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

Research Article

https://doi.org/10.1631/ENG.ITEE.2025.0154

A microservice framework for data–model fusion in situational awareness simulations

Runnan QIN[1,2]✉, Zhen YANG[1], Xiaodong PENG[1], Wenming XIE[1,2], Xiao ZHENG[1,2], Jingyi REN[1], Zeyu FAN[1]

[1]*Key Laboratory of Electronics and Information Technology for Space Systems, National Space Science Center, Chinese Academy of Sciences, Beijing 100191, China*

[2]*University of Chinese Academy of Sciences, Beijing 101408, China*

Abstract: In response to the increasing demand for heterogeneous data interaction and cross-disciplinary modeling in aerospace situational awareness simulations, we propose a microservice platform for data–model computing (MP-DMC), which is a high-performance microservice framework built on a container cloud. The proposed MP-DMC framework unifies the control of data and models through a distributed node resource management and scheduling strategy, integrating an election-optimized leader–follower mechanism, a predictive model based on a double-moving-average long short-term memory (DMA-LSTM) network for dynamic elastic scaling, and an intelligent load migration algorithm to address management inefficiencies, prevent node crashes, and mitigate resource oscillations under high-concurrency conditions. Experimental results demonstrate that the proposed MP-DMC framework outperforms mainstream algorithms in terms of election performance, node scaling efficiency, task response time, and load balancing, including consensus algorithms (Paxos, Raft, and PBFT), elastic scaling methods (HPA, DMA-HPA, ProSmart HPA, and RL), and scheduling algorithms (round-robin, purely random, and weighted random), achieving exceptional resource allocation performance and system availability.

Key words: Data–model fusion; High-performance simulation computing; Scheduling strategy; Node load prediction; Microservice

1 Introduction

Aerospace technologies involve large-scale systems, complex processes, and multidisciplinary integration, leading to prohibitively high experimental costs. In this context, simulation technology has become an indispensable solution, offering precise modeling of space entity deployment, operational dynamics, and trends. It provides critical visual analytics for aerospace system design, mission planning, performance validation, and operational optimization, thereby establishing itself as a cornerstone of modern aerospace engineering. The advantages of simulation technology are multifaceted: significant cost reduction, experimental repeatability, enhanced operational reliability, and the ability to overcome spatiotemporal limitations (Ding WH et al., 2025).

In the aerospace field, situational awareness simulation further intensifies these demands, requiring systematic modeling of space entities, heterogeneous experimental scenarios, and intricate environmental variables. This necessitates advanced data–model fusion capabilities to address various challenges, including multisource heterogeneous data integration and multidisciplinary coupled modeling, which impose stringent requirements on simulation accuracy and real-time responsiveness. However, achieving this fusion presents critical technical hurdles:

1. Management instability. Frequent leader elections in distributed nodes cause management inefficiencies and service interruptions.

2. Burst load handling. The inability to accurately predict and proactively scale resources for large-scale, bursty computing tasks leads to node crashes and performance degradation.

3. Resource oscillation. Under high concurrency, reactive load migration can trigger unstable resource allocation and load oscillations, thereby compromising system stability.

Compounding these challenges, the rapid expansion of spacecraft and ground station deployments has triggered an

✉ Runnan QIN, qinrunnan@nssc.ac.cn
Runnan QIN, https://orcid.org/0009-0007-7668-2385
CLC number: TP391.9

Received: Nov. 30, 2025; Revision accepted: May 22, 2026;
Crosschecked: June 9, 2026

exponential growth in spacecraft telemetry and space situational awareness data volumes. Concurrently, the computational complexity of situational awareness simulations has increased dramatically, driven by demands for higher-fidelity modeling.

