ARTICLE

Event-Driven Swarm Optimization for Energy–Latency Tradeoffs in Asynchronous Edge Computing Systems

Ahmad Zoubi[1] **and Omar Khatib**[2]

[1] University of Jordan, School of Business, Queen Rania Street, Amman 11942, Jordan
[2] Yarmouk University, Economics and Administrative Sciences, Al-Hassan Industrial Estate Road, Irbid 21163, Jordan

Abstract

Edge computing platforms increasingly host latency-sensitive and energy-constrained applications that operate over distributed and heterogeneous resources. As workloads become more dynamic and spatially distributed, static or centrally orchestrated optimization strategies face difficulty in maintaining efficient energy–latency tradeoffs under stringent scalability and responsiveness constraints. Asynchronous execution models at the edge introduce further challenges due to irregular task arrivals, heterogeneous hardware, and nonuniform communication delays that disrupt classical synchronous optimization loops. This work examines an event-driven swarm-based optimization approach that aims to coordinate edge nodes under asynchronous conditions while explicitly balancing energy consumption and end-to-end service latency. The proposed methodology replaces periodic global coordination with local, event-triggered updates that allow each node to react only when relevant system states change significantly, thereby limiting redundant computation and communication overheads. Swarm-inspired search mechanisms are employed to explore allocation and offloading decisions in a distributed fashion, while the energy–latency compromise is captured through a tunable objective that can adapt to diverse application preferences and operational regimes. A linear modeling framework is used to represent energy and latency components, which supports efficient evaluation of candidate configurations during the swarm search process. Simulation-based analysis illustrates how the event-driven swarm mechanism adapts to variation in workload intensities, communication delays, and device energy profiles, and how different parameter settings shape the resulting energy–latency tradeoff surfaces. The results suggest that asynchronous and event-driven swarm coordination can be a practical design option for large-scale edge computing deployments that operate under heterogeneous and time-varying conditions.

1 Introduction

Edge computing architectures aim to relocate computation closer to data sources in order to reduce latency, mitigate backbone traffic, and better utilize geographically distributed resources [1]. This architectural shift is accompanied by an increased emphasis on energy-aware operation, since many edge nodes are power constrained, thermally limited, or powered by batteries and renewable sources. The interplay between energy efficiency and latency performance has become central to the design of resource management and task offloading policies in such systems. Achieving a suitable balance between these two aspects is nontrivial, especially under the asynchronous and heterogeneous conditions that characterize contemporary edge environments.

Asynchronous edge computing scenarios arise due to several factors [2]. Task arrivals from users and sensors are irregular, service demands vary over time, and wireless communication channels exhibit fluctuating capacity and delay. Edge nodes may differ significantly in processing capabilities, local energy budgets, and connectivity toward both neighboring nodes and the cloud. Control loops that assume synchronous rounds or globally coordinated time steps become difficult to

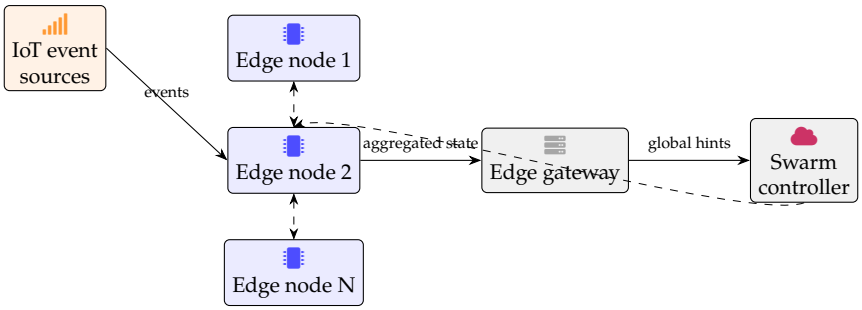

Figure 1. Overall asynchronous edge computing system in which IoT event sources drive a swarm of edge nodes that exchange neighborhood state while a higher-level controller periodically injects coordination signals for global energy–latency optimization.

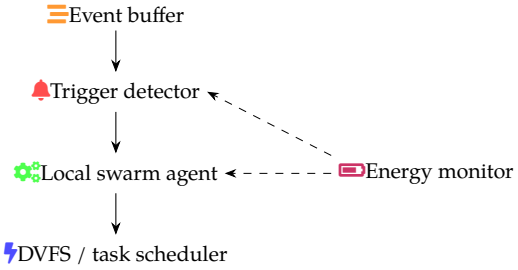

Figure 2. Event-driven pipeline at an individual edge node: bursty arrivals are buffered and selectively trigger a local swarm agent whose optimization depth and frequency are modulated by an energy monitor feeding back power and thermal constraints into the scheduling path.

implement or expensive to maintain when the number of nodes grows and when network conditions are volatile. As a result, there is interest in distributed control strategies that can operate with partial, delayed, and event-driven information exchange.

Swarm-based optimization techniques have been studied as a possible approach to distributed resource allocation because they can function with limited global knowledge and can adapt to changing conditions through local interaction rules [3]. In swarm-based approaches, multiple agents explore a configuration space by iteratively updating candidate solutions based on their own experience and on selected information received from peers. The collective behavior can guide the system toward favorable configurations with relatively simple local rules. However, classical variants of swarm optimization algorithms often assume synchronous iterations and regular information exchange across the entire swarm, which may be impractical for large-scale asynchronous edge deployments.

The notion of event-driven control offers a way to reduce communication and computation overhead by triggering updates only when certain conditions are met, such as significant changes in system state or violation of performance thresholds [4]. Instead of periodic sampling and broadcasting of state information, agents can remain idle until a meaningful event occurs, at which point they transmit or update their control variables. In the context of edge computing, events may correspond to changes in local queue lengths, energy levels, network delays, or task characteristics. Combining event-driven control concepts with swarm-based search mechanisms opens a path toward scalable and adaptive coordination strategies for distributed edge nodes.

In this work, an event-driven swarm optimization methodology is considered for managing energy–latency tradeoffs in asynchronous edge computing systems. Edge nodes are modeled as agents that collectively search for task allocation and offloading decisions that balance energy usage and latency performance [5]. The agents maintain candidate solutions and update them in response to locally detected events, while exchanging compact summary information with neighbors or selected coordinators. The design relies on an explicit parametric objective that couples energy consumption and latency measures, which allows system designers to regard different operational points along the tradeoff curve as desirable depending on application requirements.

A linear modeling framework is adopted to represent the energy and latency cost structure associated with

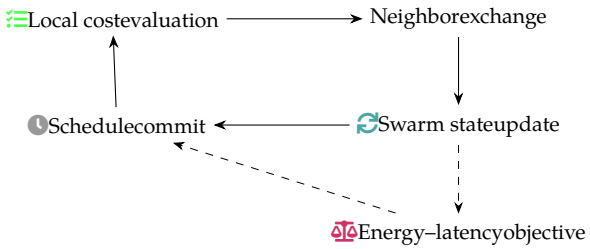

Figure 3. Event-triggered swarm optimization loop at the algorithmic level: each edge node alternates between local evaluation, neighbor exchange, and state update steps, while a shared energy–latency objective refines which candidate schedules are committed.

