

Task Scheduling in Cloud Computing: A Critical Review of Heuristic, Metaheuristic and Learning-Assisted Approaches

Nirmal Godara and Sanjeev Kumar

Department of Computer Science & Engineering
Guru Jambheshwar University of Science & Technology, Hisar, Haryana

Abstract—Mapping user tasks onto virtual machines is one of the central problems of cloud resource management, and it has become harder as clouds have grown more heterogeneous. Because the problem is NP-hard, recent work relies on heuristics and nature-inspired metaheuristics rather than exact methods. This paper reviews 24 studies published between 2016 and 2019: five surveys, one benchmark of rule-based heuristics, and eighteen papers that each propose a new scheduler. We group the proposals into single-population metaheuristics, hybrid metaheuristics, multi-objective and preference-aware schedulers, and learning-assisted schedulers, with rule-based heuristics as the reference point. For each group we summarize what was optimized, how it was evaluated and what the reported gains amount to. The comparison shows a productive but uneven field. Nearly all results come from CloudSim or MATLAB simulations, often with fewer than 100 tasks or ten virtual machines, and the baselines are frequently FCFS, Min–Min or an untuned genetic algorithm. Only five studies use real workload traces. We close with open problems, including adaptive trade-off control, dependent-task workloads, scheduler overhead and the need for shared benchmarks.

Index Terms—Cloud computing, task scheduling, metaheuristics, particle swarm optimization, ant colony optimization, multi-objective optimization, load balancing.

I. INTRODUCTION

A cloud provider sells capacity by the hour, and increasingly by the second, and the user pays for whatever the scheduler decides to use. That is why task scheduling is more than an internal efficiency concern. A poor mapping of tasks to virtual machines (VMs) lengthens the makespan, leaves some VMs idle while others build long queues, and raises both the user's bill and the provider's energy costs. Moon et al. [1] make the point with a simple example: if two schedulers finish the same workload in 151s and 130s, a user on pay-per-use pricing pays for the extra 21 s of the slower one.

Even the basic version of the problem, assigning n independent tasks to m heterogeneous VMs so that the makespan is minimized, is NP-hard, and practical settings add deadlines, budgets, data transfer and energy limits. Exact methods do not scale to realistic instance sizes, so the literature has settled on two broad strategies. The first relies on fast rule-based heuristics such as Min–Min, Max–Min and Sufferage. The second uses population-based metaheuristics, among them the genetic algorithm (GA), particle swarm

optimization (PSO) and ant colony optimization (ACO), which spend more computation in exchange for better schedules. Several surveys have catalogued these methods [2]–[6]. Singh and Chana [2], for example, screened 1206 papers and analyzed 110 of them in detail, while Singh et al. [3] concentrate on bio-inspired and swarm-based schedulers for workflows.

This review has a narrower aim. Rather than list every algorithm, we examine a set of 24 papers published between 2016 and 2019 and ask three questions of each proposal. What exactly was optimized? How was it evaluated? And how much of the reported improvement would survive a stronger comparison? The set contains five surveys, one benchmark study of classical heuristics [7], and eighteen papers that propose new scheduling methods. The proposals range from standard PSO [8] to a fractional-order grey wolf optimizer [9], an entropy-guided ant colony for hybrid clouds [10] and a classifier that chooses which optimizer to run [11].

The rest of the paper is organized as follows. Section II sets out a common formulation of the problem and its metrics, so that papers written in different vocabularies can be compared. Section III reviews the approaches in five groups. Section IV compares the studies side by side and discusses recurring weaknesses in how they were evaluated. Section V lists open problems that follow from those weaknesses, and Section VI concludes.

II. PROBLEM FORMULATION AND PERFORMANCE METRICS

A. Task and Resource Model

Most of the reviewed papers share the same core model, even when their notation differs. A set of tasks $T = \{t_1, t_2, \dots, t_n\}$ is submitted to a data center hosting a set of VMs $V = \{v_1, v_2, \dots, v_m\}$. Each task has a length in million instructions (MI) and each VM a processing capacity in million instructions per second (MIPS), so the expected time to compute task i on VM j , written ETC_{ij} , is simply the ratio of the two. Li and Wang [12] express a complete schedule as a binary assignment matrix multiplied by the ETC matrix, which is a compact way of writing most of the objectives below. Tasks are normally taken to be independent and non-preemptive, and each VM runs one task at a time [8], [13].

