 2.6	−1.96 ± 2.23
M 8	0.00 ± 0.00	—	—
M 16	—	14.4 ± 5.8	−2.31 ± 2.26
M 128	0.29 ± 1.27	—	—
annealed 2→128	—	162.5 ± 11.8	−6.83 ± 2.74

Descent length is the number of accepted steepest-descent moves before no improving move remains. Differences against the makespan column are significant (Wilcoxon, $p \le 8.8\times10^{-5}$).

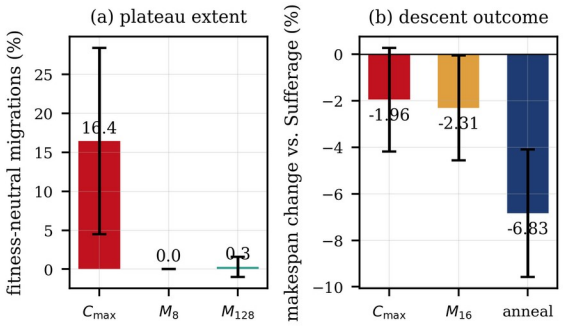

Fig. 2. (a) Fraction of single-task migrations that leave the objective unchanged. (b) Makespan reached by steepest descent from a Sufferage start.

V. THE APEX-BC ALGORITHM

APEX-BC (Annealed P-EXponent Bee Colony) maintains SN food sources, each a complete assignment vector, and refines them by descent under the annealed surrogate.

A. Initialisation

Four sources are seeded with the Sufferage, Max-Min, Min-Min and minimum-completion-time solutions; the remainder are sampled with per-task probabilities proportional to the reciprocal of the row of the ETC matrix, so that fast and well-matched VMs are preferred. Seeding every competing metaheuristic identically is essential for a fair comparison, and Section VII-C shows it is the single largest contributor to final quality.

B. Incremental evaluation and candidate list

A migration changes only two coordinates of C. Writing S = Σ j (C j / σ)^p for a scale σ = C max, the power sum after moving task *i* from *u* to *v* is

$$S' = S - (C\,u/\sigma)^{p} - (C\,v/\sigma)^{p} + ((C\,u - e\,i\,u)/\sigma)^{p} + ((C\,v + e\,i\,v)/\sigma)^{p} \quad (8)$$

which costs constant time per candidate. Restricting source machines to the q most loaded VMs bounds a steepest-descent step to about $q \cdot n/m \cdot m = q \cdot n$ candidate evaluations. We use $q = 2$.

C. Bee phases

In the employed-bee phase each source performs one steepest-descent step under the current exponent. In the onlooker phase, SN additional steps are allocated to sources drawn with probability inversely proportional to their makespan, concentrating effort on promising solutions. A source that admits no improving move for more than *limit* consecutive attempts is subjected to a scout phase implemented as ruin-and-recreate: a random quarter of its tasks are reassigned by the affinity-proportional sampler. The incumbent is the solution of least makespan encountered, ties being broken by the terminal power mean.

Hyperparameters were fixed once on a *tuning* seed set (SN = 8, q = 2, p0 = 2, pT = 128, limit = 4) and never re-tuned; every number reported below comes from a disjoint *test* seed set.

VI. EXPERIMENTAL SETUP

A discrete-event simulator implementing the space-shared execution model of Section III was written in Python. Table II lists the parameters.

Table II. Simulation parameters

Parameter	Value
Task length	five-mode mixture, 1.5×10^4 – 9.0×10^5 MI
Task classes	CPU / memory / I/O / mixed (0.32, 0.24, 0.22, 0.22)
VM capacity	600 – 1600 MIPS, five families
Affinity η	0.30 – 1.00, inconsistent
ETC perturbation ξ	lognormal, σ = 0.10
Power model	45 – 88 W idle, 105 – 215 W peak
Arrivals (online)	Markov-modulated Poisson, burst ×6
Batch sizes	n = 100, 200, 400 (static)
VM count	m = 10, 20, 40, 80
Evaluation budget	500 × n candidate evaluations, all methods
Repetitions	30 seeds (static), 12–15 (online)

Baselines are Round-Robin, Throttled, minimum execution time, minimum completion time, Min-Min, Max-Min and Sufferage, together with artificial bee colony, particle swarm optimisation and a genetic algorithm, each seeded identically and given the same budget. Statistical comparisons use the Wilcoxon signed-rank test with Holm correction [23] and the Vargha-Delaney effect size [22], following the recommendations of García et al. [21]. The linear-programming relaxation of the unrelated-machine formulation [4] provides a lower bound, so we report absolute optimality gaps.

