Unified Load Balancing Strategies for Enhanced Cloud Computing Solutions

Tearlach Magri and Rebecca Camilleri

Department of Computer Information Systems, University of Malta, Msida, Malta

Abstract. Cloud computing provides scalable, on-demand resources that support a wide range of services and applications. Efficient load balancing in cloud environments is critical when maintaining performance and quality of service. A hybrid Ant Colony Optimisation – Genetic Algorithm (ACO-GA) method is proposed for task scheduling in a hybrid cloud, implemented and evaluated using the CloudAnalyst simulator. The custom algorithm leverages ACO's rapid local search for assigning workloads to virtual machines and GA's global evolutionary search to diversify solutions. The ACO-GA is compared against Round Robin, pure ACO and pure GA strategies. Performance is measured by overall response time and data centre processing time. Simulation results indicate that the proposed ACO-GA outperforms the baseline strategies in both response time and data centre processing time, demonstrating that combining ACO's pheromone-guided optimisation and GA's genetic exploration leads to more balanced loads.

Keywords: Cloud Computing, Load Balancing, Round Robin, Ant Colony Optimisation, Genetic Algorithm

1 Introduction

Cloud computing is an emerging field that is rapidly advancing in industry as well as in research, moving computing and data away from desktop and portable PCs into large data centres [1]. In this environment, users are unaware of the physical locations of the underlying infrastructure. They simply consume services via the cloud paradigm and pay only for the resources they use [2]. Virtualisation underpins cloud computing by allowing a single physical server to host multiple operating systems and applications simultaneously, delivering services through isolated virtual instances [2]. The National Institute of Standards and Technology (NIST) describes cloud computing as a framework that provides users with seamless, on-demand network access to a shared pool of configurable computing resources — such as networks, servers, storage, applications and services — that can be quickly allocated and released with minimal management overhead or direct interaction with the service provider [3].

In cloud computing, the primary challenge is efficiently and accurately processing the massive, rapidly arriving stream of user requests. This requirement makes load balancing essential: by evenly distributing work, it prevents any single server or virtual machine (VM) from becoming a bottleneck, thereby maximising throughput and user satisfaction. In large-scale hybrid clouds — where private and public data centres coexist — static task assignment cannot cope with unpredictable workload shifts. Random request surges can leave some nodes overloaded while others remain idle, harming overall performance [4]. Consequently, adaptive, real-time scheduling mechanisms are needed to reallocate tasks dynamically and maintain balanced utilisation.

Traditional methods like Round Robin (RR) or simple throttling are easy to implement but often fail under heterogeneous workloads, leading to increased response times or bottlenecks. Nature-inspired metaheuristics, notably Ant Colony Optimisation (ACO) and Genetic Algorithm (GA), have been widely studied for task scheduling in different

Bibhu Dash et al: EDU, IOTCB, MLDA - 2025
pp. 29-39, 2025. IJCI – 2025
DOI:10.5121/ijci.2025.140403

areas. ACO simulates ant foraging behaviour with pheromone trails to find good allocation paths, while GA uses evolutionary operators, such as selection, crossover and mutation, to evolve scheduling solutions [4,5]. Both techniques provide complementary advantages that can be leveraged together. ACO quickly exploits good local paths [4], whereas GA provides global search to escape local optima. Integrating ACO and GA can potentially combine these advantages. Indeed, recent work has shown that hybrid metaheuristics approaches can outperform single-method strategies [5, 6]. The specific integration of ACO and GA used in this work is novel: the proposed algorithm introduces a distinct two-phase synergy (ACO followed by GA) including a candidate-selection step and adaptive mutation schedule.

A comprehensive experimental evaluation benchmarks the proposed approach against RR, ACO-only and GA-only schedulers using overall response time and data centre processing time metrics. The analysis demonstrates that the hybrid method delivers significant improvements in both response and processing times, aligning with previous studies that highlight the superiority of hybrid metaheuristics over individual heuristics [5, 6]. Finally, the implications of these findings for future cloud load balancing strategies are discussed.