To address these computational imperatives, critical research priorities emerge, focusing on enabling real-time processing of multisource situational data and supporting large-scale concurrent model computations. Current approaches leverage high-throughput data transmission systems, including topic-based architectures, e.g., Kafka (Wei et al., 2025), RabbitMQ (Pathak and Kalaiarasan, 2021), RocketMQ (Ma Y et al., 2017), and ActiveMQ (Yin et al., 2015), alongside content/channel/type-driven publish–subscribe paradigms (Yang et al., 2004; Dayal et al., 2014; Ding TC et al., 2020). Parallel developments in distributed scheduling systems, exemplified by ElasticJob (Liu F and Weissman, 2015), XXL-JOB (Xu XL, 2015), and the Spring Cloud Scheduler (Pivotal Software, Inc., 2018), have evolved from monolithic to microservice architectures to satisfy escalating computational demands (Dragoni et al., 2017; Feng et al., 2020). However, integrating these discrete solutions for coupled data–model fusion simulations reveals a critical limitation; i.e., incompatible resource management strategies across systems lead to node underutilization and architectural redundancy.

To address these challenges, we propose a microservice platform for data–model computing (MP-DMC), which synergizes distributed cloud management with adaptive scheduling strategies. The architecture of MP-DMC has the following features:

1. secure high-throughput data synchronization channels;

2. leader–follower stability with dynamic election;

3. container elastic scaling by predicting node load;

4. load-aware migration mechanism to realize optimal resource scheduling.

Validated through large-scale space operation simulations, the proposed MP-DMC framework demonstrates superior performance in high-throughput data interaction, high-concurrency task processing, and stable service operation while avoiding load-balancing oscillations. The primary contributions of this study are summarized as follows:

1. Lightweight microservice architecture. A container-native framework is proposed for unified data–model fusion computing.

2. Distributed scheduling strategy. An election-optimized, load-predictive algorithm is proposed, addressing:

(1) a leader–follower election algorithm that reduces election frequency by narrowing the candidate node set, addressing management inefficiencies caused by frequent elections among cloud nodes;

(2) a dynamic elastic expansion algorithm based on the double-moving-average long short-term memory (DMA-LSTM) model, which enables dynamic threshold adjustment through node load demand prediction, resolving node crash issues triggered by large-scale, bursty computing tasks;

(3) a load migration scheduling algorithm leveraging a load migration model to mitigate unbalanced resource allocation and load oscillation under high-concurrency scenarios.

2 Related works

2.1 Secure and coherent transmission of large-scale data

The evolution of distributed cloud computing has revolutionized large-scale data processing by enabling efficient resource sharing across server clusters (Wang et al., 2023). Early frameworks, e.g., Hadoop, demonstrated the potential of distributed systems through components, including the Hadoop Distributed File System (HDFS) and MapReduce; however, their dependence on offline batch processing limits applicability in real-time scenarios that require low-latency data streaming.

Publish–subscribe systems have addressed this gap by supporting asynchronous communication with high scalability and multithreaded concurrency (Tsenos and Kalogeraki, 2021; Zhang CY et al., 2021). Recent advancements have increasingly focused on the integration of cloud computing platforms and publish–subscribe systems. For example, Ullah et al. (2013) proposed a publish–subscribe middleware integrated with Hadoop, employing content-based matching algorithms to analyze subscription intent and deliver more accurate content to subscribers. However, synchronization challenges persist due to performance disparities among cloud nodes, leading to data misalignment and loss.

To address these issues, modern approaches integrate lightweight consensus protocols with adaptive architectures. For example, hierarchical federated learning architectures, e.g., HierFedML, demonstrate how edge–cloud collaboration can enhance data coherence while reducing communication overhead, achieving 15% faster response in mobile edge computing environments (Xu ZC et al., 2023). Despite the progress, Byzantine fault tolerance remains a critical challenge in consensus protocols. In addition, although practical Byzantine fault tolerance (PBFT) (Lamport, 1998) improves robustness over Paxos and Raft (Li et al., 2024; Zheng et al., 2025), its limitations in terms of scalability hinder deployment in ultralarge clusters.