A. Data availability

The Google cluster trace, the GoCJ job-length dataset [25] and the Parallel Workloads Archive were not reachable from the execution environment used for this study. The workload generator is therefore a *calibrated reconstruction* whose magnitude range and right skew reproduce the published characteristics of Google cloud jobs, not the datasets themselves. All results in this paper are simulation results; no hardware measurements are claimed.

VII. RESULTS AND DISCUSSION

A. Static batch scheduling

Table III reports the static protocol on the inconsistent ETC model. APEX-BC attains the lowest makespan at every batch size, and the margin grows with the batch. Crucially, the three conventional metaheuristics are statistically indistinguishable from the Sufferage solution that seeded them: at an equal budget of roughly 2×10^5 evaluations they consume between 2.39 and 2.79 seconds of decision time and return essentially their seed, whereas APEX-BC improves on it in 59.7 milliseconds.

Table III. Static batch, inconsistent ETC, m = 20, 30 seeds. Makespan in seconds; Δ is relative to APEX-BC

Method	n=200	Δ (%)	n=400	Δ (%)	LP gap (%)
Round-Robin	4365.2	+285	7665.8	+248	268.5
Throttled	2600.6	+129	4149.5	+88	99.6
MCT	1572.5	+39	2751.0	+25	31.8
Min-Min	1711.2	+51	2826.2	+28	35.4
Max-Min	1414.3	+25	2910.6	+32	39.6
Sufferage	1201.8	+6.0	2337.2	+6.2	12.1
ABC [15]	1200.6	+5.9	2333.5	+6.0	11.9
PSO [19]	1189.3	+4.9	2326.2	+5.7	11.5
GA	1191.9	+5.1	2333.8	+6.0	11.9
APEX-BC	**1134.0**	**—**	**2201.3**	**—**	**5.4**

LP gap is at n = 400. All APEX-BC comparisons significant after Holm correction ($p \le 1.9\times10^{-8}$); effect size medium at n = 400.

The optimality gap is the more informative statistic. At n = 400 the best heuristic sits 12.08 per cent above the linear-programming bound and the conventional metaheuristics between 11.5 and 11.9 per cent, while APEX-BC reaches 5.40 per cent. Roughly half of the remaining gap is therefore recovered by changing the search signal alone. APEX-BC also achieves the highest mean utilisation (0.987 against 0.975 for Sufferage), the lowest degree of imbalance (0.033 against 0.053) and the lowest energy (1994 Wh against 2102 Wh).

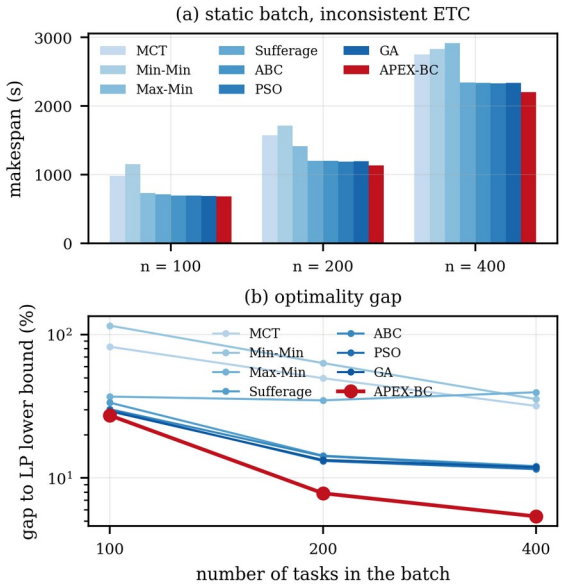

Fig. 3. Static batch on the inconsistent ETC model. (a) Makespan by batch size. (b) Gap to the linear-programming lower bound.