The remainder of this paper is organised as follows. Section 2 reviews related work on cloud load balancing, ACO, GA and hybrid algorithms. Section 3 describes the hybrid ACO-GA methodology in detail. Section 4 outlines the experimental setup in CloudAnalyst. Section 5 presents simulation results and discussion. Section 6 concludes the paper and suggests future work.

2 Related Work

Efficient load balancing algorithms must optimise response time and resource utilisation [7]. A study by Shafiq et al. [8] note that a good load balancer distributes workloads to ensure high user satisfaction by effectively utilising VM resources. Load balancing algorithms are often classified as static or dynamic. Static methods, such as RR, use pre-known information but cannot adapt to workload changes. Dynamic methods adapt to current loads but require more complexity. Common algorithms include RR, Throttled and Active Monitoring, which maintain VM allocation tables or track current loads [9].

RR is a baseline strategy that cyclically assigns requests to servers. It is stateless and easy to implement, but can overload slow servers if capacities differ. Throttled Load Balancer checks for an available VM and queues requests if none is free, improving on naive approaches.

To improve performance, researchers have applied metaheuristic algorithms. The ACO, introduced by Dorigo in the 1990s [10], simulates pheromone-guided paths taken by ants to find optimal routes. In cloud load balancing, ACO variants have been proposed to assign tasks to VMs based on pheromone intensity reflecting server capacity or response time [4]. For instance, Nishant et al. [4] present a modified ACO that continuously updates a single pheromone matrix for all ants, achieving more balanced node workloads than classical ACO. Their work demonstrates that ACO can effectively improve load distribution in a cloud setting. However, ACO alone may converge to suboptimal allocations if pheromone information becomes biased or outdated.

Genetic Algorithm is another popular metaheuristic for scheduling. It treats each potential scheduling as a chromosome in a population and evolves solutions via selection, crossover and mutation. Dasgupta et al. [11] propose a GA-based load balancing strategy, simulating it with CloudAnalyst. They prioritise tasks and initialise GA populations accordingly. Their results show GA outperforming simple schedulers like First-Come First-Served, RR and Stochastic Hill Climbing in terms of response time. GAs are well-suited to

NP-hard problems like load balancing, as they can explore diverse assignments and avoid local optima [6,11]. However, pure GA may require many generations to converge, and its performance is sensitive to operator settings.

Several load balancing strategies, including recent metaheuristic hybrids, have been proposed. For instance, Lilhore et al. [5] integrate ACO with Water Wave Optimisation in a multi-objective cloud scheduler, achieving notable efficiency gains. Dey and Sangaraju [12] introduced a bio-inspired hybrid strategy using ACO and Particle Swarm Optimisation, demonstrating improved response times and energy efficiency in cloud data centers. The proposed approach is unique due to its two-phase ACO-GA design. The process begins with a dedicated ACO phase to narrow down candidate VMs, followed by a GA phase applied to that subset. Additionally, an adaptive mutation mechanism is employed, where the mutation probability decreases with each generation, alongside a specific candidate selection step. These innovative elements of the integration form the core contribution.

3 Proposed ACO-GA Load Balancing Methodology

This section presents the hybrid ACO–GA load balancing algorithm's design. It begins with a concise overview of ACO and GA fundamentals, followed by a detailed explanation of their integration.

3.1 Ant Colony Optimisation Principles

ACO is a metaheuristic that finds high-quality paths through graphs by mimicking ant foraging. In ACO, a set of artificial ants iteratively construct solutions; at each step, an ant chooses the next node to assign based on a probability that depends on pheromone levels and heuristic desirability. When an ant completes a tour, that is a full assignment of tasks, the quality of the solution (low response time) is evaluated. Pheromones are then updated: paths that led to better solutions receive more pheromone reinforcement, while pheromone evaporates on others [4]. Over time, good allocations accumulate pheromone and attract more ants, guiding the search toward effective load distributions.