The independence assumption matters more than it first appears. Workflow applications, whose tasks have precedence

constraints and exchange data, make up much of scientific computing on clouds, and Singh et al. [3] observe that even the workflow literature rarely considers storage cost or failure rate. Among the eighteen proposals reviewed here, only a few include communication at all. FGMTS [9] divides each task into subtasks linked by bandwidth, the ANP-based scheduler [12] adds a connection cost matrix between physical machines, and the mobile cloud framework [14] counts wireless transmission energy. Others state openly that bandwidth and data transfer were left out of the model [15], [16].

B. Objectives and Metrics

Nearly every paper reports the makespan, the time at which the last VM finishes its work:

$$MS = \max_{1 \leq j \leq m} CT_j \quad (1)$$

where CT_j is the completion time of the last task assigned to VM j . Makespan reflects the user's point of view. Flowtime, the sum of the completion times C_i of all tasks,

$$FT = \sum_{i=1}^n C_i \quad (2)$$

reflects average responsiveness, and Cui et al. [17] combine it with makespan in a weighted objective. Load balance is measured in several ways. The most common is the degree of imbalance,

$$DI = (T_{\max} - T_{\min}) / T_{\text{avg}} \quad (3)$$

where $T_{\max}$, $T_{\min}$ and T_{avg} are the maximum, minimum and average execution times across the VMs [7], [18], [19]. Zhong et al. [15] use a close relative, the ratio of the lightest to the heaviest VM load.

Beyond these time-based measures, the papers optimize execution cost (usually the VM price multiplied by busy time), resource utilization, energy consumption, the number of deadline violations [10] and the task guarantee ratio [16]. Which metrics an author chooses tends to follow whose interests the scheduler serves. Thaman and Singh [4] capture this with a distinction between goal-oriented schemes, which serve the provider by maximizing utilization and revenue, and constraint-oriented schemes, which let the user fix QoS limits through a service level agreement (SLA). Singh and Chana [2] counted how often each QoS parameter appeared in papers from 2009 to mid-2015. Cost led with 19%, followed by energy (18%), execution time (17%), response time (15%) and utilization (14%), while scalability and migration cost appeared in only 3% and 2% of papers. The papers reviewed here follow much the same pattern.

III. A REVIEW OF SCHEDULING APPROACHES

Fig. 1 groups the reviewed methods by the way they search for a schedule. The boundaries between groups are not sharp. A hybrid metaheuristic can also be multi-objective, and both learning-assisted schedulers contain a conventional algorithm inside them. We place each paper according to what its authors present as the main contribution.

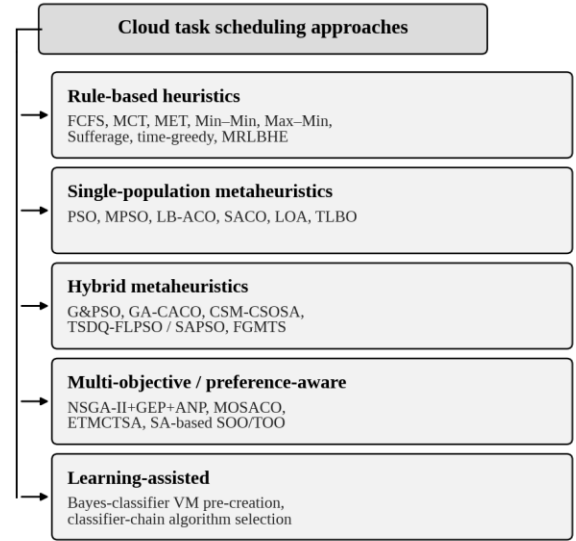


Fig. 1. Classification of the scheduling approaches covered in this review, with the specific methods discussed in Section III.

A. Rule-Based Heuristics

Rule-based heuristics make one pass over the task list and assign each task by a fixed rule. They are fast and deterministic, which is why they remain the default baseline. Madni et al. [7] provide the most useful reference point in this set. They implemented six heuristics (FCFS, MCT, MET, Min-Min, Max-Min and Sufferage) in CloudSim and ran them on homogeneous and heterogeneous VMs, using four synthetic task distributions (uniform, normal, left-skewed and right-skewed) and two real logs from the Parallel Workloads Archive, HPC2N and NASA. No single rule won everywhere. Min-Min gave the best overall results, Max-Min and Sufferage came close, and MET, which ignores current VM load entirely, achieved the best degree of imbalance. The authors recommend that these heuristics serve as standard baselines for new proposals, advice that most of the papers below follow only in part.

