ENGINEERING Information Technology & Electronic Engineering
https://www.academax.com/EITEE; https://www.engineering.org.cn
https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=11344065
E-mail: eng.itee@zju.edu.cn

Supplementary materials for

Runnan QIN, Zhen YANG, Xiaodong PENG, Wenming XIE, Xiao ZHENG, Jingyi REN, Zeyu FAN, 2026. A microservice framework for data–model fusion in situational awareness simulations. *ENGINEERING Inform Technol Electron Eng*, 27(7):250154. https://doi.org/10.1631/ENG.ITEE.2025.0154

1 Distributed data–model fusion management and scheduling strategy

Additional flowchart descriptions of the algorithm pseudocode for the distributed data–model fusion management and scheduling strategy are provided in Figs. S1–S3.

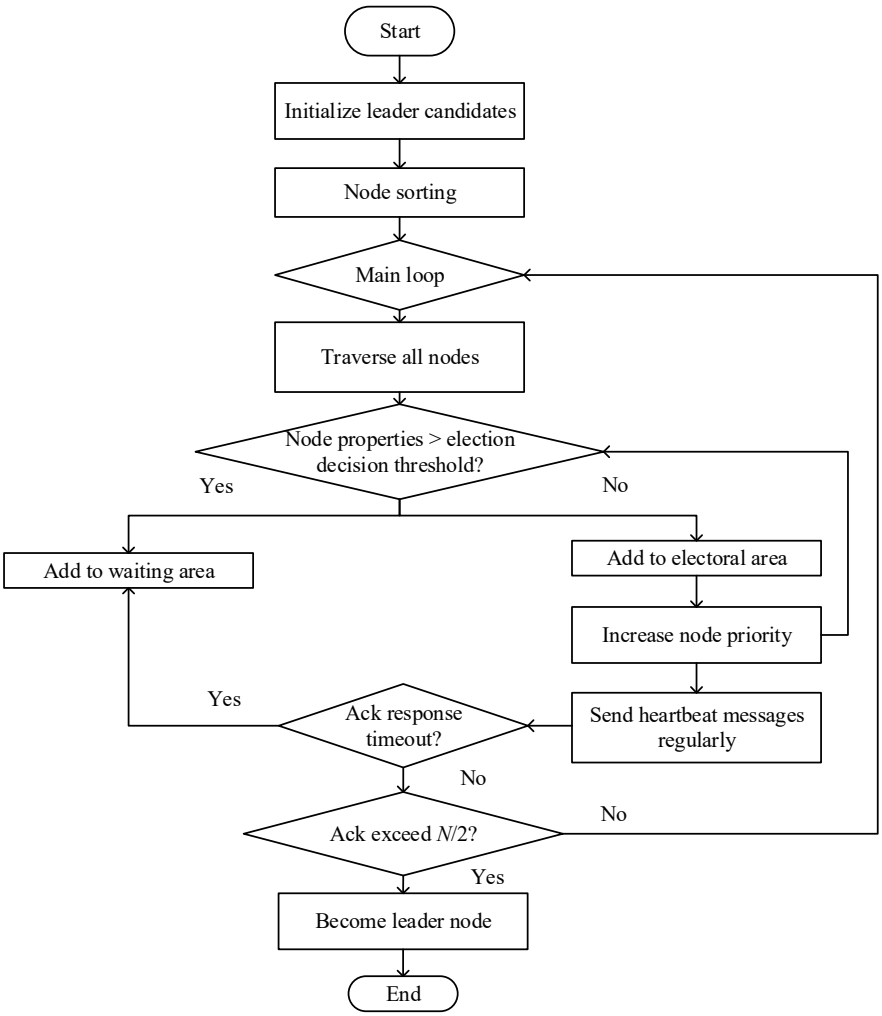

Fig. S1 Flow diagram of Algorithm 1

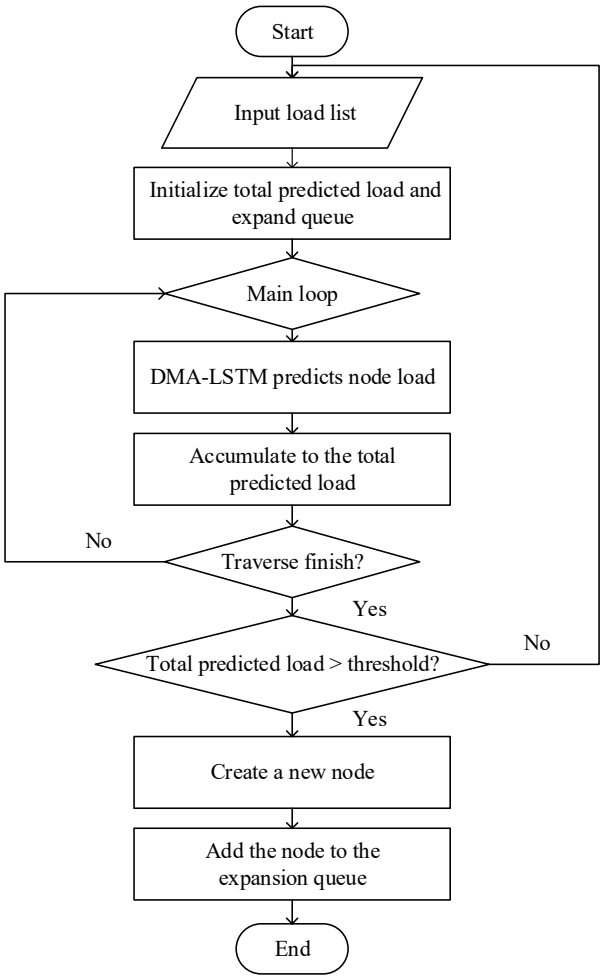

Fig. S2 Flow diagram of Algorithm 2

2 Experimental analysis

Detailed descriptions of the experimental settings, including the experimental environment configuration and the configuration parameters of our algorithm and baseline algorithms in comparative experiments, beyond those presented in the main text, are provided here.

2.1 Experimental setup

As depicted in Fig. S4, the MP-DMC framework employs a distributed container cloud architecture with centralized orchestration management.

Fig. S5 presents the MP-DMC management console featuring eight functional modules for cluster orchestration. It includes functionalities for managing computing nodes, computational tasks, computational workflows, operational reports, task scheduling, scheduling logs, executor management, and user management.

2.2 Dynamic election analysis

To ensure a fair and reproducible comparison, the key operational parameters for each algorithm were configured as summarized in Table S1. These parameters (e.g., timeouts and intervals) directly influence the election behavior and were set according to their typical values for the five-node network environment.