Formally, if ant k at state (task assignment) chooses VM j, the transition probability is often proportional to $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$, where τ_{ij} is pheromone, η_{ij} is a heuristic (e.g. inverse of current load) and α, β tune their importance. This indirect communication allows ants to exploit promising assignments [4,5]. However, pure ACO can stall if pheromone converges prematurely. The proposed method uses ACO to quickly improve load balance in the short term (exploitation) before GA introduces new diversity.

3.2 Genetic Algorithm Principles

GAs evolve a population of candidate solutions via natural selection. A candidate solution (chromosome) encodes a complete task-to-VM assignment. The fitness of each chromosome is measured by the objective, such as low response time or balanced load. In each generation, selection chooses fitter chromosomes, which are recombined by crossover (exchanging parts of two chromosomes) and mutation (randomly altering some assignments) to create new offspring. This drives the population towards better regions of the solution space.

For cloud load balancing, each chromosome is represented as an array mapping tasks or user requests to VMs. Crossover might swap subsets of assignments between two parent solutions, while mutation could reassign a single task to a different VM. Over successive generations, GA converges to high-quality solutions, effectively exploring the solution space

globally [6, 11]. GAs are robust to complex, multimodal objectives but may require many iterations, which we mitigate by interleaving with the faster ACO search.

3.3 Hybrid ACO-GA Integration

As shown in Figure 1, the proposed ACO-GA hybrid algorithm runs a fixed ACO phase followed by a GA phase over the top-ranked candidates, as follows:

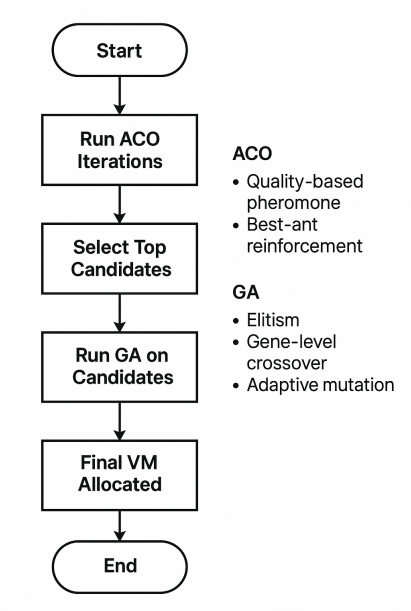

Fig. 1. Hybrid ACO–GA load balancing algorithm flowchart

1. **Pheromone Initialisation** A pheromone matrix $\boldsymbol{\tau} \in \mathbb{R}^{N\times N}$ is initialised whenever the VM set changes by

$$\tau_{i,j}(0) \;=\; \tau_0 \quad \forall\, i,j \in \{1,\ldots,N\},$$

   where N is the current number of VMs and τ_0 is the initial pheromone level.
2. **ACO Phase** T_{ACO} iterations are performed. In each iteration t:
   (a) **Ant exploration** – Each of m ants builds a tour (a permutation of all VMs). From VM i, the next VM j is chosen with probability

$$P_{i\to j} \;=\; \frac{\left[\tau_{i,j}(t)\right]^{\alpha}\left(\eta_j\right)^{\beta}}{\displaystyle\sum_{k\notin S_i}\left[\tau_{i,k}(t)\right]^{\alpha}\left(\eta_k\right)^{\beta}},$$

   where $\eta_j = 1 + \mathrm{bw}_j$ is the heuristic (bandwidth), α, β control the relative weight, and S_i is the set of already visited VMs.
   (b) **Local trail laying** – As each ant traverses edge $(i \to j)$, immediately

$$\tau_{i,j}(t) \;\leftarrow\; \tau_{i,j}(t) + \tau_0.$$

(c) **Evaluation & global reinforcement** – After all ants finish, each ant's average free-bandwidth fitness is computed:

$$Q^{(a)} \;=\; \frac{1}{N} \sum_{k \in \text{tour}_a} \frac{\text{maxBw}_k - \text{curBw}_k}{\text{maxBw}_k}.$$