Their weaknesses are equally well documented. Keivani et al. [5] note that Min-Min is simple and quick but starves long tasks and loads resources unevenly. Zhang and Zhou [16] saw this directly: under Min-Min the high-capacity VMs were heavily used while the small ones sat nearly idle, and under Max-Min the pattern reversed. Heuristics are also usually built around a single goal. Mao et al. [20] compare a time-greedy rule, which considers nothing but task running time, with a main-resource load balancing rule (MRLBHE) that looks at each task's dominant resource. The first was fastest and the second used the least energy, but neither could be told to prefer one goal over the other. That limitation motivated the tunable scheduler discussed in Section III-D.

B. Single-Population Metaheuristics

PSO and ACO are the two workhorses of this group. Jamali et al. [8] apply standard PSO to minimize makespan and

waiting time in a CloudSim data center with 50 hosts and 50 VMs and report that it beats both GA and FCFS, especially on larger task sets. The claim is plausible, but the results are shown only as charts, without the number of runs or any measure of spread, so the size of the gain is hard to judge. Nzanywayingoma and Yang [21] modify PSO by setting each particle's inertia weight according to its fitness. Particles with above-average fitness receive the larger weight and keep exploring, while the rest receive the smaller weight and search locally. The resulting MPSO converged faster than PSO and GA in MATLAB tests and gave lower execution time and higher utilization in CloudSim, although the CloudSim scenarios used only 5 and 10 VMs.

For ACO, Gupta and Garg [13] build a load-balancing variant, LB-ACO, that uses non-dominated sorting to return a Pareto front between makespan and load balance. It was tested with 20 to 100 tasks on 8 to 32 VMs and compared only with NSGA-II. Moon et al. [1] target a different weakness of ACO, namely that leading ants can reinforce long paths whose pheromone was accumulated by mistake. In their slave ants scheme (SACO), each group of ants uses different probability parameters, and after every tour the ant with the best makespan takes the leading role. With 50 VMs and 100 to 700 tasks, the makespan gap to plain ACO grew from about 15 s at 200 tasks to more than 50 s at 700. The authors also report that SACO lost to both ACO and IACO at 100 tasks once its preprocessing time was counted. That kind of candor is rare in this set, where few papers show a case in which their own method loses.

A third strand borrows newer metaphors. Almezeini and Hafez [18] adapt the Lion Optimization Algorithm, in which the population is split into resident prides and nomads with different movement rules, and report lower makespan and degree of imbalance than GA and PSO in CloudSim. Shrivastava and Kumar [22] apply Teaching-Learning-Based Optimization (TLBO) to service request scheduling. TLBO needs no algorithm-specific parameters beyond population size and iteration count, which is a practical advantage, but the evaluation amounts to one MATLAB plot of fitness against iterations beside GA. Singh et al. [3] found that ACO, PSO and GA still account for most of the literature and that newer algorithms have been tried on few objectives. From the evidence here, it is not yet clear what the lion or classroom analogy adds beyond a different balance between exploration and exploitation.

C. Hybrid Metaheuristics

Hybridization is the most common strategy in the set, and the motivation rarely changes. One algorithm explores well but converges slowly, another converges quickly but gets trapped, and so the two are combined. The simplest version improves the starting population. Zhong et al. [15] seed PSO with a greedy assignment (G&PSO), which gave faster early convergence and more even VM loads than random initialization in tests with 5 and 10 VMs and 50 and 500 tasks.

Cui et al. [17] use GA to provide the starting point for a chaotic ACO (GA-CACO). They model task arrivals as a Poisson process feeding an M/G/S queue and minimize a weighted sum of makespan and flowtime. With 80 tasks the objective value was 24.04 for GA-CACO, 25.56 for ACO and 26.77 for GA, and across 20 to 80 tasks the margin stayed at about 6% to 7% over ACO and 9% to 12% over GA.