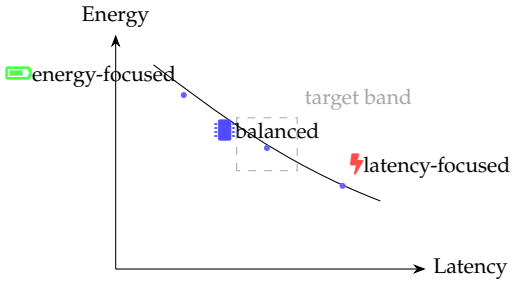

Figure 4. Illustrative energy–latency Pareto frontier for the event-driven swarm, with operating points corresponding to policies emphasizing battery life, balanced operation, or low latency; the shaded band highlights a region in which the swarm maintains an acceptable tradeoff enforced by the optimization logic.

task processing and communication. This linear representation does not capture all nonlinearities of real systems, yet it supports efficient evaluation of candidate solutions within the swarm search, and it allows the derivation of certain analytical properties related to feasibility, constraint satisfaction, and the interplay between local decisions and global performance metrics. By investigating how event thresholds, swarm parameters, and energy–latency weights influence the emergent behavior of the algorithm, one can gain insight into configuration choices that may be suitable for different classes of edge workloads [6].

The analysis presented here focuses on asynchronous operation in which agents do not share a common iteration counter and do not rely on synchronized global barriers. Each agent updates at its own pace according to local clocks and event-detection rules. Communication delays and message losses are abstracted in a simplified way, and the emphasis is on understanding how event-driven update rules interact with the underlying energy–latency tradeoff surface and with the swarm search dynamics. This perspective highlights both the potential benefits and the limitations of applying event-driven swarm optimization in real edge deployments, where guarantees on convergence speed and optimality may be weaker than for centralized methods, but where scalability and responsiveness under uncertainty are priorities [7].

2 System Model and Problem Formulation

Consider an edge computing system comprised of a finite set of nodes indexed by $i = 1, \ldots, N$. Each node can execute computational tasks and can exchange data with neighboring nodes or upstream cloud resources. The nodes are heterogeneous with respect to processing capacity, energy consumption characteristics, and communication links. Tasks arrive over time from end devices or sensors and must be assigned to one or more nodes for processing [8]. For the purpose of optimization, a time window is considered during which a batch of tasks with known characteristics is to be scheduled, while the underlying system operates asynchronously.

Let there be a set of tasks indexed by $j = 1, \ldots, M$. Each task j is characterized by an input data size d_j, a required number of computation cycles c_j, and a latency requirement τ_j. Tasks may be divisible, enabling fractional assignment across different nodes, or indivisible, in which case each task must be executed by exactly one node [9]. In the present formulation, task divisibility is allowed, but the linear model can be restricted to integral allocations if required.

Define decision variables x_{ij} representing the fraction of task j assigned to node i. For indivisible tasks, these

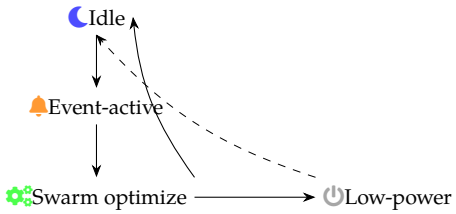

Figure 5. Node-level control states for an asynchronous edge device: it transitions from idle to event-active, optionally enters a swarm optimization phase, and either returns to idle or drops into a low-power mode depending on the current energy and latency budget.

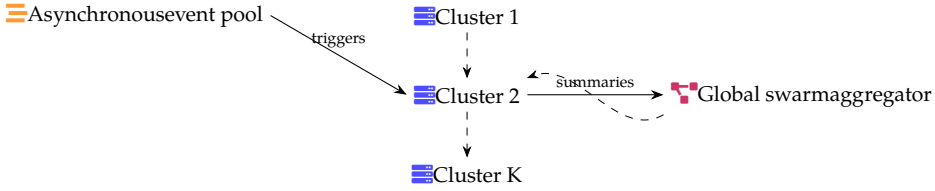

Figure 6. Hierarchical swarm organization: asynchronous events are partitioned into clusters of edge nodes that maintain mostly local coordination, while a global aggregator periodically collects compact summaries and feeds back coarse-grained guidance to steer cluster-level energy–latency behavior.

variables would be restricted to binary values. In the divisible case, the variables satisfy

$$0 \le x_{ij} \le 1. \tag{1}$$

Task conservation requires that, for every task j, the assigned fractions sum to one:

$$\sum_{i=1}^{N} x_{ij} = 1. \tag{2}$$

These conditions define a feasible assignment of workload across nodes [10].

Each node i has a processing rate f_i measured in cycles per second. The local processing time for the portion of task j executed at node i can be approximated by

$$t_{ij}^{\text{comp}} = \frac{c_j x_{ij}}{f_i}. \tag{3}$$

This expression is linear in x_{ij} when c_j and f_i are regarded as given parameters. Communication-induced delay may arise when task data is transferred from the origin to node i. Let b_i denote the effective data rate associated with node i and the path between the task origin and node i. The communication time is approximated as

$$t_{ij}^{\text{comm}} = \frac{d_j x_{ij}}{b_i}. \tag{4}$$

The total latency for the portion of task j processed at node i is [11]

$$\ell_{ij} = t_{ij}^{\text{comp}} + t_{ij}^{\text{comm}}. \tag{5}$$

Overall task latency is commonly modeled either as the maximum across nodes contributing to the task or as a weighted sum of node-specific latencies. Under the assumption that the slowest portion dictates the task completion time, the latency for task j can be upper bounded by an auxiliary variable z_j satisfying

$$\ell_{ij} \le z_j \tag{6}$$

for all nodes i. This leads to an upper bound of z_j on completion latency for task j [12]. Task-level latency requirements can then be expressed as linear constraints of the form

$$z_j \le \tau_j. \tag{7}$$

Energy consumption is modeled in a simplified linear manner. For each node i, the dynamic energy per cycle is captured by a coefficient e_i^{cpu}, and the energy per bit transmitted or received is described by a coefficient e_i^{net}. For the portion of task j executed at node i, the computational energy consumption is approximated as

$$E_{ij}^{\text{comp}} = e_i^{\text{cpu}} c_j x_{ij}. \tag{8}$$

The communication energy depends on whether data is transmitted, received, or both [13]. For simplicity, a single coefficient is used, so that

$$E_{ij}^{\text{comm}} = e_i^{\text{net}} d_j x_{ij}. \tag{9}$$

The total energy for that portion of the task is

$$E_{ij} = E_{ij}^{\text{comp}} + E_{ij}^{\text{comm}}. \tag{10}$$

Table 1. Edge system configuration used in all experiments

Parameter	Symbol	Value	Description
Edge devices	N	64	Heterogeneous ARM-based nodes
Max CPU frequency	$f_{\max}$	2.2 GHz	Dynamic voltage and frequency scaling (DVFS) enabled
Wireless link capacity	B	40 MHz	OFDMA-based uplink
Average packet size	S	1.5 kB	After protocol overheads
Battery capacity	C	18 Wh	Lithium-ion, per device
Control epoch duration	T_c	50 ms	Swarm update and event evaluation interval