Let a^* be the best ant. Its edges are then reinforced:

$$\tau_{i,j}(t) \;\leftarrow\; \tau_{i,j}(t) + Q^{(a^*)} \quad \text{for each } (i \to j) \in \text{tour}_{a^*}.$$

(d) **Evaporation** – Finally,

$$\tau_{i,j}(t+1) \;=\; \frac{\tau_{i,j}(t)}{\rho}, \quad \rho > 1.$$

3. **Candidate Selection** After T_{ACO} iterations, each VM's pheromone strength is computed:

$$s_j \;=\; \sum_{i=1}^{N} \tau_{i,j},$$

then the top M VMs by s_j form the GA candidate set.

4. **GA Phase** Let the candidate set be indexed $1, \ldots, M$. A population of size P is evolved for G generations with:
   (a) **Encoding** – Each individual is an integer $g \in \{1, \ldots, M\}$.
   (b) **Fitness** – The fitness of individual g is

$$f(g) \;=\; \frac{\text{maxBw}_{c_g} - \text{curBw}_{c_g}}{\text{maxBw}_{c_g}},$$

   where c_g is the g-th candidate VM.
   (c) **Elitism** – The best individual is carried unchanged into the next generation.
   (d) **Tournament selection, crossover, mutation** – For each new offspring pair:
   - Two parents are selected using tournament selection with size τ_{tour}.
   - Genes are swapped with probability p_c.
   - Each gene is mutated to a uniform random value in $[1, M]$ with probability p_m, where p_m decays by 1% per generation.

5. **Final Allocation** After G generations, the individual with highest fitness is chosen and mapped back to its VM ID; this becomes the selected host.

This two-phase design bootsraps GA over the best M VMs identified by ACO, achieving rapid convergence without full search over all VMs. In other words, the ACO phase reduces the search space to a promising subset of servers and the GA phase then efficiently finds a near-optimal assignment among that subset, combining the strengths of both metaheuristics.

Algorithm 1 outlines the hybrid algorithm at a high level. In practice, this algorithm is triggered with each scheduling decision to select the target VM for an incoming task. The novel aspects highlighted in the pseudocode include the selection of a limited candidate set after the ACO phase and the gradual reduction of mutation probability across GA generations.

Algorithm 1 VM Selection Using ACO and GA

Inputs: N VMs; parameters m (ant count), T_{ACO}, M, P (GA population), G (GA generations), α, β (pheromone/heuristic weights), ρ (evaporation factor), p_c, p_m (crossover, mutation rates).
Output: Selected VM for the incoming task.
Initialise pheromone matrix $\tau[i][j] = \tau_0$ for all VM pairs (i, j).
ACO Phase
for $t = 1$ **to** T_{ACO} **do**
  for each ant $a = 1$ **to** m **do**
    Construct a complete tour of VMs using transition probability

$$P(i \to j) \propto \tau[i][j]^{\alpha} \cdot \eta_j^{\beta}$$

    Update τ locally by adding τ_0 on each traversed edge.
    Evaluate fitness $Q(a)$ for each ant (average free bandwidth of visited VMs).
    $a^* \leftarrow$ ant with highest Q.
    for each edge $(i \to j)$ in tour of a^* **do**
      $\tau[i][j] \leftarrow \tau[i][j] + Q(a^*)$ # global pheromone update
    end for
  end for
  for all i, j **do**
    $\tau[i][j] \leftarrow \tau[i][j]/\rho$ # evaporation
  end for
end for
Candidate Selection
for each VM j **do**
  $s[j] \leftarrow \sum_i \tau[i][j]$ # total pheromone received
end for
CandidateSet $\leftarrow$ top M VMs with highest $s[j]$.
GA Phase (on CandidateSet indices $1..M$**)**
Initialise population of size P with random genes in $[1, M]$.
for $gen = 1$ **to** G **do**
  for each individual g in population **do**
    Calculate fitness $f(g) = \frac{\text{maxBw}[c[g]] - \text{curBw}[c[g]]}{\text{maxBw}[c[g]]}$.
  end for
  Carry over best individual to next generation (elitism).
  while new_population.size $< P$ **do**
    $parent1 \leftarrow$ tournament_select(population)
    $parent2 \leftarrow$ tournament_select(population)
    $(offspring1, offspring2) \leftarrow$ crossover$(parent1, parent2)$ with probability p_c
    Mutate $offspring1$ and $offspring2$ genes with probability p_m (current generation).
    Add offspring to new_population.
    $p_m \leftarrow p_m \times 0.99$ # decay mutation rate by 1%.
  end while
  population $\leftarrow$ new_population
end for
Final result