Gabi et al. present the Cloud Scalable Multi-Objective Cat Swarm Optimization based Simulated Annealing algorithm (CSM-CSOSA) in two conference papers [23], [24]. When the global best solution stops improving, the tracing mode of cat swarm optimization hands over to a simulated annealing search whose settings are chosen with a Taguchi orthogonal design. Against MOGA, MOSACO and MOPSO, the first paper reports execution-time improvements of 34.59%, 30.37% and 17.87% on normally distributed synthetic tasks, and 47.86%, 42.68% and 11.31% on the HPC2N log [23]. The second paper takes 5000 to 14,000 tasks from the SDSC-SP2 log and adds a scalability analysis based on isospeed efficiency [24]. As processing elements increased from 5 to 50, CSM-CSOSA stayed inside the authors' acceptable range of $0 < \psi < 1$, whereas each baseline left that range at least once beyond 30 elements. Readers should be aware that the two papers describe essentially the same algorithm on different workloads.

Ben Alla et al. [19] redesign the queueing structure as well as the optimizer. Incoming tasks are first ordered and distributed across dynamic dispatch queues (TSDQ), and are then mapped to VMs by PSO whose inertia weight is set either by a fuzzy logic controller (TSDQ-FLPSO) or by simulated annealing (TSDQ-SAPSO). The evaluation is broader than usual. It uses synthetic data together with logs from the Parallel Workloads Archive, and it compares against PSO-LDIW, PSO-RIW, SGA, MGA and MACOLB. In the summary comparison with FCFS, TSDQ-FLPSO reduced the makespan from 18.75 to 14.09, lowered the degree of imbalance from 1.09 to 0.92, raised utilization from 75% to 82.1% and cut cost from \$375 to \$281.8.

FGMTS [9] brings fractional calculus into the grey wolf position update, so that each move carries some memory of earlier positions, and scores schedules with a six-term fitness covering execution time, communication time, execution cost, communication cost, energy and utilization. Read closely, the results are mixed. In the second of two cloud setups FGMTS cut execution time to 0.1906 against 0.2816 for GA and GWO, a clear gain. In the first setup, however, PSO reached a lower execution cost (0.0768 against 0.0814), GA reached higher utilization, and the reported energy figure for FGMTS (0.3048) was roughly ten times that of PSO (0.0287). A single weighted score with six terms will always trade some objectives against others, and a Pareto analysis would have shown those trade-offs more openly than a combined fitness value does.

D. Multi-Objective and Preference-Aware Schedulers

Several proposals accept that no single schedule is best for every user and try to expose the trade-off instead. Li and Wang [12] optimize four objectives together (task completion time, total completion time, energy cost and load balance) using NSGA-II with search operators taken from gene expression programming (GEP). The resulting Pareto set is passed to an analytic network process (ANP) model that ranks solutions according to user preference, which can be set to favor time, to favor energy and load, or to weight everything equally. The appeal is that a change of preference does not require the optimizer to be run again, as a weighted-sum formulation would. The evaluation, however, used randomly generated data in MATLAB with a population of 30, and only 10 representative solutions were screened by the ANP stage.

Zuo et al. [10] address the hybrid cloud, in which cheap but slower private resources are supplemented by more expensive public ones. MOSACO combines a time-first strategy (minimize completion time under a cost constraint), a cost-first strategy (minimize cost under a deadline) and an entropy optimization model that balances user QoS against provider profit, all searched with an improved ant colony. In simulation it gave the shortest completion time, the lowest cost and the fewest deadline violations of the five methods tested, and it was also compared with SLPSO-SA and SPSO-SA on three IaaS application examples taken from earlier work. The main simulation baselines, Min-Min and FIFO, are weak, but the application examples make this one of the more convincing evaluations in the set.

Mao et al. [20] take a lighter route. Their ETMCTSA scheduler blends the time-greedy and energy-aware heuristics through a single probability parameter α , so the operator can lean toward speed or toward saving energy. With $\alpha = 0.6$ it used 10.5% more energy than MRLBHE but took 9.72% less time, and it took 5.8% more time than the time-greedy rule while using 4.5% less energy. With $\alpha = 0.3$ the balance shifted toward speed, giving 13.16% less time than MRLBHE and 2.06% less energy than the time-greedy rule. These are modest gains, but they are clearly stated, tested with 1000 to 5000 tasks, and accompanied by a measurement of scheduling overhead. The limitation the authors themselves point out is that α is fixed in advance instead of following the workload.