2.2 High-concurrency task scheduling for model computations

The shift from monolithic architectures to microservice architectures has transformed task scheduling paradigms, with frameworks like Spring Cloud (Larsson, 2021), Dubbo (Liang, 2017), and Tars (Mesbahi et al., 2019) enabling rapid service decoupling. However, dynamic resource allocation under high concurrency remains a bottleneck (Cui et al., 2021). ElasticJob and XXL-JOB attempt to address this through task sharding and lightweight scheduling; however, their static resource allocation models may suffer from inefficiencies in fluctuating workloads. In addition, the centralized Azkaban architecture, which is suitable for small-scale cluster scheduling, enables rapid configuration of task flows. The distributed Apache DolphinScheduler architecture (Guo, 2019) further simplifies the configuration process within its framework. However, these systems may experience server lag or even single points of failure when handling an excessive number of tasks.

Recent studies have proposed innovative solutions. For

example, Liu WG et al. (2023) developed a space-efficient fair cache scheme for Non-Volatile Memory Express (NVMe) solid-state drives (SSDs), combining machine learning with adaptive load prediction to improve I/O throughput in high-concurrency scenarios. However, due to limited computational power, artificial intelligence-driven scheduling is not widely used in industrial contexts. The focus remains on research associated with load-balancing algorithms.

Static load-balancing algorithms, e.g., ratio-based ones (Luo et al., 2022), frequently fail to account for heterogeneous task complexities. In contrast, dynamic load-balancing algorithms such as weighted round-robin can improve data interaction efficiency and optimize server resource allocation (Shona and Sharma, 2023; Shrimal et al., 2024). However, two primary challenges remain:

1. Resource allocation for task nodes is frequently rigid or overly simplistic, with excessive node backups leading to significant resource wastage.

2. An influx of tasks to lightly loaded nodes may trigger additional load-balancing cycles, causing periodic oscillations that degrade server performance.

2.3 Computational analysis of data–model fusion

Data–model fusion technology serves as a bridge between heterogeneous data streams and computational models, facilitating real-time analytics. In simulation-based applications, the effectiveness of data–model fusion has been well established. For example, integrating key monitoring data from coal mining machines enables real-time rendering (Ma HC et al., 2023). Similarly, the fusion of data-driven and simulation-based methods has been applied to fault diagnosis in plunger pumps (Zhang Y, 2023), and a real-time hybrid simulation platform that combines digital models with physical control and protection devices has been developed (Lei et al., 2024).

However, the increasing demand for situational awareness simulation necessitates the integration of high-throughput data transmission and model scheduling, an area where existing solutions remain inadequate. In this context, two primary challenges dominate:

1. Leader–follower stability. Frequent elections in dynamic clusters degrade availability.

2. Elastic scaling. Rigid scaling threshold settings cause container crashes under sudden loads.

In summary, to the best of our knowledge, very limited efforts have been made to ensure dynamic cloud node management and optimal resource scheduling for microservice-based applications in situational awareness simulation. This study attempts to fill that gap.

3 Design of the microservice architecture for data–model fusion computing

3.1 Overall architecture design

The proposed MP-DMC framework adopts a four-layer distributed architecture comprising the node management center, model calculation scheduling, data security transmission, and data storage management layers, as shown in Fig. 1.