B. The uniform speed model is degenerate

Most published cloud load-balancing studies model execution time as task length divided by VM speed. Under that model the machines are uniform rather than unrelated, and Max-Min degenerates to the longest-processing-time rule. Table IV shows the consequence: Max-Min lands 0.65 per cent from the linear-programming bound, and artificial bee colony, particle swarm, genetic search and APEX-BC all reproduce that value to within 0.02 per cent. No search method can improve on a solution that is already within one per cent of optimal.

Table IV. Gap to the LP lower bound under three speed models (n = 200, m = 20)

Method	uniform	+10% jitter	affinity ETC
Max-Min	0.65	18.52	34.75
Sufferage	3.52	8.65	14.30
ABC	0.65	8.46	14.20
PSO	0.63	7.41	13.14
GA	0.65	7.83	13.35
APEX-BC	**0.65**	**5.80**	**7.82**

Introducing only a ten per cent multiplicative perturbation raises the Max-Min gap from 0.65 to 18.52 per cent, so the degeneracy is fragile as well as complete.

This is a negative result about the experimental practice of the field rather than about any particular algorithm. Improvements reported for metaheuristic schedulers evaluated solely under the uniform model cannot be attributed to better search, because there is no headroom to exploit; they are more plausibly explained by comparison against unseeded initialisations or unequal budgets. It also delimits our own contribution: APEX-BC is worth its complexity only when machine performance is genuinely inconsistent, as production traces indicate [17], [18].

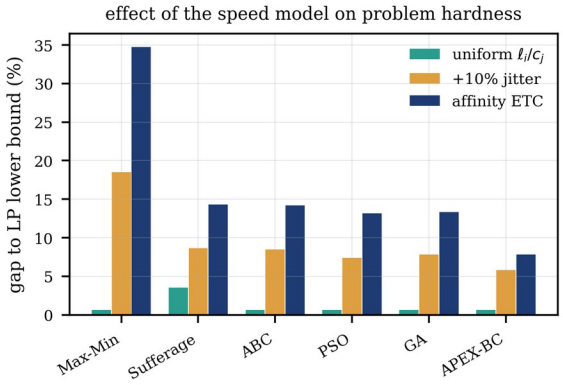

Fig. 4. Problem hardness under three speed models. The uniform model leaves no headroom for any search method.

C. Ablation

Table V isolates the components. Removing heuristic seeding is by far the most damaging change, costing 17.03 per cent; this confirms that a substantial part of the quality of any seeded metaheuristic comes from its seed and not from its search. Replacing the annealed power mean by the plain makespan within otherwise identical machinery costs 3.93 per cent, which is the contribution attributable to the surrogate itself. Annealing contributes a further 0.61 per cent over the best fixed exponent.

Table V. Ablation on the static protocol (n = 200, m = 20, 30 seeds)

Variant	Makespan (s)	Δ vs full (%)
without heuristic seeding	1366.8	+17.03
fixed p = 2	1189.0	+4.62
makespan objective	1180.4	+3.93
fixed p = 8	1165.1	+2.67
fixed p = 128 (no annealing)	1144.5	+0.92
fixed p = 32	1141.0	+0.61
APEX-BC (full)	**1134.0**	—

All differences significant after Holm correction ($p \le 2.4\times10^{-3}$), though the effect size of annealing alone is negligible by the Vargha-Delaney criterion.

We report the ordering as measured rather than as hypothesised. The annealing effect is large in pure descent, where it triples the descent length and quadruples the makespan gain (Table I), but modest inside the full algorithm, because the population and the ruin-and-recreate scout already supply diversification. The honest summary is that the *surrogate* matters a great deal and the *schedule* on the surrogate matters somewhat.