The mobile cloud framework of Liu et al. [14] has the richest model in the set. Tasks are assigned to individual cores on mobile devices or cloud servers, energy is modeled across devices, network and data center with DVFS and heat recirculation taken into account, and the interacting factors are represented as a tensor. Simulated annealing solves both a single-objective version (energy) and a triple-objective version (energy, reliability and quality of experience). The evaluation is much thinner than the model. The single-objective version is compared with one earlier scheme, and the triple-objective version only with itself, since the authors found no comparable method.

E. Learning-Assisted Schedulers

Two papers use learning not to replace the optimizer but to decide how it should be used. Zhang and Zhou [16] train a classifier, designed on the principle of a Bayes classifier, on historical scheduling data to predict which VM types will be needed, and they create those VMs in advance. A second stage then matches arriving tasks to the pre-created VMs. This addresses a real delay that most papers ignore, the time taken to start a VM. On a modified Google cluster trace of 10,000 tasks, the method gave lower makespan, waiting time and failure rate than Min-Min and Max-Min, and VM utilization stayed nearly flat across VM types. The only baselines, however, are those two heuristics.

Rjoub and Bentahar [11] run ACO, artificial bee colony (ABC) and PSO on each data center and train a multi-label classifier chain to predict which of the three will give the best execution time for a given number of VMs and task size. Over 2 to 50 VMs and task sizes from 30 to 2700 bytes, averaged across 112 runs, the combined model reduced makespan by between 7% and 75% relative to the individual algorithms. The width of that range tells its own story, since the benefit depends on how badly the wrong algorithm would otherwise have performed. Choosing among algorithms is a sensible job for a learner, and it deserves testing on workloads larger and less regular than 50 tasks spread over three data centers.

This kind of work should be kept apart from a looser use of the term machine learning. Gondhi and Gupta [6] survey GA, simulated annealing, tabu search, ACO, PSO, cuckoo search, the bat algorithm and several others under the heading of machine learning-based scheduling. These are search heuristics rather than learning methods, and the broad label hides how little genuine learning has yet entered this literature.

IV. COMPARATIVE ANALYSIS

Table I summarizes the eighteen proposals by method, objectives, evaluation setting and baselines. Several patterns stand out when they are placed side by side.

Simulation dominates. Every proposal was evaluated in simulation, most often in CloudSim or an extension of it, three in MATLAB and one in a custom Java simulator, and two do not name their simulator at all. None was run on a real cloud, although Zhang and Zhou [16] list this as future work. Simulation is a sensible first step, but CloudSim results depend on configuration choices, such as VM speeds and task length distributions, that are reported unevenly, so figures from different papers cannot be compared directly.

Real workloads are the exception. Only five studies use real traces: HPC2N and NASA [7], HPC2N [23], SDSC-SP2 [24], the Parallel Workloads Archive [19] and the Google cluster trace [16]. The rest rely on uniformly or normally distributed synthetic tasks, which are easier to schedule than the heavy-tailed mixes seen in production.

Instances are small. Several evaluations stop at 5 or 10 VMs [15], [21], 100 tasks [13] or 80 tasks [17]. Metaheuristics usually justify their extra computation on large problems, so

small tests say little about the case that matters most. The CSM-CSOSA scalability study [24], the 700-task runs of

SACO [1] and the 5000-task runs of ETMCTSA [20] are rare cases that vary scale deliberately.