The node management center layer governs all cloud

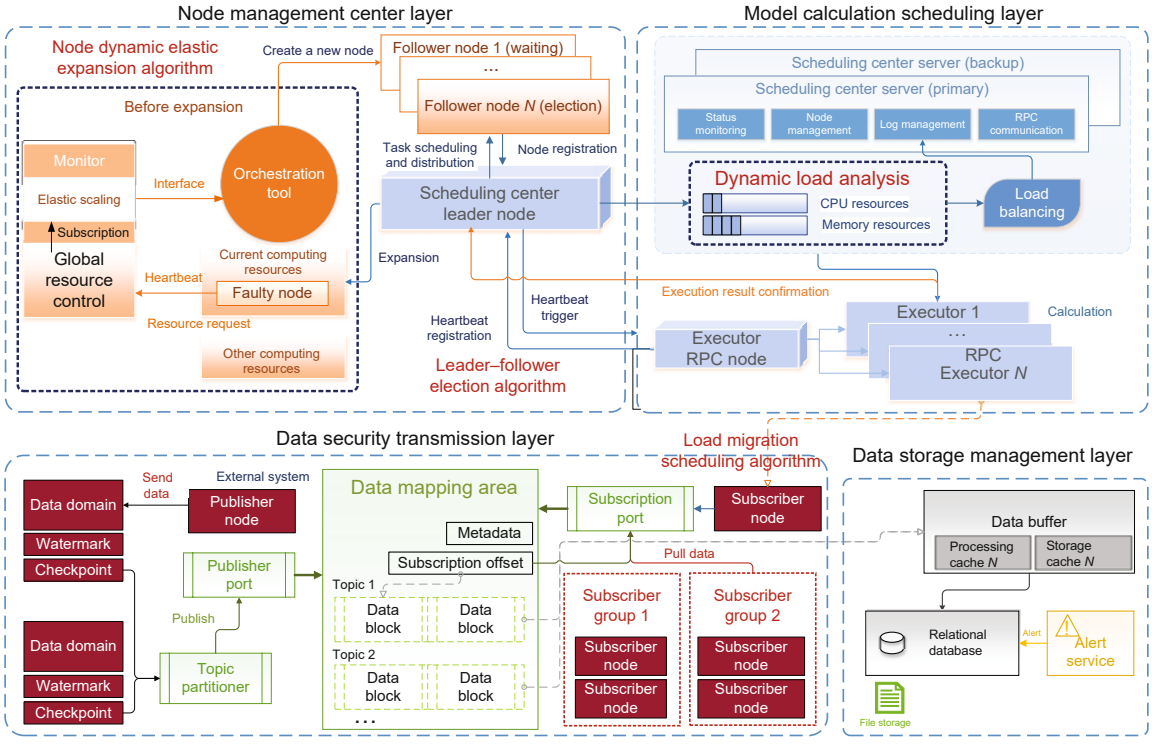


Fig. 1 Overall architecture of the proposed microservice platform for data–model computing (MP-DMC), which adopts a four-layer distributed architecture where similar functional structures are expressed using the same color

nodes through a leader–follower control mechanism. Here, the leader node serves as the scheduling hub, with a dynamic election algorithm that ensures seamless failover by electing a new leader node upon its failure. For follower nodes experiencing faults or insufficient computational resources, a dynamic elastic expansion algorithm triggers resource expansion. The model calculation scheduling and data security transmission layers employ an adaptive load migration scheduling algorithm to adjust node resources dynamically based on real-time workloads. This algorithm ensures ordered data publishing/subscribing for internode synchronization. The data storage management layer performs data buffering, backup, and secure archival storage, achieving high memory throughput and strong I/O performance.

3.2 Node management control design

The node management center layer implements leader–follower control and dynamic elastic expansion, enabling rapid recovery from process interruptions or system crashes. In the event of a leader node failure, follower nodes initiate an election process to select a new leader node.

As shown in Fig. 2, the election process is facilitated by two primary components, i.e., the elector and the communicator. The elector is responsible for the receipt and submission of votes, and the communicator handles communication between nodes during the voting process. Each node may cast its vote for either itself or another node, with the vote being transmitted across the network to the designated node's vote queue, where it is ultimately placed in the vote box. If any follower node receives greater than 50% of the total votes, it assumes the role of the leader node.