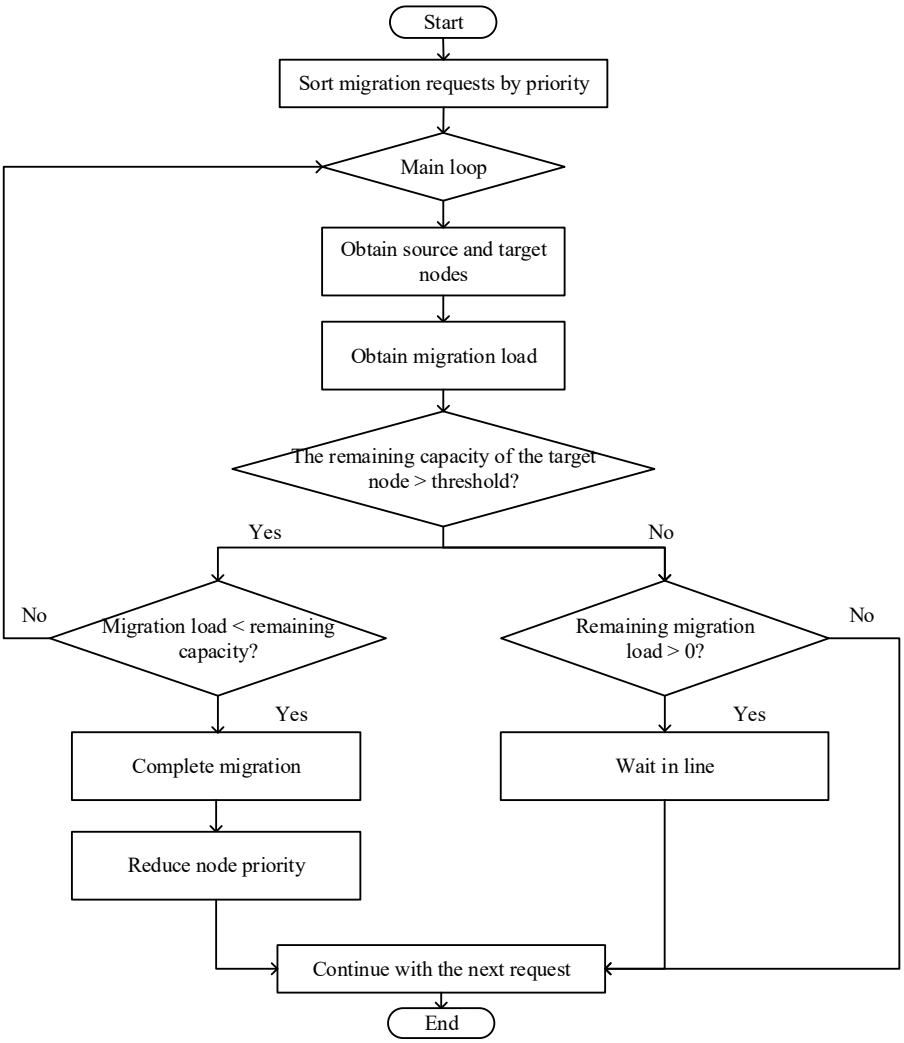

Fig. S3 Flow diagram of Algorithm 3

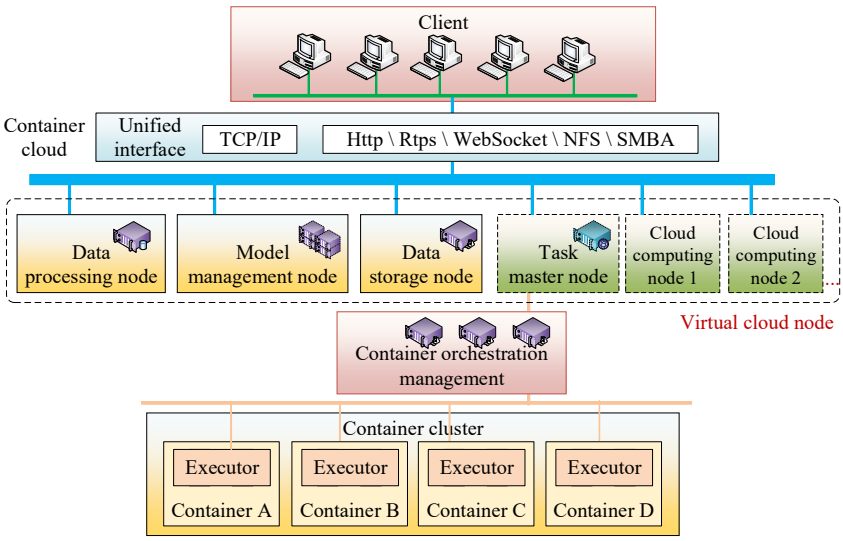

Fig. S4 Hardware environment deployment diagram

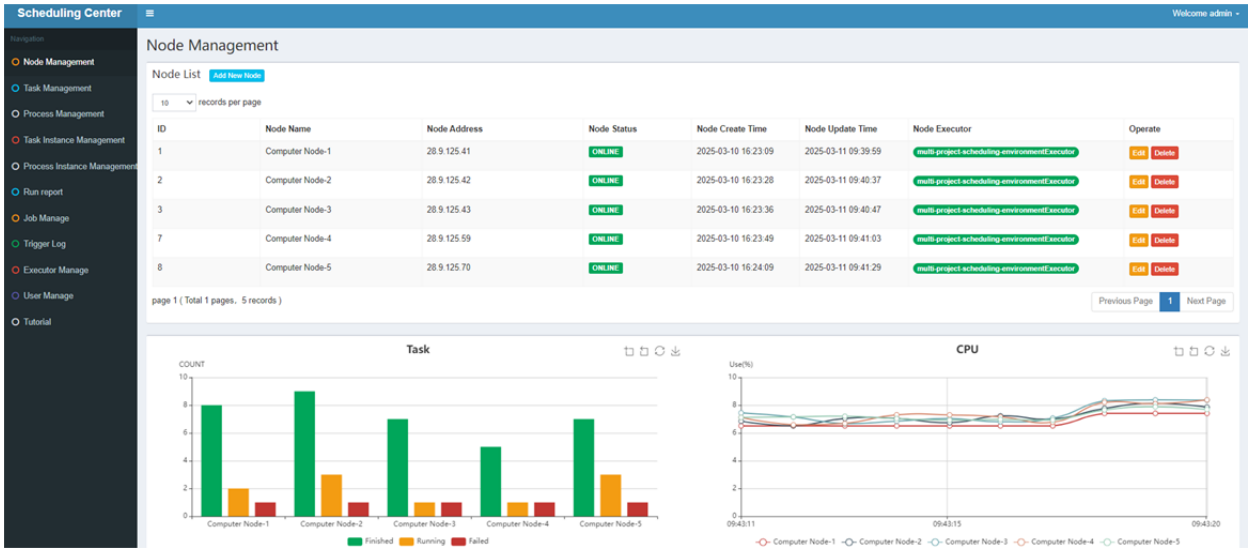

Fig. S5 Client management interface of MP-DMC. It can visually display the task processing status on each computing node

Table S1 Configuration of election algorithm parameters for the five-node experiment

Algorithm	Key parameter	Value
Paxos	Proposer timeout	200 ms
Raft	Election timeout	1500 ms
	Heartbeat interval	100 ms
PBFT	View-change timeout	1500 ms
	Quorum size	$3f+1$ (f=1 for 5 nodes)
MP-DMC	Heartbeat interval	100 ms

2.3 Elastic expansion performance

2.3.1 DMA-LSTM prediction model analysis

To analyze the performance of the node dynamic elastic scaling algorithm, it is first necessary to validate the efficacy of the internal DMA-LSTM prediction model. The DMA-LSTM model was a dual-branch hybrid where the lower branch captures smooth baseline trends via DMA, and the upper LSTM branch exclusively corrects short-term prediction residuals. It was trained using the Adam optimizer with a learning rate of 0.001, a batch size of 64, and over 100 epochs on a dataset comprising continuous high-frequency load traces from 10 server nodes. Each node's load (CPU, memory, I/O) was sampled every 20 s over a five-month operational period, yielding a total of approximately 650 000 time steps per metric for model training and evaluation. The dataset was split into training, validation, and test sets with a ratio of 7:2:1, and all features were normalized to the [0, 1] interval. For completeness, the detailed hyperparameter configurations of all baseline models compared in this study are summarized in Table S2.

2.3.2 Dynamic elastic expansion analysis

To ensure a fair and reproducible comparison, we configured the baseline algorithms as follows. For all HPA-based baselines (HPA, DMA-HPA, and ProSmart HPA), common Kubernetes scaling parameters were set uniformly: the target CPU utilization percentage was 75%; the stabilization window was 300 s for scaling down and 60 s for scaling up, to prevent thrashing; the minimum and maximum numbers of pod replicas were

set to 1 and 10, respectively. Beyond these common settings, each algorithm was configured with its own unique parameters, as detailed in Table S3.