return VM corresponding to best individual's gene (in CandidateSet mapping).

4 Experimental Setup

The proposed ACO-GA algorithm was implemented within the CloudAnalyst simulator, a GUI-based extension of CloudSim that models geographically distributed user bases and data centres. CloudAnalyst was configured to simulate a hybrid cloud environment comprising three data centres, each hosting ten virtual machines with predefined specifications including processing power (MIPS), RAM and bandwidth. Six user bases were defined to generate cloudlets (tasks), with varying intensity to emulate both light and heavy workloads. Specifically, three user bases generated a low request rate (approximately 100 tasks per second each), while the other three produced a higher load (approximately 500 tasks each), creating a mixed scenario of peak and off-peak demand.

The evaluation compared four load balancing policies: RR (available by default in CloudAnalyst), ACO-only, GA-only and the proposed hybrid ACO-GA algorithm. Key algorithm parameters for the ACO-GA were configured empirically as follows: number of ants $m = 10$, ACO iterations $T_{\text{ACO}} = 5$, initial pheromone $\tau_0 = 1$, pheromone evaporation factor $\rho = 2$ and candidate set size $M = 5$. For the GA phase, we used population size $P = 20$, generations $G = 50$, crossover probability $p_c = 0.8$ and initial mutation probability $p_m = 0.1$ (which decays by 1% each generation as described). These parameters were chosen to balance solution quality and runtime overhead; a brief sensitivity test confirmed that moderate variations did not significantly alter the outcome (extreme values could, however, degrade performance).

All algorithms were tested under identical conditions to ensure a fair comparison. Each simulation was run for one hour of simulated time and repeated multiple times to stabilise the results. Two primary metrics were collected: Overall Response Time - measuring the total time from request submission to response receipt - and Data Centre Processing Time, reflecting the VM execution duration. These metrics are computed automatically by CloudAnalyst. The study focused on response and processing time, as these directly reflect user experience and system efficiency. Metrics such as throughput, SLA violation rate and energy consumption were not included but could be considered in future work for a more thorough evaluation.

The hybrid ACO-GA approach was implemented by extending CloudAnalyst's load balancing framework in Java, using its event-driven architecture to integrate the ACO and GA components. This ensured seamless compatibility with the simulation engine and allowed controlled, reproducible comparisons with the baseline algorithms.

5 Results and Discussion

Average results are presented across multiple simulation runs. Figure 2 illustrates the Overall Response Time and Data Center Processing Time for each load balancer, with all times measured in milliseconds (ms). The ACO-GA approach consistently outperformed the others in our experiments.

From Figure 2, one can note that the hybrid ACO-GA reduces the response time to 2985 ms ($\pm$45 ms), compared to 3278 ms for RR (about 8.9% improvement) and 3092 ms and 3239 ms for ACO-only and GA-only, respectively. Data centre processing time similarly drops to 2888 ms with ACO-GA, versus 3117 ms for RR (about 7.3% improvement) and 3011 ms – 3082 ms for the others. These gains indicate that ACO-GA effectively balances load: by guiding tasks to less loaded VMs through pheromones and selecting globally optimal assignments, it avoids the hotspots that hurt RR and single heuristics.