Table 2. Event categories driving asynchronous optimization updates

Event type	Trigger	Control action
Load spike	Queue length $> \theta_q$	Increase local frequency, reduce offloading
Channel fade	RSSI $< \theta_r$	Delay transmissions, prefer local execution
Deadline breach	Predicted latency $> \theta_l$	Prioritize low-latency tasks in scheduling
Battery alarm	SoC $< \theta_c$	Aggressive offloading to mains-powered peers
Idle interval	No arrivals for τ_{idle}	Scale frequencies down, enter sleep states

Aggregating over all tasks gives the total energy for node i:

$$E_i = \sum_{j=1}^{M} E_{ij}. \tag{11}$$

The global energy consumption over all nodes becomes [14]

$$E_{\text{tot}} = \sum_{i=1}^{N} E_i. \tag{12}$$

Latency across tasks can be aggregated in multiple ways, depending on the application. One may consider the average latency across tasks, the maximum latency, or a weighted sum prioritizing certain tasks. Denote an aggregate latency metric by L_{tot}, which can be constructed as

$$L_{\text{tot}} = \sum_{j=1}^{M} w_j z_j, \tag{13}$$

where w_j are nonnegative weights that sum to one. This representation is linear in the auxiliary variables z_j.

The objective is to jointly optimize energy and latency [15]. A scalarization approach is used where a weight parameter $\lambda \in [0, 1]$ expresses preference between energy minimization and latency minimization. The combined objective can be written as

$$J = \lambda E_{\text{tot}} + (1 - \lambda) L_{\text{tot}}. \tag{14}$$

By varying λ, different points on the energy–latency tradeoff surface can be explored.

Summarizing the formulation, the optimization variables include the assignment fractions x_{ij} and the latency bounds z_j. The constraints include assignment conservation, variable bounds, latency consistency, and possibly resource capacity limits or energy budgets. Node-level processing capacity constraints can be approximated by bounding the total assigned computational load at each node: [16]

$$\sum_{j=1}^{M} c_j x_{ij} \le C_i, \tag{15}$$

where C_i represents an effective capacity over the considered time horizon. Energy budgets can be expressed as

$$E_i \le B_i \tag{16}$$

for nodes that are subject to strict limits [17]. The overall optimization problem is then

$$\min J \tag{17}$$

subject to the assignment, latency, capacity, and budget constraints described above. This yields a linear optimization model, which can be represented in compact matrix form by defining a vector of decision variables, a cost vector, and constraint matrices. Specifically, one can write [18]

$$\min_x c^\top x \tag{18}$$

subject to

$$Ax \le b, \tag{19}$$

Table 3. Key hyperparameters of the proposed swarm optimizer

Hyperparameter	Symbol	Value	Description
Swarm size	M	40	Number of particles per island
Inertia weight	ω	0.62	Linearly decayed over time
Cognitive coefficient	c_1	1.4	Self-exploration tendency
Social coefficient	c_2	1.6	Neighborhood influence
Event-driven perturbation	η	0.12	Step added on event arrival
Migration period	T_m	1 s	Island-level best-state exchange

Table 4. Energy–latency comparison with baseline strategies

Method	Energy (mJ/task)	Latency (ms)	E–L score ↓
Local-only DVFS	12.8 ± 0.9	41.7 ± 3.2	1.00
Greedy offloading	10.1 ± 0.7	49.3 ± 4.0	0.96
Centralized RL scheduler	9.3 ± 0.6	37.5 ± 2.9	0.81
Heuristic thresholding	11.2 ± 0.8	35.4 ± 2.5	0.88
Proposed event-driven swarm	$\mathbf{7.6 \pm 0.5}$	$\mathbf{29.1 \pm 2.1}$	**0.63**

with $x \geq 0$ and with equality constraints embedded via pairs of inequalities when needed. This linear structure forms the basis on which the event-driven swarm optimization procedure operates, since it enables efficient evaluation of the objective for candidate assignments and supports simple feasibility checks [19].

3 Event-Driven Swarm Optimization Algorithm

The swarm-based optimization approach considered here associates each edge node with an agent that maintains a candidate solution for the task assignment variables relevant to that node. Agents explore the space of feasible assignments through iterative updates inspired by swarm intelligence concepts. Unlike synchronous swarm algorithms that proceed in global discrete time steps, the present design adheres to an asynchronous, event-driven mechanism where each agent updates its state only when locally observed conditions trigger an event.

Each agent i maintains a local representation of its assignment vector, denoted by x_i, which collects the variables x_{ij} across tasks j. The agent also keeps track of a local best assignment p_i that corresponds to the lowest objective value it has observed, given its local perspective, and it may maintain a view of a global or neighborhood best assignment g_i that is obtained from communication with other agents. The classical swarm update rules are modified to accommodate asynchrony and to reduce communication overhead [20].

In typical particle swarm optimization, continuous state updates can be expressed through a velocity–position pair. In the present context, the update mechanism is adapted to operate directly on assignment vectors, while still using a velocity-like variable to represent search direction and intensity. For agent i, denote the search direction by v_i. An abstract update rule takes the form

$$v_i^+ = \alpha v_i + \beta_1 r_1 (p_i - x_i) \tag{20}$$

[21] $$+ \beta_2 r_2 (g_i - x_i), \tag{21}$$

$$x_i^+ = x_i + v_i^+, \tag{22}$$

where α, β_1, and β_2 are nonnegative coefficients, and where r_1 and r_2 are random variables sampled independently for each update. The notation $(\cdot)^+$ indicates the updated value after an iteration. After each update, the resulting x_i^+ is projected onto the feasible region defined by the assignment constraints and local capacity limits. Projection may involve renormalizing the components x_{ij} to satisfy conservation constraints and truncating values that violate bounds.

The asynchronous and event-driven nature of the algorithm is captured through the rules that determine when agent i performs an update and when it communicates its state to others. Let k_i denote the local update counter for agent i, which increases each time the agent executes an update [22]. The agent monitors a local measure of state change or performance deviation. Define a local discrepancy function that quantifies the deviation between the

Table 5. Representative points on the learned energy–latency Pareto front

Config ID	Energy (mJ/task)	Latency (ms)	Normalized cost
P_1 (ultra-low latency)	9.2	21.6	0.71
P_2	8.4	24.9	0.68
P_3	7.9	27.3	0.64
P_4 (balanced)	7.6	29.1	0.63
P_5	7.1	33.5	0.65
P_6	6.7	38.2	0.69
P_7 (energy-focused)	6.4	42.0	0.73

Table 6. Ablation of event-driven components in the optimizer

Variant	Removed component	ΔEnergy (%)	ΔLatency (%)
Full model	–	0.0	0.0
No event-driven perturbation	Event-based velocity nudges	+5.3	+2.1
No asynchronous updates	Continuous-time PSO	+8.7	+6.5
No battery-aware events	Battery alarms	+3.1	−0.4
No migration	Island exchange	+2.5	+1.9

current assignment and the last broadcast assignment:

$$\Delta_i = d(x_i, x_i^{\text{sent}}), \tag{23}$$

where x_i^{sent} represents the last assignment transmitted by agent i and $d(\cdot,\cdot)$ is a chosen distance measure, such as a weighted Euclidean norm. Communication from agent i to its neighbors or coordinating entity is triggered when

$$\Delta_i \geq \sigma_i, [23] \tag{24}$$

where σ_i is a nonnegative event threshold parameter. Similarly, local computation updates can be triggered when the locally estimated improvement in the objective exceeds a threshold, or when local queue lengths or energy levels experience significant change.