When follower nodes experience faults or lack sufficient computational resources, the resource expansion process is triggered, and the required node resource information is transmitted to the global resource control module via a heartbeat mechanism. Simultaneously, a dedicated monitor tracks cluster status through heartbeat subscriptions, initiating vertical scaling when thresholds are breached. Finally, the elastic scaling module stores new node configurations in an expansion queue, subsequently deployed as follower nodes by container orchestration tools.

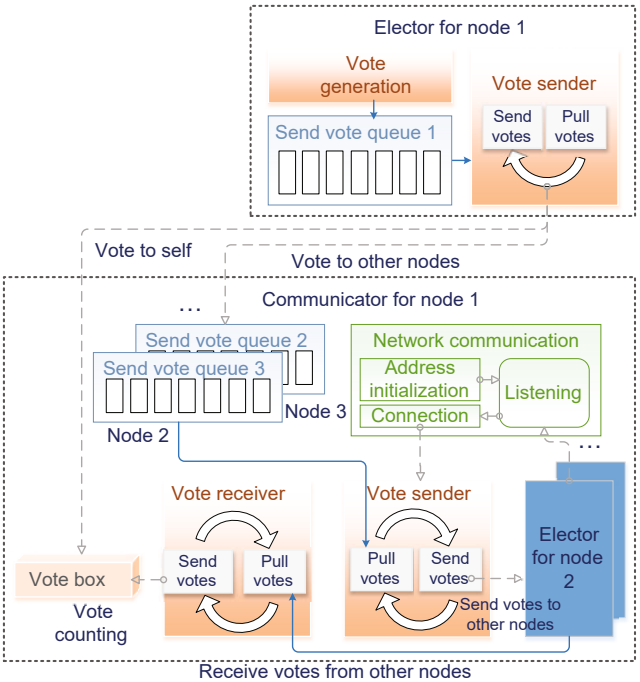


Fig. 2 Node election process

3.3 Model calculation scheduling design

The model calculation scheduling layer utilizes the leader node of the scheduling center as the control hub, rapidly analyzing the users' computational demands and triggering the executor center to allocate tasks. This system schedules tasks dynamically based on the computational resources available on the follower nodes, supporting a fully asynchronous design and distributed deployment.

For time-axis-driven, high-concurrency tasks, the system employs a time wheel storage structure to manage tasks, enabling timer and delayed scheduling functionalities while generating task configuration tables. Here, the leader node of the scheduling center polls the task configurations in the time wheel and distributes tasks accordingly. Executors register with the scheduling center via heartbeat signals, establishing remote communication and standing by for task assignments. Incoming task requests are buffered in a reception queue, and then decomposed according to the current computational resources of each node. Then, the routing strategy triggers the appropriate executor to perform high-concurrency task scheduling. In addition, the system adjusts tasks on overloaded executors dynamically based on real-time load conditions and facilitates the reclamation of expired tasks from offline executors. This mechanism ensures efficient task allocation and execution. The specific scheduling process is illustrated in Fig. 3.

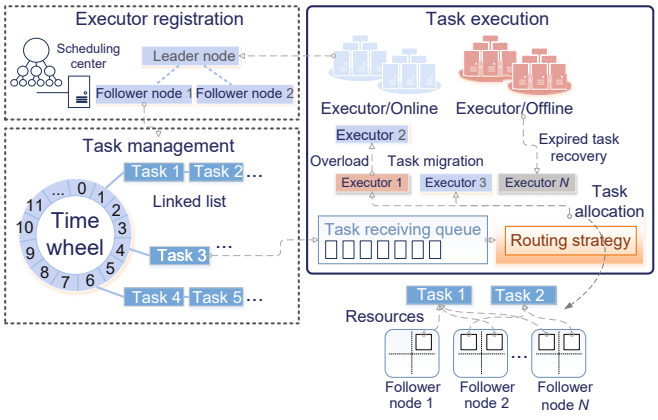


Fig. 3 Computation scheduling process

3.4 Data security transmission design

The data security transmission layer adopts a publish/subscribe architecture that comprises publisher nodes, subscriber nodes, and a data mapping area (Fig. 4).