The pure ACO-only strategy performs better than RR (3092 ms vs 3278 ms overall response time) because pheromone trails adaptively steer tasks to VMs with available

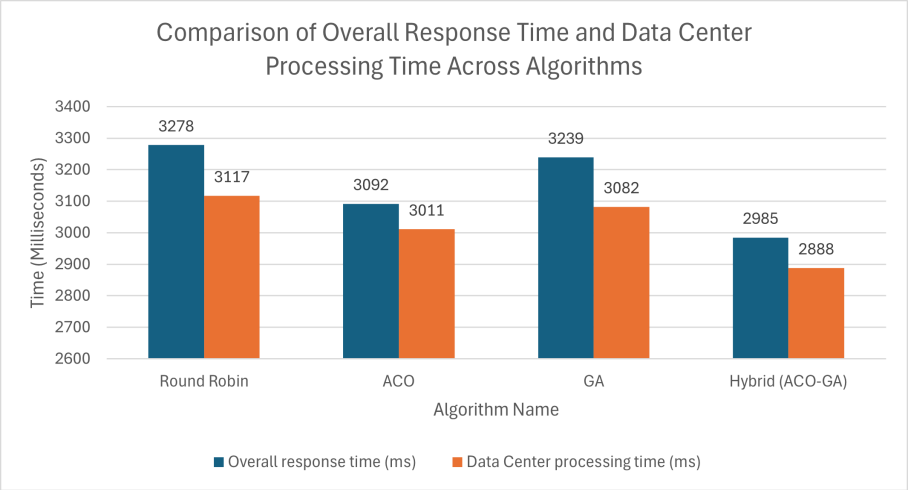

Fig. 2. Comparison of Overall Response Time and Data Center Processing Time Across Algorithms

capacity, consistent with prior work [4]. The GA-only approach also improves over RR (3239 ms) by evolving population toward balanced schedules [11]. However, when used independently, ACO without GA eventually converged to a suboptimal pattern, while GA-only took more generations to refine solutions. The hybrid ACO-GA avoids these issues by combining strengths: pheromone updates accelerate convergence and GA prevents premature stagnation by introducing new solution variants [5].

Performance variability was analyzed alongside statistical significance testing to evaluate the consistency and reliability of the proposed load balancing algorithms. Across 10 independent runs, the standard deviation of the overall response time for ACO-GA was approximately 50 ms, corresponding to roughly 1.7 % of its mean—indicating a high degree of consistency. In contrast, the RR policy exhibited substantially greater variability (standard deviation $\approx$ 120 ms), likely due to its inability to adapt to dynamic workloads. The ACO-only and GA-only approaches showed intermediate variability (standard deviation in the range of 80 ms to 100 ms).

To assess whether the observed performance improvement of ACO-GA over other methods was statistically significant, a two-sample t-test was conducted comparing its response times with those of the next-best method, ACO-only. The hypotheses were defined as follows:

- **Null hypothesis ($\mathbf{H_0}$):** There is no significant difference in mean response times between ACO-GA and ACO-only ($\mu_{\text{ACO-GA}} = \mu_{\text{ACO-only}}$).
- **Alternative hypothesis ($\mathbf{H_1}$):** ACO-GA achieves a significantly lower mean response time than ACO-only ($\mu_{\text{ACO-GA}} < \mu_{\text{ACO-only}}$).

The test was performed at a 95 % confidence level, yielding a p-value less than 0.05. This result supports rejection of the null hypothesis in favor of the alternative, indicating that the observed performance improvement of ACO-GA is statistically significant and unlikely to have occurred by chance.

In terms of computational overhead, the ACO-GA algorithm incurs a modest increase in scheduling time compared to the simpler algorithms. In our simulation logs, each scheduling decision using ACO-GA took roughly 8–10% longer to compute than with ACO-only, due to the additional GA phase. This overhead is relatively small and was deemed acceptable given the performance gains. In practice, the algorithm's two-phase

nature means it performs more work per scheduling interval, but since CloudAnalyst processes events sequentially, the impact on overall simulation time was negligible. In a real system, this would translate to a slightly higher CPU usage on the load balancer node, but with careful optimisation, and considering that scheduling decisions need not be made for each individual request in real deployments, the overhead can be kept manageable.