D. Scalability

Figure 5 varies the number of VMs at a fixed batch of 400 tasks. APEX-BC improves on Sufferage by 5.29, 5.42, 2.89 and 3.09 per cent at m = 10, 20, 40 and 80 respectively, while artificial bee colony never exceeds 2.57 per cent and costs two orders of magnitude more decision time. The advantage narrows as m grows, because with 80 VMs the batch provides only five tasks per machine and the assignment becomes coarse; Max-Min becomes competitive in that regime.

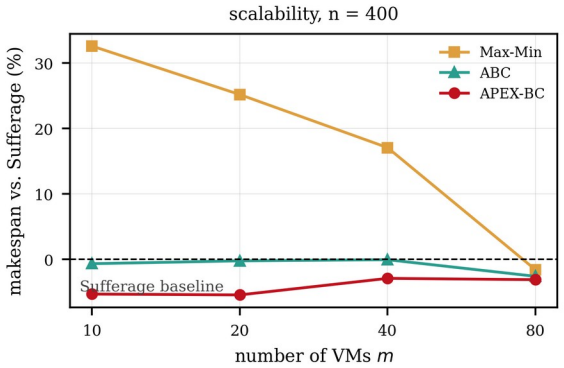

Fig. 5. Makespan relative to Sufferage as the number of VMs varies (n = 400).

E. Online rolling horizon: a negative result

The static protocol is the one used by the literature, but production schedulers operate online. Table VI reports the rolling-horizon protocol with 50 VMs. Two observations stand out, and neither favours metaheuristic search.

First, the makespan ceases to discriminate. Once the pathological Round-Robin and Throttled policies are excluded, every method finishes within roughly a tenth of a per cent of the others, because at high utilisation the completion of the last task is governed by the arrival process rather than by the assignment. Reporting makespan gains in an online setting is therefore close to meaningless, and the discriminating metrics are response time, tail latency and the rate of deadline misses.

Second, on those metrics the simple Sufferage heuristic is the best method at both load levels, and every metaheuristic is worse. APEX-BC is 5.61 per cent worse at load 0.85 and 11.97 per cent worse at load 0.95; artificial bee colony is worse still, by 8.24 and 18.93 per cent. The explanation is that batch makespan, and hence its power-mean surrogate, is simply not aligned with mean response time: minimising the completion profile of a batch says nothing about the order in which individual tasks finish, whereas the greedy earliest-completion rule underlying MCT and Sufferage is directly aligned with it.

Table VI. Online rolling horizon, m = 50, 15 seeds. Response and tail latency in seconds

Method	resp.	p95	SLA (%)	makespan	sched. (s)
Throttled	309.6	632.0	14.21	4971.6	0.02
MCT	144.6	300.3	0.50	3808.2	0.00
Min-Min	145.8	330.1	0.40	3827.2	0.01
Max-Min	145.3	278.9	1.23	3779.7	0.01
Sufferage	138.7	277.4	0.90	3782.9	0.02
ABC	171.1	418.8	2.72	3788.4	4.54
PSO	146.9	307.1	1.09	3786.8	2.26
GA	146.9	307.1	1.09	3786.8	3.52
APEX-BC	**157.6**	**363.3**	**1.99**	**3782.4**	**1.78**

Load 0.95. Sufferage is the champion; all differences against it are significant ($p = 4.9\times10^{-4}$).

The one advantage APEX-BC retains online is cost: it spends 1.78 seconds of cumulative decision time against 4.54 for artificial bee colony, so among metaheuristics it is the least unattractive. But the correct engineering conclusion from Table VI is that a practitioner scheduling an online cloud workload should use Sufferage, not a metaheuristic.