TABLE I
SUMMARY OF THE REVIEWED SCHEDULING PROPOSALS

Ref.	Method	Main objectives	Evaluation setting	Baselines
<i>Single-population metaheuristics</i>				
[8]	Standard PSO	Makespan, waiting time	CloudSim; 50 hosts, 50 VMs	GA, FCFS
[21]	PSO with fitness-dependent inertia (MPSO)	Execution time, cost, utilization	MATLAB and CloudSim; 5–10 VMs	PSO, GA
[13]	ACO with Pareto sorting (LB-ACO)	Makespan, load balance	CloudSim; 20–100 tasks, 8–32 VMs	NSGA-II
[1]	ACO with slave ants (SACO)	Makespan	Simulation; 100–700 tasks, 50 VMs	ACO, IACO
[18]	Lion optimization algorithm	Makespan, DI, utilization, cost	CloudSim 3.0.3; one data center	GA, PSO
[22]	TLBO	Request–VM mapping fitness	MATLAB	GA
<i>Hybrid metaheuristics</i>				
[15]	Greedy-seeded PSO (G&PSO)	Completion time, load balance	CloudSim; 5–10 VMs, 50–500 tasks	PSO
[17]	GA + chaotic ACO (GA-CACO)	Weighted makespan and flowtime	Simulation; 10–80 tasks, M/G/S queue	GA, ACO
[23]	CSO + Taguchi-tuned SA (CSM-CSOSA)	Execution time, cost	CloudSim; synthetic and HPC2N	MOGA, MOSACO, MOPSO
[24]	CSM-CSOSA, scalability study	Time, cost, isospeed scalability	CloudSim; SDSC-SP2, 5000–14,000 tasks	MOGA, MOSACO, MOPSO
[19]	Dynamic queues + fuzzy- or SA-tuned PSO	Waiting time, makespan, cost, DI, utilization	CloudSim; synthetic and PWA logs	PSO variants, SGA, MGA, MACOLB, FCFS
[9]	Fractional grey wolf optimizer (FGMTS)	Six-term weighted fitness	Java/CloudSim; two cloud setups	PSO, GA, GWO
<i>Multi-objective and preference-aware schedulers</i>				
[12]	NSGA-II + GEP with ANP selection	Time, total time, energy, load	MATLAB; random data	GA, weighted-sum MOO
[10]	ACO + entropy model (MOSACO), hybrid cloud	Time, cost, deadlines, private-cloud use	CloudSim; three IaaS examples	Min–Min, FIFO, SLPSO-SA, SPSO-SA
[20]	Tunable greedy blend (ETMCTSA)	Energy, execution time	MultiRECloudSim; 1000–5000 tasks	Simple, Random, MRLBHE, time-greedy
[14]	SA over tensor model with DVFS	Energy, reliability, QoE	Custom JVM simulator	One prior scheme (energy only)
<i>Learning-assisted schedulers</i>				
[16]	Bayes-style classifier + VM matching	Makespan, waiting, failure, guarantee ratio	CloudSim; Google cluster trace	Min–Min, Max–Min
[11]	Classifier chain over ACO/ABC/PSO	Makespan	CloudSim 2.1; 2–50 VMs, 50 tasks	ACO, ABC, PSO

DI: degree of imbalance; PWA: Parallel Workloads Archive; QoE: quality of experience; MOO: multi-objective optimization.

Baselines are weak or few. FCFS, Min–Min, Max–Min and standard GA or PSO account for most comparisons. LB-ACO [13] and TLBO [22] are each compared with a single method, and the triple-objective mobile scheduler [14] with none. A new metaheuristic is rarely compared with another recent metaheuristic tuned with equal care. Without that comparison, an improvement of 30% or more over an untuned GA is difficult to interpret.

Statistics and overhead are underreported. Only a handful of papers state how many runs were averaged (10 for MOSACO [10], 30 for CSM-CSOSA [24], 112 for MLCCSI [11]), and measures of spread such as standard deviations or confidence intervals are the exception rather than the rule. Scheduler overhead, the time the scheduler itself needs to produce a mapping, is measured by Mao et al. [20] and acknowledged by Moon et al. [1] but otherwise ignored, even though a population-based search run for 1000 iterations, the setting used for MPSO [21], is not free.

Taken together, these observations do not mean that the proposed methods are ineffective. Most of them very likely do

improve on FCFS and Min–Min, and some, TSDQ-FLPSO and MOSACO in particular, were tested with reasonable care. The difficulty is that the published evidence does not let a reader rank the methods against each other, which is the question a practitioner would actually ask.

V. OPEN ISSUES AND RESEARCH DIRECTIONS

A. Adaptive Trade-Off Control

Mao et al. [20] fix α before each run and note that it ought to follow the workload instead. The same applies to the fitness weights in GA-CACO [17] and FGMTS [9] and to the preference settings of the ANP stage [12]. A scheduler that leans toward energy saving during quiet periods and toward response time at peak load would be more useful than any fixed compromise, and it would also remove a parameter that users are poorly placed to choose.

B. Dependent Tasks and Data Movement

The independent-task model underlies nearly every proposal reviewed here. Singh et al. [3] and Thaman and Singh [4] both identify workflows, data dependency and batch jobs as under-explored, and the present set supports that view. Once data transfer enters the model, a task's cost no longer depends only on its length and the VM's speed, and many of the simple encodings used by PSO and ACO schedulers would need to change.