Table S2 Hyperparameter configurations for baseline prediction models

Model	Key parameter	Value
Historical mean	Sliding window size	20 timesteps
SVM	Kernel	RBF
	Regularization coefficient	1.0
	γ	0.01
	Epsilon	0.1
LSTM	Layer	1 LSTM layer with 128 units
	Optimizer	Adam (lr=0.001)
	Batch size	64
	Number of epochs	100
	Input sequence length	20 timesteps
DMA	Short-term MA window	5 time steps
	Long-term MA window	20 time steps

Table S3 Algorithm key parameter configuration

Algorithm	Key parameter	Value
HPA	Target CPU utilization percentage	75%
	Scale down stabilization window	300 s
	Scale up stabilization window	60 s
	Number of pod replicas	1–10
DMA-HPA	Target CPU utilization percentage	75%
	Rolling history window	10 min
	Prediction horizon	90 s
ProSmart HPA	Target CPU utilization percentage	75%
	Threshold step (Δ)	5%
	Adaptation trigger	3

The RL-based baseline was implemented using a tabular Q-learning approach, following the threshold adjustment policy proposed by Rossi et al. (2020). Unlike deep Q-networks (DQNs) which use neural networks for function approximation, this implementation employs a discretized state space mapped to a Q-table. This design choice prioritizes low computational overhead and deterministic convergence in the operational environment. The core design of the Q-learning agent is as follows:

State space: The state is defined as

$$s = \left(\theta,\ u\right) \tag{S1}$$

where θ is the current CPU utilization target, and u is the average CPU utilization of the target microservice over a sliding window.

To facilitate tabular updates, the continuous state variables θ and u are discretized into a finite grid (bin width=0.05, resulting in 400 discrete states), each corresponding to a row in the Q-table.

Action space: The agent can choose from three actions

$$A = \left\{-\delta,\ 0,\ +\delta\right\} \tag{S2}$$

corresponding to reducing the threshold by a quantum δ, maintaining it, or increasing it.

Reward function: We adopt a cost function (negative reward) that balances performance and resource efficiency, as given by Eq. (1) in Rossi et al. (2020):

$$c\left(s,a,s'\right)=w_{\text{perf}}\cdot c_{\text{perf}}\left(s'\right)+w_{\text{res}}\cdot c_{\text{res}}\left(s,a\right) \tag{S3}$$

where c_{perf} penalizes response time violations, c_{res} penalizes resource over-provisioning, and w_{perf} and w_{res} are configurable weights.

Learning parameters: We employ the standard Q-learning update rule by Eq. (2) in Rossi et al. (2020). The key parameters are set in Table S4.

Table S4 Key learning parameters of the tabular Q-learning agent

Parameter	Value
Learning rate, α	0.1
Discount factor, γ	0.99
Exploration rate, ε	0.1
Threshold quantum, δ	0.05
Valid threshold range, [$\Theta_{\min}$, $\Theta_{\max}$]	[0.5, 0.9]
Cost function weight	0.5

2.4 Migration scheduling effectiveness

We conducted a comprehensive comparative analysis of scheduling algorithms across three critical situational awareness simulation scenarios: spacecraft close-in, spacecraft detection, and space environment modeling. We benchmarked the load migration algorithm against three classic scheduling strategies: round-robin, purely random, and weighted random.

The detailed configuration of these baseline schedulers is provided in Table S5, ensuring the reproducibility of the comparison. The performance was evaluated on two metrics: task response time and node load-balancing time. The latter metric is operationally defined as the mean maximum duration required for RLP stabilization across computing nodes during scheduling transitions.

Table S5 Configuration of baseline scheduling algorithms

Algorithm	Key parameter	Value
Round-robin	Fixed time quantum	10 ms
	Queue order	FIFO queue
Purely random	Probability distribution	Uniform random
Weighted random	Weight assignment	Proportional to RLP
MP-DMC	Migration threshold	85% of RLP
	Migration decision	10 ms

Reference

Rossi F, Cardellini V, Presti FL, 2020. Self-adaptive threshold-based policy for microservices elasticity. 28th Int Symp on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, p.1-8. https://doi.org/10.1109/mascots50786.2020.9285951