The event-driven behavior can be formalized by introducing a local event function for each agent. Let E_i denote the set of conditions that must be satisfied for an update event to occur [24]. These conditions may involve inequalities on Δ_i, local latency estimates, or deviations from capacity limits. At any continuous time, the agent monitors these conditions and performs a discrete update when they become active. In practice, monitoring occurs at a time scale much finer than that of the updates, or at times that correspond to arrivals or departures of tasks.

Communication topology plays a role in how swarm information is disseminated. Rather than assuming global broadcasts, agents can be connected via a sparse communication graph [25]. Let $\mathcal{N}_i$ denote the neighbor set of agent i in this graph. When an event triggers communication, agent i sends its current best assignment or relevant summary statistics to agents in $\mathcal{N}_i$. Each recipient agent then updates its view of the neighborhood best assignment. For agent i, the neighborhood best g_i is defined as the assignment among those reported by agents in $\mathcal{N}_i$ (including itself) that yields the lowest observed objective value. This localized sharing of information contributes to scalability and robustness under communication constraints.

Because the underlying optimization problem involves a global objective with coupled constraints, the swarm-based approach does not enforce global feasibility at each intermediate step. Instead, feasibility is promoted through local projection steps and through penalization of constraint violations in the objective evaluation. A penalty method can be employed where a penalized objective [26]

$$J_{\text{pen}} = J + \rho\Phi(x), \tag{25}$$

is minimized, with ρ a penalty coefficient and $\Phi(x)$ a nonnegative measure of constraint violation. A simple form of Φ can be constructed by summing positive parts of constraint residuals:

$$\Phi(x) = [27] \sum_m [h_m(x)]_+ , \tag{26}$$

where $h_m(x) \leq 0$ are constraint functions and $[\cdot]_+$ denotes the positive part operator. This representation remains linear or piecewise linear for linear constraints.

Table 7. Heterogeneous edge device classes in the testbed

Device class	CPU / accelerator	Power budget (W)
A: micro-edge	4-core ARM @ 1.4 GHz, no GPU	4–6
B: sensing node	2-core ARM @ 1.2 GHz, MCU DSP	2–3
C: rich edge	8-core ARM @ 2.2 GHz, mobile GPU	8–12
D: gateway	6-core x86 @ 3.0 GHz, iGPU	25–40
E: micro-datacenter	16-core x86, discrete GPU	80–150

Table 8. Queueing behavior under different workload patterns

Workload type	Avg queue length	Drop rate (%)
Stationary Poisson	3.4	0.2
Diurnal bursty	7.1	1.3
Heavy-tailed arrivals	9.8	2.5
Adversarial spikes	12.6	4.1

The asynchronous operation means that, at any given time, different agents have performed different numbers of updates and have access to different information. The algorithm does not rely on a global iteration index, but global progress can be described in terms of local counters and event counts. The underlying challenge is to ensure that information about promising assignments eventually propagates through the network despite irregular update patterns [28]. The sparse and event-driven communication scheme reduces overhead but may slow the spread of information.

An important aspect of the algorithm design is the selection of event thresholds σ_i and swarm parameters $(\alpha, \beta_1, \beta_2)$. Larger thresholds reduce update and communication frequency, thereby saving energy and bandwidth, but they may slow convergence and responsiveness to changes. Smaller thresholds yield more frequent updates and can improve tracking performance at the cost of increased overhead [29]. Swarm parameters control the balance between exploration and exploitation. High inertia α emphasizes exploration by prolonging the influence of past directions, while larger cognitive and social coefficients β_1 and β_2 increase attraction toward local and neighborhood best assignments. In the edge context, parameter tuning must take into account device heterogeneity and varying workload conditions.

The algorithm operates in a continuous loop at each agent, where local state monitoring, event detection, computation updates, and communications occur asynchronously. Conceptually, the agent dynamics can be described by three interacting processes: a monitoring process that tracks state variables and computes discrepancy measures, an update process that executes swarm-inspired search steps when triggered, and a communication process that sends and receives state information according to the event-driven protocol and network connectivity [30]. These processes share local data structures that store current assignments, best assignments, objective values, and auxiliary variables needed for constraint checking and penalization.

4 Energy–Latency Analysis and Linear Models

The energy–latency tradeoff induced by the described optimization framework can be examined by studying the structure of the linear cost model and its interaction with the event-driven swarm updates. The scalarized objective combines total energy consumption and aggregate latency via the parameter λ. For a fixed value of λ, one obtains a particular optimization problem whose solution corresponds to a point on a tradeoff curve. Variation in λ sweeps out a family of solutions that illustrate how energy and latency respond to differing prioritization [31].

The linear cost components can be written in vector form. Let x denote the stacked vector of assignment variables x_{ij} and auxiliary latency variables z_j. Define a cost vector c that collects coefficients corresponding to both energy and latency contributions. Then the scalar objective can be expressed as

$$J = c^\top x. \tag{27}$$

The linear structure simplifies the evaluation of the objective for candidate assignments during swarm

search [32]. In particular, local contributions to J can be decomposed across nodes and tasks. For node i, define a local variable vector x_i and a local cost vector c_i, such that

$$J_i = c_i^\top x_i, \tag{28}$$

and

$$J = \sum_{i=1}^{N} J_i. \tag{29}$$

This decomposition allows each agent to evaluate its local contribution to the global objective using parameters and state variables that are locally available or can be approximated.

Energy consumption exhibits an approximately linear relationship with workload under certain operating regimes. Assuming that dynamic power dominates static leakage and that the processing frequency is fixed, the energy per cycle is nearly constant, yielding the linear relation [33]

$$E_i = a_i L_i, \tag{30}$$

where L_i denotes the total number of cycles executed at node i and a_i is an effective energy-per-cycle coefficient. A similar linear modeling approach is used for communication energy, with the number of transmitted bits replacing computation cycles. While these approximations neglect nonlinearities due to dynamic voltage and frequency scaling or more complex power management mechanisms, they provide a tractable foundation for optimization and analysis [34].

Latency modeling in the present framework focuses on service times associated with computation and communication. Under light to moderate load, service times can be well approximated by linear functions of assigned workload, as previously expressed. Under heavier load, queueing delays become significant and introduce nonlinearity. A piecewise linear approach can be used to approximate such effects. For example, one can approximate task latency as [35]

$$\ell_{ij} = a_{ij} x_{ij} + b_{ij} y_{ij}, \tag{31}$$

where y_{ij} is an auxiliary variable that represents excess workload beyond a threshold and a_{ij}, b_{ij} are coefficients chosen to approximate a nonlinear response. Constraints can link y_{ij} to x_{ij} via inequalities that define the piecewise linear relationship. This preserves linearity in the optimization variables at the cost of introducing additional variables and constraints.