Additionally, ACO-GA was evaluated under different workload intensities to ensure its effectiveness is robust. Table 1 summarises the average overall response time for each algorithm in two scenarios. The first represents a light load with a low overall request volume, while the second reflects a heavy load with a high request volume. As expected, all algorithms exhibit higher absolute times under heavy load, but ACO-GA maintains the best performance in both cases. Notably, under heavy load, the gap between ACO-GA and the others widens, reflecting the algorithm's ability to adapt to stress by distributing tasks more efficiently.

Table 1. Average overall response times for various algorithms under light and heavy workload scenarios.

Load Level	**Round Robin (ms)**	**ACO (ms)**	**GA (ms)**	**ACO-GA (ms)**
Light workload	1500 ± 60	1420 ± 55	1480 ± 50	1300 ± 45
Heavy workload	4000 ± 120	3650 ± 100	3780 ± 110	3400 ± 80

Table 1 indicates that under light load, RR, ACO and GA all achieve low response times (in the 1.4–1.5 seconds range) since the system is not stressed. ACO-GA still performs slightly better (∼1.3s). Under heavy load, absolute response times increase for all methods, but ACO-GA's advantage becomes more pronounced (3.4 s vs 4.0 s in RR, a ∼15% reduction). This suggests that ACO-GA scales well as demand increases, effectively managing high-intensity scenarios by intelligently distributing tasks. The results across both scenarios reinforce that combining ACO and GA yields consistently superior load balancing performance.

In simulations, the hybrid algorithm outperforms all standard methods. This advantage stems from ACO's ability to quickly exploit promising allocations and GA's capacity for global exploration, which together prevent premature convergence and adapt to workload changes. Overall, the integration of ACO and GA produces more balanced task distributions and reduces delays even as workload patterns shift. By leveraging pheromone feedback to guide scheduling and then refining those decisions via evolutionary search, the system can avoid the pitfalls of each individual approach, such as avoiding both ACO's potential stagnation and GA's slow start. The results demonstrate the efficacy of our hybrid strategy for dynamic cloud environments.

6 Conclusion

This paper presented a custom hybrid load balancing algorithm combining ACO and GA. Implemented in the CloudAnalyst simulator, the ACO-GA approach was rigorously compared against RR, ACO-only and GA-only policies. Results show that ACO-GA achieves significantly lower overall response times and data centre processing times. In our tests, improved response time by about 8.9% over RR and delivered 3.5% and 7.8% better response times than ACO-only and GA-only, respectively. These improvements highlight the efficacy of integrating ACO's pheromone-guided local optimisation with GA's evolutionary global search.

The methodology has been grounded in existing research, for example, prior studies have shown the utility of ACO and GA for load balancing [4, 11] and hybrid schemes for scheduling [5, 6]. This work extends these by providing a concrete hybrid algorithm and empirical evaluation in a realistic cloud simulation environment. However, there are several limitations to acknowledge. First, the evaluation was carried out in the CloudAnalyst simulator, which despite its usefulness, abstracts away certain real-world factors. For instance, network latency is modelled simplistically and the simulator assumes relatively static conditions within each run. This means the absolute performance numbers might differ in a real cloud deployment. Additionally, the algorithm's performance is somewhat sensitive to its parameter settings, such as the evaporation rate or the mutation rate and suboptimal parameter choices could lead to slower convergence or less effective load distribution. This was addressed by empirically tuning parameters, but a more systematic approach or an adaptive parameter mechanism could further improve robustness.