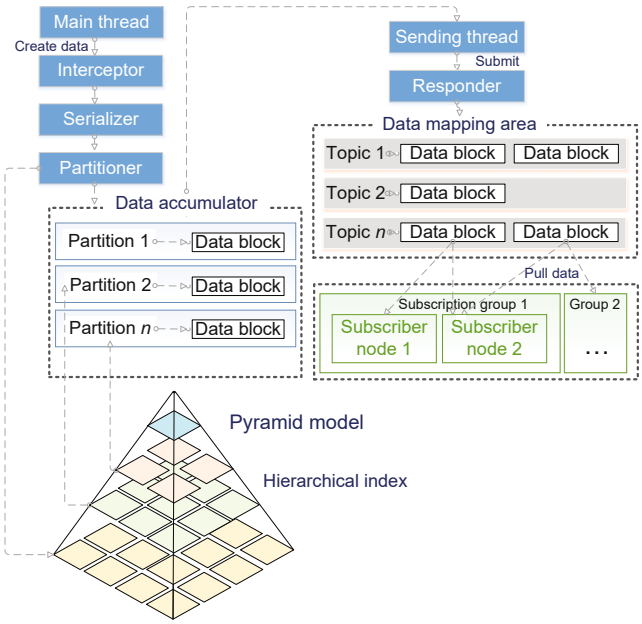


Fig. 4 Data publish/subscribe process

The publisher node operates through the coordination of two threads, i.e., the main and sending threads. In the main thread, user-generated data pass through an interceptor, serializer, and partitioner before being buffered at the end of the data accumulator queue. This process enables the sending thread to batch the transmission of data, reducing network resource consumption and enhancing performance. The partitioner employs a pyramid-style hierarchical management approach, using a layered index to partition data and facilitate topic-based subscriptions. The sending thread retrieves messages from the front of the data accumulator's double-ended queue and transmits them to the data mapping area via a responder.

The subscriber node can join a subscription group and subscribe to data topics in the data mapping area based on a watermark that indicates which data are available to be pulled by the subscriber node. After pulling the data, it saves the subscription offset. The subscription offset represents the position of the next data item to be retrieved. Thus, the data between the watermark and the subscription offset constitute the available content for retrieval.

4 Distributed data–model fusion management and the scheduling strategy

4.1 Leader–follower election algorithm

In distributed node scheduling scenarios, the proposed MP-DMC framework adopts a leader–follower architecture for cluster management. Here, the leader node assumes critical computational and communication responsibilities; however, it becomes prone to resource saturation under high-concurrency workloads, potentially triggering node failures. To ensure continuity, an election algorithm dynamically selects replacement leader nodes from follower candidates. However, frequent topology changes induced by new node integrations may degrade system availability through repeated elections. To mitigate this, we propose a leader–follower election algorithm (Algorithm 1) with the following features:

1. Priority-based node zoning. Nodes are classified into election and waiting zones via dynamically adjusted thresholds derived from cluster size and real-time load probability (RLP). RLP quantifies node fitness through weighted parameters:

$$L_j(t) = (\alpha N_c + \beta C_c + \gamma M_c)/(N_w + C_w + M_w), \quad (1)$$

where N_c and N_w denote network bandwidth (in Gb/s), C_c and C_w denote the CPU computational resources quantified in vCPUs, and M_c and M_w specify the memory size (in GB). The coefficients α, β, and γ indicate the normalized proportional weights of the network, CPU, and memory resources relative to a node's total allocable capacity, respectively. Subscripts w and c denote the server's whole resource pool and the currently consumed resources across all categories, respectively.

2. Dynamic threshold adaptation. Based on the current threshold adaptability, overloaded nodes are demoted to waiting zones directly to delay the triggering of elections.