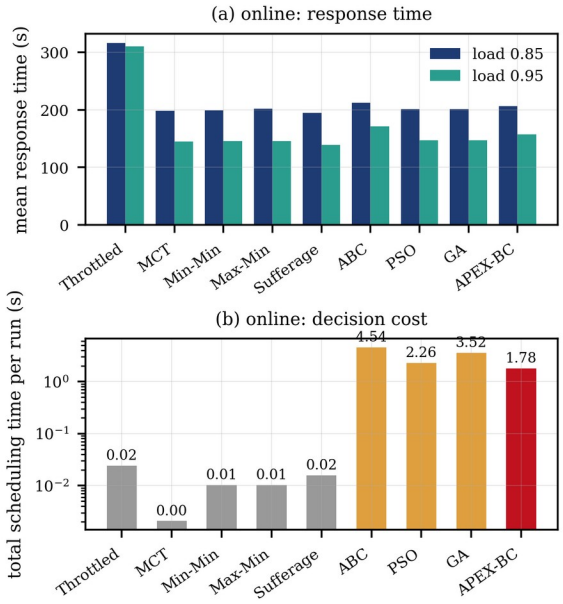

Fig. 6. Online rolling horizon. (a) Mean response time at two load levels. (b) Cumulative scheduling cost, logarithmic scale.

F. Case studies

Two application-flavoured scenarios were examined. In a *flash-sale* scenario (900 tasks, 40 VMs, burst multiplier 12, tight deadlines) Sufferage again leads with 158.7 seconds mean response and 8.09 per cent deadline misses, against 181.5 seconds and 12.90 per cent for APEX-BC and 199.0 seconds and 16.24 per cent for artificial bee colony. In a *scientific batch* scenario (600 tasks, 30 VMs, smooth arrivals, generous deadlines) the field compresses: Sufferage reaches 227.3 seconds and APEX-BC 233.7 seconds, with deadline misses of 4.06 and 4.12 per cent respectively. Both case studies reproduce the pattern of Section VII-E, and we report them as confirmation of the negative result rather than as evidence for the proposed method.

VIII. LIMITATIONS

Several limitations qualify these results. All findings are simulation results obtained with a calibrated workload generator; the real traces were inaccessible, and no hardware validation was performed. Decision times are wall-clock measurements of a Python implementation and should be read comparatively rather than absolutely, although the equal-evaluation-budget protocol makes the comparison meaningful. Tasks are assumed independent and non-preemptable, so workflows with precedence constraints, VM migration, multi-tenant interference and failures are out of scope. The linear-programming relaxation is a lower bound and is known to be loose for unrelated machines, so the true optimality gaps are smaller than those reported. Finally, the negative online result is specific to mean response time and deadline satisfaction under the arrival processes studied; a scheduler whose objective was aligned with response time might behave differently, and constructing one is the obvious next step.

IX. CONCLUSION AND FUTURE WORK

We identified a structural defect in the way metaheuristic cloud schedulers are guided. The makespan, being a maximum, is a degenerate search signal: 16.43 per cent of single-task migrations leave it unchanged and steepest descent stalls after fewer than three moves. Replacing it during the search by a normalised power mean with a geometrically annealed exponent repairs the landscape, is provably safe within a factor m^(1/p), and yields APEX-BC, which lowers the makespan by 5.81 per cent relative to the strongest heuristic and halves the gap to a linear-programming bound, at a decision cost some forty times smaller than conventional metaheuristics at an identical evaluation budget.

We also reported two results that constrain the field, and our own contribution with it. The uniform speed model used in most published work is degenerate, leaving no headroom for any search method and casting doubt on gains reported within it; and in an online rolling-horizon regime no metaheuristic, ours included, matches the Sufferage heuristic on response time or deadline satisfaction.

Future work follows directly from the limitations. The most promising direction is a surrogate aligned with mean flow time rather than with the completion profile, which would address the online deficiency head-on; a natural candidate is a power mean over per-task completion times under the shortest-processing-time ordering. Extending the model to precedence-constrained workflows, validating on real traces once accessible, and incorporating migration cost and interference are also warranted.

ACKNOWLEDGMENT

The author thanks the Research and Development Division of Tekstein Solutions Private Limited for supporting this work.

REFERENCES

[1] P. Mell and T. Grance, “The NIST definition of cloud computing,” National Institute of Standards and Technology, Special Publication 800-145, 2011.

[2] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,” J. Parallel Distrib. Comput., vol. 61, no. 6, pp. 810–837, 2001.