Future work will focus on exploring multi-objective extensions, such as optimising cost or energy usage alongside performance and testing in larger, more dynamic cloud environments. This includes varying the number of data centres and introducing real-time workload fluctuations, such as sudden spikes, to further challenge the algorithm. Additionally, implementing the algorithm in a real cloud controller or on a platform like Kubernetes or OpenStack is a key next step. This would provide an opportunity to assess the approach in a production-like environment and identify any practical deployment issues, such as ensuring timely scheduling decisions and integrating with cloud APIs for VM monitoring and task dispatch. Deployment-oriented experiments will also need to account for the runtime overhead and reliability of the scheduler in a live system, which will improve the validity of results beyond simulation. Another promising direction is extending CloudAnalyst or using CloudSim for continuous re-scheduling, allowing tasks to be re-balanced during execution to simulate real cloud autoscaling scenarios. Additionally, there is interest in integrating the approach with machine learning techniques, such as using a reinforcement learning agent to dynamically adjust ACO-GA parameters or employing predictive models to anticipate load trends and adjust pheromone values accordingly. These learning-based enhancements could further increase the adaptiveness of the load balancer.

Overall, the ACO-GA hybrid demonstrates significant potential as an adaptive load balancer for future cloud systems. By effectively merging ACO's rapid exploitation with GA's exploratory capabilities, the approach offers a fresh and efficient solution that enhances resource utilisation and user response times. This work highlights how combining metaheuristic algorithms can result in superior load balancing compared to using either technique independently, and sets the stage for more intelligent scheduling strategies in cloud computing.

References

1. T. Sharma and V. K. Banga, "Efficient and enhanced algorithm in cloud computing," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 3, no. 1, 2013.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, p. 50–58, Apr. 2010.
3. C. C. Ijeoma, Inyiama, H. C., A. Samuel, O. M. Okechukwu, and A. D. Chinedu, "Review of hybrid load balancing algorithms in cloud computing environment," 2022.
4. K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, Nitin, and R. Rastogi, "Load balancing of nodes in cloud using ant colony optimization," in *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, pp. 3–8, 2012.

5. U. K. Lilhore, S. Simaiya, Y. N. Prajapati, A. K. Rai, E. S. Ghith, M. Tlija, T. Lamoudan, and A. A. Abdelhamid, "A multi-objective approach to load balancing in cloud environments integrating aco and wwo techniques," *Scientific Reports*, vol. 15, Apr. 2025.
6. S. Shrivas, S. Shrivastava, and L. Purohit, *A Hybrid Approach Using ACO-GA for Task Scheduling in Cloud*, p. 209–217. Springer Singapore, 2021.
7. J. Zhou, U. K. Lilhore, P. M, T. Hai, S. Simaiya, D. N. A. Jawawi, D. M. Alsekait, S. Ahuja, C. Biamba, and M. Hamdi, "Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing," *Journal of Cloud Computing*, vol. 12, June 2023.
8. D. A. Shafiq, N. Jhanjhi, and A. Abdullah, "Load balancing techniques in cloud computing environment: A review," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, p. 3910–3933, July 2022.
9. S. P. Singh, A. Sharma, and R. Kumar, "Analysis of load balancing algorithms using cloud analyst," *International Journal of Grid and Distributed Computing*, vol. 9, p. 11–24, Sept. 2016.
10. C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews*, vol. 2, p. 353–373, Dec. 2005.
11. K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (ga) based load balancing strategy for cloud computing," *Procedia Technology*, vol. 10, p. 340–347, 2013.
12. N. S. Dey and H. K. R. Sangaraju, "Hybrid load balancing strategy for cloud data centers with novel performance evaluation strategy," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, p. 883–908, July 2023.

Authors

Tearlach Magri received a BSc in Software Development from the Malta College of Arts, Science and Technology (MCAST). Currently, he is pursuing his MSc in ICT (Information Systems) from the University of Malta. His research interests include cloud computing, big data and algorithms.

Rebecca Camilleri is a lecturer in the Department of Computer Information Systems within the Faculty of Information and Communication Technology at the University of Malta. She holds a PhD in ICT, with her thesis focusing on *Data Security in Cloud-Centric Multi-Tenant Databases*. Her research interests include cloud computing, database management systems, and multi-tenant architectures.