C. Learning Beyond Algorithm Selection

The two learning-assisted papers [11], [16] use learning to predict demand or to pick an algorithm. Rjoub and Bentahar [11] suggest k-nearest neighbors, decision trees and neural networks as next steps. Learning could also warm-start a metaheuristic from earlier schedules, which would ease both the overhead problem and the sensitivity to initialization that G&PSO [15] and GA-CACO [17] address by hand.

D. Energy Across the Whole Path

Energy appears as one term in several fitness functions [9], [12], but only the mobile cloud framework [14] models it across devices, network and data center with DVFS and heat recirculation. Joint models of this kind are promising, particularly as more work moves to the edge, but they need to be tested against more than one baseline before their benefits can be judged.

E. Reliability, Migration and Security

Failures are almost absent from the reviewed evaluations. Zhang and Zhou [16] report a task failure rate and Liu et al. [14] treat system reliability as an objective, but nearly every other proposal assumes that VMs never fail and that tasks never need to move. Singh et al. [3] note that newer metaheuristics such as the lion, bat and wind-driven algorithms have not yet been applied to reliability, VM migration or security objectives, and Thaman and Singh [4] count privacy, security and VM migration among the open issues of cloud scheduling in general. A scheduler that performs well only when nothing goes wrong has not yet been tested under the conditions it will meet in production.

F. Shared Benchmarks and Reporting

The most practical step would be agreement on a small set of benchmark workloads, VM configurations and baselines. Madni et al. [7] already propose the classical heuristics as a baseline set, and the Parallel Workloads Archive and Google traces are freely available. Reporting the number of runs, the spread of results and the scheduler's own running time would cost little and would make future papers far easier to compare. Scalability and migration cost, which Singh and Chana [2] found in only 3% and 2% of papers, should be measured as a matter of routine.

VI. CONCLUSION

This review examined 24 papers on cloud task scheduling, covering rule-based heuristics, single and hybrid metaheuristics, multi-objective schedulers and learning-assisted approaches. The field has produced many algorithms, and several contain ideas worth keeping: creating VMs in advance from historical demand [16], dispatching tasks through dynamic queues [19], exposing the energy and time trade-off through a single parameter [20], and letting a classifier decide which optimizer to run [11]. What the field lacks is comparable evidence. Most evaluations are small, synthetic and set against weak baselines, so the reported improvements, some above 40%, cannot be ranked with any confidence. In our view, progress over the next few years will depend less on new nature-inspired metaphors than on shared workloads, stronger baselines and schedulers that adapt to the workload they are actually given.

REFERENCES

- [1] Y. J. Moon, H. C. Yu, J. M. Gil, and J. B. Lim, "A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments," *Human-centric Computing and Information Sciences*, vol. 7, no. 1, Dec. 2017, doi: 10.1186/s13673-017-0109-2.
- [2] S. Singh and I. Chana, "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges," Jun. 01, 2016, Springer Netherlands. doi: 10.1007/s10723-015-9359-2.
- [3] P. Singh, M. Dutta, and N. Aggarwal, "A review of task scheduling based on meta-heuristics approach in cloud computing," *Knowl Inf Syst*, vol. 52, no. 1, pp. 1–51, Jul. 2017, doi: 10.1007/s10115-017-1044-2.
- [4] J. Thaman and M. Singh, "Current Perspective in Task Scheduling Techniques in Cloud Computing: A Review," *International Journal in Foundations of Computer Science & Technology*, vol. 6, no. 1, pp. 65–85, Jan. 2016, doi: 10.5121/ijfcs.2016.6106.
- [5] M. El. Mohajir, "A review of recent methods of task scheduling in cloud computing," in *Proc. 19th IEEE Mediterranean Electrotechnical Conference 2018: proceedings, Marrakech, Morocco, May 2-4*, pp. 104–109. IEEE, 2018.
- [6] N. K. Gondhi and A. Gupta, "Survey on Machine Learning based scheduling in Cloud Computing," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Mar. 2017, pp. 57–61. doi: 10.1145/3059336.3059352.
- [7] S. H. H. Madni, M. S. Abd 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, May 2017, doi: 10.1371/journal.pone.0176321.
- [8] S. Jamali, F. Alizadeh, and S. Sadeqi, "Task scheduling in cloud computing using particle swarm optimization," in *Proc. 3rd Int. CUA Graduate Students Symp.*, Ardabil, Iran, Jun. 2016, pp. 192–198.
- [9] K. Sreenu and S. Malempati, "FGMTS: Fractional grey Wolf optimizer for multi-objective task scheduling strategy in cloud computing," *Journal of Intelligent and Fuzzy Systems*, vol. 35, no. 1, pp. 831–844, 2018, doi: 10.3233/JIFS-17148.
- [10] L. Zuo, L. Shu, S. Dong, Y. Chen, and L. Yan, "A multi-objective hybrid cloud resource scheduling method based on deadline and cost constraints," *IEEE Access*, vol. 5, pp. 22067–22080, 2017, doi: 10.1109/ACCESS.2016.2633288.
- [11] G. Rjoub and J. Bentahar, "Cloud task scheduling based on swarm intelligence and machine learning," in *Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017*, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 272–279. doi: 10.1109/FiCloud.2017.52.