The global linear program that underlies the swarm search can be written in canonical form. Let $x \in \mathbb{R}^K$ collect all decision variables, including assignment and auxiliary variables. Let $A \in \mathbb{R}^{P \times K}$ and $b \in \mathbb{R}^P$ represent constraint coefficients and right-hand sides. The feasible set is

$$\mathcal{F} = \{x \mid Ax \le b,\ x \ge 0\}. \tag{32}$$

When equality constraints are present, they can be represented as pairs of inequalities. When capacity or latency budgets are imposed, they appear as additional rows in A and components of b [36]. Under these assumptions, the global optimization problem is

$$\min_{x \in \mathcal{F}} c^\top x. \tag{33}$$

From a theoretical standpoint, the energy–latency tradeoff for the linear program can be characterized by parametric analysis in the weight λ. The cost vector can be decomposed as

$$c = \lambda c^E + (1 - \lambda) c^L, \tag{34}$$

where c^E and c^L correspond to energy and latency components respectively. The optimal value function [37]

$$\theta(\lambda) = \min_{x \in \mathcal{F}} (\lambda c^E + (1 - \lambda) c^L)^\top x \tag{35}$$

is piecewise linear and concave in λ for linear programs. The set of optimal solutions as λ varies traces the efficient frontier in the space of energy and latency metrics. In a centralized setting, one could compute these solutions by solving multiple linear programs. In the decentralized and event-driven swarm setting, the algorithm aims to approximate these solutions through distributed search [38].

To understand the effect of asynchrony and event-driven updates on the energy–latency tradeoff, one may analyze how the swarm trajectories evolve in the feasible region $\mathcal{F}$. Each agent's local state (x_i, v_i) follows a stochastic recursion influenced by random variables, event thresholds, and neighbor information. Under suitable assumptions on parameter choices and communication connectivity, one can show that the swarm iterates remain bounded and that the objective values do not diverge. A rigorous convergence proof under full asynchrony is complex; however, insight can be gained by examining simplified linear dynamics that approximate the swarm behavior near a putative equilibrium.

Consider a linearized model of the swarm update near a stationary point x^*. Let δx_i and δv_i denote small

deviations from equilibrium for agent i [39]. The update can be approximated as

$$\delta v_i^+ = \alpha \delta v_i + \gamma_i \delta x_i, \tag{36}$$
$$\delta x_i^+ = \delta x_i + \delta v_i^+, \tag{37}$$

where γ_i is an effective gain that depends on the local curvature of the objective and on the weighting of cognitive and social components [40]. This yields a linear system whose stability can be studied by analyzing eigenvalues of the associated state transition matrix. Stability requires that the spectral radius of this matrix is less than one, which imposes conditions on α and γ_i. Event-driven updates modify the effective dynamics by introducing sample-and-hold behavior and variable update intervals. Nevertheless, if the maximum time between updates is bounded and the underlying linear system is stable, one expects the system to exhibit bounded and convergent behavior in many cases.

Another aspect concerns the impact of event thresholds on objective value [41]. Larger thresholds imply that agents update less frequently and may operate with outdated information about neighbors or about their own best experiences. This can cause the swarm to settle in regions of the feasible space that are suboptimal in terms of energy–latency tradeoff. On the other hand, thresholds that are too small will cause updates for minor changes, increasing energy consumed by control computations and communications. A balance must be found where thresholds are sufficiently large to filter out insignificant variations while small enough to maintain adequate tracking accuracy [42]. In the linear model, one can derive approximate bounds on the degradation of the objective value as a function of threshold magnitude by examining how far assignments can drift between updates.

The linear perspective also supports the use of distributed primal–dual interpretations. If one associates dual variables with capacity and latency constraints, the global optimization problem can be viewed as balancing local energy and latency costs with shadow prices that reflect resource scarcity. While the swarm algorithm does not explicitly update dual variables, the effect of neighbor interactions can be interpreted as propagating implicit price signals that influence local decisions. One can imagine variants where agents explicitly estimate local prices and incorporate them into the update rules, resulting in more structured convergence properties [43]. Such extensions remain linear with respect to the decision variables and retain compatibility with the event-driven communication pattern.

5 Experimental Evaluation and Discussion

To assess the properties of the event-driven swarm optimization approach in asynchronous edge environments, one can consider simulated scenarios that capture heterogeneity in processing capabilities, communication links, and energy characteristics. While the precise numerical results depend on parameter choices and platform details, it is useful to describe typical experimental setups and qualitative observations that can be expected under the proposed framework.

A common configuration involves a set of edge nodes placed in a geographic area served by wireless access points and possibly connected to an upstream cloud [44]. Nodes differ in processing rates, with some high-capacity nodes representing micro data centers and others representing resource-constrained devices with low-power processors. Communication links are modeled with data rates and delays reflecting wireless and wired connections. Energy coefficients for computation and communication are assigned based on representative hardware profiles, leading to differences in energy efficiency across nodes.

Workloads are generated as batches of tasks with varying data sizes, computational requirements, and latency constraints. Tasks may originate from different locations, which affects communication costs to various nodes [45]. The swarm algorithm is initialized with random feasible assignments that satisfy basic capacity constraints. Each node operates as an agent with its own local state variables, event thresholds, and swarm parameters. The communication graph defines which nodes exchange state information when events occur. The asynchronous dynamics arise naturally as task arrivals, local workloads, and queue lengths evolve over time.

One dimension of experimentation concerns the effect of event thresholds on system behavior [46]. When thresholds σ_i are chosen to be small, agents update frequently in response to minor changes in their local assignments or objective values. This yields a high rate of control messages and local computations, which can increase energy consumption due to control overhead. However, the swarm tracks changes in workload and system state closely, which can improve the attained energy–latency tradeoff. When thresholds are large, the opposite occurs: control overhead is reduced, but

the swarm may respond sluggishly to changes, and the system may operate at points that are further from those attainable with more active control [47].

Simulation runs can record metrics such as total energy consumption, average task latency, maximum task latency, and the number of control messages exchanged over a given time window. By varying the weight parameter λ, one can generate empirical tradeoff curves that illustrate how the swarm behaves under different relative valuations of energy and latency. For low λ, latency is emphasized, and the swarm tends to allocate tasks to nodes that provide lower service times even if their energy efficiency is lower. For high λ, energy is prioritized, and tasks may be shifted toward nodes that offer better energy profiles, possibly at the expense of increased latency.

Another dimension concerns the influence of communication topology [48]. When the swarm communication graph is dense, with many edges between agents, information about good assignments spreads quickly. This can facilitate convergence to configurations with lower objective values but introduces higher communication overhead during event-triggered broadcasts. When the communication graph is sparse, updates propagate more slowly, and different regions of the swarm may explore the solution space somewhat independently. This can be beneficial in avoiding premature convergence to local minima but may also prolong the time needed for the swarm to assemble a globally coordinated assignment [49].