[3] O. H. Ibarra and C. E. Kim, “Heuristic algorithms for scheduling independent tasks on nonidentical processors,” J. ACM, vol. 24, no. 2, pp. 280–289, 1977.

[4] J. K. Lenstra, D. B. Shmoys, and É. Tardos, “Approximation algorithms for scheduling unrelated parallel machines,” Math. Program., vol. 46, no. 1–3, pp. 259–271, 1990.

[5] R. L. Graham, “Bounds on multiprocessing timing anomalies,” SIAM J. Appl. Math., vol. 17, no. 2, pp. 416–429, 1969.

[6] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, “Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” J. Parallel Distrib. Comput., vol. 59, no. 2, pp. 107–131, 1999.

[7] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm,” J. Global Optim., vol. 39, no. 3, pp. 459–471, 2007.

[8] 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,” Softw. Pract. Exper., vol. 41, no. 1, pp. 23–50, 2011.

[9] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” Future Gener. Comput. Syst., vol. 28, no. 5, pp. 755–768, 2012.

[10] L. D. Dhinesh Babu and P. Venkata Krishna, “Honey bee behavior inspired load balancing of tasks in cloud computing environments,” Appl. Soft Comput., vol. 13, no. 5, pp. 2292–2303, 2013.

[11] S. G. Domanal and G. R. M. Reddy, “Optimal load balancing in cloud computing by efficient utilization of virtual machines,” in Proc. 6th Int. Conf. Communication Systems and Networks (COMSNETS), 2014, pp. 1–4.

[12] H. Chen, F. Wang, N. Helian, and G. Akanmu, “User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing,” in Proc. Nat. Conf. Parallel Computing Technologies (PARCOMPTECH), 2013, pp. 1–8.

[13] M. Kumar and S. C. Sharma, “Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment,” Comput. Electr. Eng., vol. 69, pp. 395–411, 2018.

[14] M. Adhikari and T. Amgoth, “Heuristic-based load-balancing algorithm for IaaS cloud,” Future Gener. Comput. Syst., vol. 81, pp. 156–165, 2018.

[15] W. Kimpan and B. Kruekaew, “Heuristic task scheduling with artificial bee colony algorithm for virtual machines,” in Proc. Joint 8th Int. Conf. Soft Computing and Intelligent Systems and 17th Int. Symp. Advanced Intelligent Systems, 2016, pp. 281–286.

[16] S. H. H. Madni, M. S. A. Latiff, M. Abdullahi, S. M. Abdulhamid, and M. J. Usman, “Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment,” PLoS ONE, vol. 12, no. 5, e0176321, 2017.

[17] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale: Google trace analysis,” in Proc. 3rd ACM Symp. Cloud Computing (SoCC), 2012, pp. 1–13.

[18] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in Proc. 10th European Conf. Computer Systems (EuroSys), 2015, pp. 1–17.

[19] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in Proc. IEEE Int. Conf. Neural Networks, 1995, pp. 1942–1948.

[20] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” IEEE Trans. Syst., Man, Cybern. B, vol. 26, no. 1, pp. 29–41, 1996.

[21] S. García, D. Molina, M. Lozano, and F. Herrera, “A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: a case study on the CEC’2005 special session on real parameter optimization,” J. Heuristics, vol. 15, no. 6, pp. 617–644, 2009.

[22] A. Vargha and H. D. Delaney, “A critique and improvement of the CL common language effect size statistics of McGraw and Wong,” J. Educ. Behav. Stat., vol. 25, no. 2, pp. 101–132, 2000.

[23] S. Holm, “A simple sequentially rejective multiple test procedure,” Scand. J. Stat., vol. 6, no. 2, pp. 65–70, 1979.

[24] G. H. Hardy, J. E. Littlewood, and G. Pólya, Inequalities, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1952.

[25] A. M. Hussain and M. Aleem, “GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures,” Data, vol. 3, no. 4, art. 38, 2018.

[26] P. Moscato, “On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms,” Caltech Concurrent Computation Program, Rep. 826, 1989.