- [12] K. Li and J. Wang, "Multi-objective optimization for cloud task scheduling based on the ANP model," *Chinese Journal of Electronics*, vol. 26, no. 5, pp. 889–898, 2017, doi: 10.1049/cje.2017.07.019.
- [13] A. Gupta and R. Garg, "Load Balancing Based Task Scheduling with ACO in Cloud Computing," in *Proc. Int. Conf. Comput. Appl. (ICCA)*, Doha, Qatar, 2017, pp. 174–179.
- [14] H. Liu et al., "A Holistic Optimization Framework for Mobile Cloud Task Scheduling," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 2, pp. 217–230, Apr. 2019, doi: 10.1109/TSUSC.2017.2765520.
- [15] Z. Zhong, K. Chen, X. Zhai, and S. Zhou, "Virtual Machine-Based Task Scheduling Algorithm in a Cloud Computing Environment," 2016.
- [16] P. Y. Zhang and M. C. Zhou, "Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 772–783, Apr. 2018, doi: 10.1109/TASE.2017.2693688.
- [17] H. Cui, X. Liu, T. Yu, H. Zhang, Y. Fang, and Z. Xia, "Cloud service scheduling algorithm research and optimization," *Security and Communication Networks*, vol. 2017, 2017, doi: 10.1155/2017/2503153.
- [18] N. Almezeini and P. Alaaeldin Hafez, "Task Scheduling in Cloud Computing using Lion Optimization Algorithm," 2017. [Online]. Available: www.ijacsa.thesai.org.
- [19] H. ben Alla, S. ben Alla, A. Touhafi, and A. Ezzati, "A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment," *Cluster Comput*, vol. 21, no. 4, pp. 1797–1820, Dec. 2018, doi: 10.1007/s10586-018-2811-x.
- [20] L. Mao, Y. Li, G. Peng, X. Xu, and W. Lin, "A multi-resource task scheduling algorithm for energy-performance trade-offs in green clouds," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 233–241, Sep. 2018, doi: 10.1016/j.suscom.2018.05.003.
- [21] F. Nzanywayingoma and Y. Yang, "Effective task scheduling and dynamic resource optimization based on heuristic algorithms in cloud computing environment," *KSII Transactions on Internet and Information Systems*, vol. 11, no. 12, pp. 5780–5802, Dec. 2017, doi: 10.3837/tiis.2017.12.006.
- [22] K. Shrivastava and R. Kumar, "SERVICE REQUEST SCHEDULING IN CLOUD COMPUTING USING META-HEURISTIC TECHNIQUE: TEACHING LEARNING BASED OPTIMIZATION (TLBO)," *International Research Journal of Engineering and Technology*, 2017, [Online]. Available: www.irjet.net
- [23] D. Gabi, A. S. Ismail, A. Zainal, Z. Zakaria, and A. Al-Khasawneh, "Cloud scalable multi-objective task scheduling algorithm for cloud computing using cat swarm optimization and simulated annealing," in *Proc. 8th Int. Conf. Inf. Technol. (ICIT)*, Amman, Jordan, 2017, pp. 599–604.
- [24] D. Gabi, A. Zainal, A. S. Ismail, and Z. Zakaria, "Scalability-aware scheduling optimization algorithm for multi-objective cloud task scheduling problem," in *Proc. 6th ICT Int. Student Project Conf. (ICT-ISPC)*, Skudai, Malaysia, 2017, pp. 1–6.