The heterogeneity of edge nodes is reflected in both performance and roles within the swarm. Nodes with higher processing capacity and better energy efficiency may gradually become preferred destinations for tasks in energy-focused scenarios. Their local agents may also accumulate better local best assignments and therefore exert stronger influence on neighbors through the neighborhood best vectors. Conversely, low-capacity nodes may tend to offload tasks more frequently and can act as relay agents that propagate information between different parts of the communication graph.

Asynchronous operation is manifested in the irregular timing of updates across agents [50]. Some nodes, especially those experiencing heavy workloads or rapidly changing conditions, may trigger events frequently, while others remain inactive for extended periods. The distribution of update intervals can be characterized statistically and related to factors such as workload variance, threshold settings, and network conditions. In many cases, the asynchronous and event-driven structure allows nodes with stable conditions to remain quiet, saving energy and bandwidth, while nodes in dynamic situations adapt their behavior more actively.

Experiments can also examine robustness to delayed and lost messages. Event-driven communication relies on delivery of state updates to neighbors, but wireless links may introduce variable delays or packet losses [51]. Simulated scenarios with random delays and drop probabilities allow exploration of how such imperfections affect swarm performance. In general, moderate levels of delay and loss do not completely disrupt the swarm, since agents continue to operate based on local information and occasionally updated neighbor states. However, high levels of unreliability can degrade performance as agents act on outdated or incomplete information, highlighting potential benefits of incorporating redundancy, acknowledgment mechanisms, or more robust neighbor selection rules.

An additional aspect concerns the energy cost of control operations themselves [52]. In edge devices with limited energy budgets, the energy used for optimization computations and communications may become nonnegligible relative to application-related energy. The linear cost model used for the optimization can be extended to include control energy, with coefficients representing the energy cost per update and per control message. Experimental evaluation can then assess net gains by comparing total energy including both application and control components under different parameter settings. This can reveal parameter regimes where increased control activity yields diminishing returns or where control overhead outweighs the benefits of improved assignment decisions.

The analysis of experimental results typically involves comparing the event-driven swarm approach to baseline strategies [53]. One baseline is a purely local policy in which each node decides independently whether to accept tasks based solely on its own capacity and energy profile, without coordination. Another baseline is a centralized optimization that assumes perfect knowledge of all system parameters and solves the linear program using a central solver. The centralized strategy can produce a lower bound on achievable objective values, but may be impractical in large or dynamic systems. A periodic distributed

control approach with synchronous rounds might also serve as a comparison, highlighting advantages and disadvantages of event-driven as opposed to time-driven coordination [54].

Across a range of simulated conditions, event-driven swarm optimization can be expected to exhibit intermediate performance between uncoordinated local heuristics and fully centralized optimization. It may offer energy–latency tradeoff curves that are close to those of centralized methods in some regimes while significantly reducing coordination overhead. In other regimes, especially those with rapidly changing conditions or highly constrained communication, the gap to centralized performance may widen. Understanding these patterns is important for assessing the suitability of the method for specific use cases and for guiding parameter selection.

6 Conclusion

This study has examined an event-driven swarm optimization framework for managing energy–latency tradeoffs in asynchronous edge computing systems [55]. The system model incorporates heterogeneous edge nodes, divisible tasks with data and computation requirements, and linear approximations of energy and latency costs. By formulating the resource allocation problem as a linear program with a scalarized objective, the analysis provides a tractable basis for investigating how distributed, swarm-inspired algorithms can approximate centralized optimization outcomes under realistic constraints.

The proposed algorithm associates each edge node with an agent that maintains local candidate assignments and updates them according to swarm rules adapted to an event-driven and asynchronous setting. Event thresholds govern when local state changes trigger computational updates or communication with neighbors, which allows the algorithm to reduce overhead by avoiding unnecessary actions when the system state remains relatively stable. The swarm structure facilitates distributed exploration of the assignment space, while the linear model supports efficient evaluation of candidate solutions and integration of capacity, energy budget, and latency constraints [56].

From a modeling perspective, the use of linear energy and latency expressions enables decomposition of the global objective into local contributions, thereby aligning well with the agent-based architecture. The analysis of parameterized objectives as a function of an energy–latency weight illustrates how different operating points along the tradeoff frontier can be targeted. Theoretical considerations of linearized swarm dynamics provide qualitative insight into stability and convergence under asynchrony, even though rigorous guarantees in fully realistic settings remain challenging due to stochastic elements, event-driven timing, and network imperfections.

Simulation-based discussion indicates that event thresholds, swarm parameters, communication topology, and workload heterogeneity jointly influence the emergent performance of the system [57]. Lower thresholds and denser communication tend to improve tracking of favorable assignments but incur higher control costs, while higher thresholds and sparser connectivity reduce overhead at the risk of operating further from the optimal energy–latency frontier. Asynchronous operation allows nodes to adapt to local conditions without global synchronization, which can be beneficial in large-scale and dynamic deployments, but it also introduces additional complexity in analyzing convergence and in tuning parameters.

The presented framework can serve as a basis for further refinement and extension. Potential directions include incorporating more detailed energy models that account for dynamic voltage and frequency scaling, idle and sleep states, and hardware-specific behaviors; extending latency models to capture nontrivial queueing dynamics more accurately through piecewise linear or nonlinear approximations; and exploring hybrid schemes that combine event-driven swarm mechanisms with elements of primal–dual or consensus-based optimization. Another avenue is the integration of learning mechanisms that allow agents to adapt their thresholds and swarm parameters over time based on observed performance [58].

In practical deployments, additional issues such as security, privacy, and fairness across tasks and users must also be considered. Swarm-based algorithms may need to be augmented with mechanisms that prevent malicious or faulty agents from unduly influencing global decisions, and that respect constraints on data movement imposed by privacy regulations or user preferences. Fairness considerations could be incorporated by augmenting the objective with terms that penalize disproportionate allocation of resources to particular tasks or regions. the event-driven swarm optimization approach

examined here offers a structured way to coordinate asynchronous edge nodes while explicitly considering energy–latency tradeoffs within a linear modeling framework. While not a universal solution, it illustrates how ideas from swarm intelligence and event-triggered control can be combined to address resource management challenges in emerging edge computing environments, and it opens opportunities for further study of algorithmic variants and system-level design choices [59].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgement

This work was supported without any funding.

References

[1] J. Yan, J.-J. Liao, and C.-H. Shih, "Multi-agent hybrid mechanism for financial risk management," *Journal of Industrial Engineering and Management*, vol. 8, no. 2, pp. 435–452, Mar. 25, 2015. DOI: 10.3926/jiem.1313

[2] X. Lin, Y. Zheng, and L. Wang, "Consensus of switched multi-agent systems with random networks," *International Journal of Control*, vol. 90, no. 5, pp. 1113–1122, Jul. 15, 2016. DOI: 10.1080/00207179.2016.1201865

[3] S. Lee, "Consensus of linear multi-agent systems with an arbitrary network delay," *Journal of IKEEE*, vol. 18, no. 4, pp. 517–522, Dec. 31, 2014. DOI: 10.7471/ikeee.2014.18.4.517

[4] R. Chandrasekar and S. Misra, "Introducing an aco based paradigm for detecting wildfires using wireless sensor networks," in *2006 International Symposium on Ad Hoc and Ubiquitous Computing*, IEEE, 2006, pp. 112–117.

[5] O. A. Antamoshkin, O. A. Antamoshkina, and N. A. Smirnov, "Multi-agent automation system for monitoring, forecasting and managing emergency situations*," *IOP Conference Series: Materials Science and Engineering*, vol. 122, no. 1, pp. 012 003–, Apr. 22, 2016. DOI: 10.1088/1757-899x/122/1/012003

[6] J. Tožička, M. Štolba, and A. Komenda, "The limits of strong privacy preserving multi-agent planning," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, pp. 297–305, Jun. 5, 2017. DOI: 10.1609/icaps.v27i1.13828

[7] M. Dastani, *Agent-Oriented Software Engineering - A Survey of Multi-agent Programming Languages and Frameworks*. Springer Berlin Heidelberg, Mar. 17, 2014. DOI: 10.1007/978-3-642-54432-3_11

[8] G. Y. Sato, H. J. Azevedo, and J.-P. A. Barthès, "Agent and multi-agent applications to support distributed communities of practice: A short review," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 1, pp. 87–129, Apr. 5, 2011. DOI: 10.1007/s10458-011-9170-9

[9] C. Zhang and V. Lesser, "Multi-agent learning with policy prediction," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, pp. 927–934, Jul. 4, 2010. DOI: 10.1609/aaai.v24i1.7639

[10] A. Drogoul, D. Vanbergue, and T. Meurisse, "Mabs - multi-agent based simulation: Where are the agents?" In Germany: Springer Berlin Heidelberg, Apr. 15, 2003, pp. 1–15. DOI: 10.1007/3-540-36483-8_1

[11] W. L. Yeung, "Formal verification of negotiation protocols for multi-agent manufacturing systems," *International Journal of Production Research*, vol. 49, no. 12, pp. 3669–3690, Jun. 15, 2011. DOI: 10.1080/00207543.2010.492407

[12] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An ant odor analysis approach to the ant colony optimization algorithm for data-aggregation in wireless sensor networks," in *2006 International Conference on Wireless Communications, Networking and Mobile Computing*, IEEE, 2006, pp. 1–4.

[13] I. Shioya, "An accelerated resource utilization in autonomous stochastic mobile multi-agents," *International Journal of Innovation in the Digital Economy*, vol. 5, no. 1, pp. 1–14, Jan. 1, 2014. DOI: 10.4018/ijide.2014010101

[14] J. A. Torres, D. Sahabandu, R. Dhal, and S. Roy, "Iccps - local open- and closed-loop manipulation of multi-agent networks," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ACM, Apr. 14, 2015, pp. 21–30. DOI: 10.1145/2735960.2735982

[15] C. Nowzari, J. E. Cortes, and G. J. Pappas, *Cooperative control of multi-agent systems - event-triggered communication and control for multi-agent average consensus*, Mar. 25, 2017. DOI: 10.1002/9781119266235.ch7

[16] A. Botea and P. Surynek, "Multi-agent path finding on strongly biconnected digraphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, Feb. 18, 2015. DOI: 10.1609/aaai.v29i1.9430

[17] Q. Gao and Y. I. Cho, "A multi-agent improved semantic similarity matching algorithm based on ontology tree," *Journal of Institute of Control, Robotics and Systems*, vol. 18, no. 11, pp. 1027–1033, Nov. 1, 2012. DOI: 10.5302/j.icros.2012.18.11.1027

[18] C. Zeng, T. Gu, L. Chang, and L. Fengying, "A novel multi-agent evolutionary algorithm for assembly sequence planning," *Journal of Software*, vol. 8, no. 6, pp. 1518–1525, Jun. 1, 2013. DOI: 10.4304/jsw.8.6.1518-1525

[19] M. Štula, D. Stipaničev, and J. Maras, "Distributed computation multi-agent system," *New Generation Computing*, vol. 31, no. 3, pp. 187–209, Aug. 3, 2013. DOI: 10.1007/s00354-012-303-8

[20] D. I. Tapia, S. Rodríguez, and J. M. Corchado, *Pervasive Computing, Innovations in Intelligent Multimedia and Applications - A Distributed Ambient Intelligence Based Multi-Agent System for Alzheimer Health Care*. Springer London, Aug. 11, 2009. DOI: 10.1007/978-1-84882-599-4_9

[21] T. Srinivasan, V. Vijaykumar, and R. Chandrasekar, "An auction based task allocation scheme for power-aware intrusion detection in wireless ad-hoc networks," in *2006 IFIP International Conference on Wireless and Optical Communications Networks*, IEEE, 2006, 5–pp.

[22] J. Hu and J. Cao, "Consensus seeking for multi-agent systems by hybrid output feedback protocol:" *Transactions of the Institute of Measurement and Control*, vol. 39, no. 7, pp. 0 142 331 216 630 365–1113, Feb. 22, 2016. DOI: 10.1177/0142331216630365

[23] D. Roy, D. Anciaux, T. Monteiro, and L. Ouzizi, "Multi-agent architecture for supply chain management," *Journal of Manufacturing Technology Management*, vol. 15, no. 8, pp. 745–755, Dec. 1, 2004. DOI: 10.1108/17410380410565339

[24] D. Kim, J. Choi, and C. Woo, "Development of intelligent multi-agent in the game environment," *Journal of Internet Computing and Services*, vol. 16, no. 6, pp. 69–78, Dec. 31, 2015. DOI: 10.7472/jksii.2015.16.6.69

[25] S. Schlechtweg, T. Germer, and T. Strothotte, "Renderbots—multi-agent systems for direct image generation," *Computer Graphics Forum*, vol. 24, no. 2, pp. 137–148, Jul. 7, 2005. DOI: 10.1111/j.1467-8659.2005.00838.x

[26] G. Yang and V. Kapila, "A dynamic-programming-styled algorithm for time-optimal multi-agent task assignment," in *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, IEEE, Nov. 13, 2002. DOI: 10.1109/.2001.981193

[27] M. Batty and B. Jiang, "Multi-agent simulation: Computational dynamics within gis," in CRC Press, Mar. 30, 2000, pp. 67–84. DOI: 10.1201/9781482268263-9

[28] T. S. Sundresh, "Semantic reliability of multi-agent intelligent systems," *Bell Labs Technical Journal*, vol. 11, no. 3, pp. 225–236, Nov. 8, 2006. DOI: 10.1002/bltj.20191

[29] G. A. Fink and A. D. McKinnon, "Effects of network delays on swarming in a multi-agent security system," in *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, ACM, May 6, 2014, pp. 11–8. DOI: 10.1145/2602945.2602955

[30] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and M. Nivedita, "Exploring the synergism of a multiple auction-based task allocation scheme for power-aware intrusion detection in wireless ad-hoc networks," in *2006 10th IEEE Singapore International Conference on Communication Systems*, IEEE, 2006, pp. 1–5.

[31] B. Li, X. Zhang, J. Wu, and J. Zhu, "Task schedulable problem and maximum scheduling problem in a multi-agent system," *Journal of Software*, vol. 6, no. 11, pp. 2225–2231, Nov. 1, 2011. DOI: 10.4304/jsw.6.11.2225-2231

[32] D. Borri and D. Camarda, "Spatial ontologies in multi-agent environmental planning," in IGI Global, Jan. 18, 2011. DOI: 10.4018/9781609600914.ch015

[33] T. Manev and S. Filiposka, "Semantic aware multi-agent system advantages," *International Journal of Informatics and Communication Technology (IJ-ICT)*, vol. 3, no. 1, pp. 1–12, Apr. 1, 2014. DOI: 10.11591/ijict.v3i1.pp1-12

[34] J. Weijin, Z. Luo, and W. Xing, "Research on multi-agent automated negotiation and dynamic cooperation model," in Germany: Springer Netherlands, Jan. 4, 2012, pp. 1467–1475. DOI: 10.1007/978-94-007-2169-2_174

[35] Z. Ma, Z. Liu, Z. Chen, and M. Sun, "Flocking of distributed multi-agent systems with prediction mechanism," in Germany: Springer Berlin Heidelberg, Dec. 12, 2015, pp. 113–123. DOI: 10.1007/978-3-662-48365-7_12

[36] M. Kim and E. T. Matson, "Paams (workshops) - a cost-optimization model in multi-agent system routing for drone delivery," in Germany: Springer International Publishing, May 28, 2017, pp. 40–51. DOI: 10.1007/978-3-319-60285-1_4

[37] F. Wang and H. Yang, "Containment consensus of multi-agent systems with communication noises," in Germany: Springer Singapore, Sep. 22, 2016, pp. 189–197. DOI: 10.1007/978-981-10-2338-5_19

[38] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 2007, pp. 2701–2706.

[39] P. Werlang et al., "Iberamia - estimation of parameters of mycobacterium tuberculosis growth: A multi-agent-based simulation approach," in Germany: Springer International Publishing, Nov. 12, 2014, pp. 599–610. DOI: 10.1007/978-3-319-12027-0_48

[40] Y. D. Yang and C. Guo, "Research on intelligent monitoring and fault diagnosis system based on multi-agent model," *Applied Mechanics and Materials*, vol. 401-403, pp. 1466–1469, Sep. 3, 2013. DOI: 10.4028/www.scientific.net/amm.401-403.1466

[41] L. Wang, Z. Chen, Z. Liu, and Z. Yuan, "Finite time agreement protocol design of multi-agent systems with communication delays," *Asian Journal of Control*, vol. 11, no. 3, pp. 281–286, May 3, 2009. DOI: 10.1002/asjc.104

[42] A. Marchewka, Z. Lutowski, B. Marciniak, M. Śrutek, and S. Bujnowski, "Control algorithms in multi-agent environment," *ipc*, vol. 17, no. 3, pp. 47–62, Dec. 1, 2012. DOI: 10.2478/v10248-012-0021-3

[43] T. Ahlbrecht, J. Dix, and N. Fiekas, *EUMAS/AT - Scalable Multi-agent Simulation Based on MapReduce.* Germany: Springer International Publishing, Jun. 23, 2017. DOI: 10.1007/978-3-319-59294-7_31

[44] S. De, S. R. Sahoo, and P. Wahi, "Trajectory tracking control with heterogeneous input delay in

multi-agent system," *Journal of Intelligent & Robotic Systems*, vol. 92, no. 3, pp. 521–544, Oct. 3, 2017. DOI: 10.1007/s10846-017-0715-2

[45] Z. Ya-Kun, G. Xinping, and L. Xiaoyuan, "Finite-time consensus of heterogeneous multi-agent systems," *Chinese Physics B*, vol. 22, no. 3, pp. 038 901–, Mar. 14, 2013. DOI: 10.1088/1674-1056/22/3/038901

[46] L. Astefanoaei and F. de Boer, "The refinement of multi-agent systems," in Springer US, Jul. 7, 2010, pp. 35–65. DOI: 10.1007/978-1-4419-6984-2_2

[47] R. Chandrasekar and S. Misra, "Using zonal agent distribution effectively for routing in mobile ad hoc networks," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 3, no. 2, pp. 82–89, 2008.

[48] A. Stranjak, I. Čavrak, and M. Zagar, "Scenario-based modeling of multi-agent systems," in InTech, Apr. 1, 2011, pp. 57–76. DOI: 10.5772/15671

[49] M. Zhao and J. Wu, "Design of a multi-agent system for aero engine health management," *The Open Automation and Control Systems Journal*, vol. 6, no. 1, pp. 1071–1075, Dec. 31, 2014. DOI: 10.2174/1874444301406011071

[50] T. Ma, F. L. Lewis, and Y. Song, "Exponential synchronization of nonlinear multi-agent systems with time delays and impulsive disturbances," *International Journal of Robust and Nonlinear Control*, vol. 26, no. 8, pp. 1615–1631, Jun. 24, 2015. DOI: 10.1002/rnc.3370

[51] R. Pinheiro and E. Furtado, "An adaptive multi-agent environment," in IGI Global, Jan. 18, 2011. DOI: 10.4018/9781591405566.ch001

[52] M. Tousi, S. H. Hosseinian, and M. B. Menhaj, "A multi-agent-based voltage control in power systems using distributed reinforcement learning," *SIMULATION*, vol. 87, no. 7, pp. 581–599, May 19, 2010. DOI: 10.1177/0037549710367904

[53] F. Aityacine, B. Hssina, and B. Bouikhalene, "Jade multi-agent middleware applied to contribute to certificate management of students," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 16, no. 1, pp. 176–176, Oct. 1, 2015. DOI: 10.11591/tijee.v16i1.1601

[54] G. Miao, Z. Wang, Q. Ma, and J. Lu, "Consensus of second-order multi-agent systems with nonlinear dynamics and time delays," *Neural Computing and Applications*, vol. 23, no. 3, pp. 761–767, Jun. 21, 2012. DOI: 10.1007/s00521-012-0991-5

[55] S. Su, *Distributed coordination control of complex multi-agent networks with dynamic interaction topologies*, Dec. 11, 2017. DOI: 10.18130/v3pz51k8z

[56] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, 2006, pp. 1–6. DOI: 10.1109/ICCIS.2006.252326

[57] Y. Wang and Y. Wang, "A collaborative work system of urban management based on multi-agent," in Germany: Springer Berlin Heidelberg, Mar. 8, 2017, vol. 13, pp. 194–203. DOI: 10.1007/978-3-662-54395-5_17

[58] D. A. Vidhate and P. Kulkarni, "Multi-agent cooperation models by reinforcement learning (mcmrl)," *International Journal of Computer Applications*, vol. 176, no. 1, pp. 25–29, Oct. 17, 2017. DOI: 10.5120/ijca2017915511

[59] D. Boskos and D. V. Dimarogonas, *Robust connectivity analysis for multi-agent systems*, Jan. 1, 2015. DOI: 10.48550/arxiv.1503.07143