Algorithm 1 Leader–follower election

Require: N, Q_n $(n = 1, 2, \ldots, N)$, θ, L_{thr}
Initialize A_{election} and A_{wait}
Sort Q_i $(i = 1, 2, \ldots, N)$ by priority or load in ascending order
for $i = 1$ to N **do**
  if $|A_{\text{election}}| > \theta$ **then**
    Add Q_i to A_{wait}
  else if $|A_{\text{wait}}| > \theta$ **then**
    Add Q_i to A_{election}
  end if
end for
Timer $\rightarrow$ Multicast(Q_i, heartbeat message)
if $L_i > L_{\text{thr}}$ **then**
  Add Q_i to A_{wait}
end if
if Q_i has priority **then**
  Multicast(move over message) and wait for acknowledgments
  if Number of acknowledgments $> N/2$ **then**
    Q_i becomes the leader node
  else
    Multicast(end message)
  end if
end if
return Q_n

In Algorithm 1, nodes are initially prioritized, and an election decision threshold θ is established based on the current cluster size. Nodes exceeding this threshold are placed in the election zone A_{election}, and those falling below the threshold are placed in the waiting zone A_{wait}. Only follower nodes within the election zone are eligible to participate in the election for the leader node. After the election, the priority of the participating follower nodes is increased. The threshold is adjusted dynamically according to real-time conditions. Here, if the number of nodes in the election zone becomes excessive, nodes with lower priority are demoted to the waiting zone. In addition, nodes whose real-time load surpasses the threshold L_{thr} are immediately relocated to the waiting zone, which reduces the frequency of elections triggered by nodes exceeding load limits during cluster membership. In the event of a leader node failure due to an unexpected process interruption or crash, an election is initiated in the election zone. A follower

node that receives greater than 50% of the votes assumes the leader node's duties.

Fig. S1 in the supplementary materials provides the flowchart of Algorithm 1.

4.2 Node dynamic elastic expansion algorithm

In the orchestration management and scaling of container clouds, the increasing diversity of user requests presents significant challenges, particularly, when handling large-scale, sudden computational tasks. The inflexibility of predefined scaling thresholds, coupled with inefficiencies in scaling operations, can lead to substantial delays. Once the load surpasses these thresholds, scaling operations cannot be performed instantaneously, which places heavily loaded containers at risk of failure due to overwhelming demand. Autoscaling is an essential approach for addressing such issues because it manages computing resources dynamically among microservices in response to workload fluctuations without human intervention. A prominent solution for executing autoscaling of microservice deployments is the horizontal pod autoscaler (HPA) (Nguyen et al., 2020). However, existing HPAs exhibit certain limitations, including the inability to scale microservice deployment beyond the predefined resource limits, delayed responses to workload fluctuations, and architectural vulnerabilities that are prone to failures. To address these limitations, research efforts have evolved along two main paths, i.e., enhanced rule-based heuristics such as DMA-HPA, Smart HPA, and ProSmart HPA (Ahmad et al., 2025), and data-driven approaches that employ artificial intelligence. Notably, reinforcement learning (RL), particularly Q-learning, has been actively investigated for proactive and adaptive scaling policies by learning optimal decisions directly from interactions with the environment (Rossi et al., 2020). RL methods show promise in handling complex dynamics; however, they frequently face challenges in training stability, sample efficiency, and generalizability across diverse workload patterns. Drawing on the advantages of both paradigms, i.e., the interpretability of model-based prediction and the adaptability of learning, we propose a dynamic elastic expansion model named DMA-LSTM (Algorithm 2). The DMA-LSTM model is designed for large-scale data–model fusion computing tasks, aiming to deliver robust and accurate scaling decisions by decoupling and jointly learning the deterministic trends and stochastic residuals in resource demand.

The DMA-LSTM model adopts a dual-branch hybrid architecture, as shown in Fig. 5. Here, the lower branch applies the DMA principle with a short-term window of five time steps and a long-term window of 20 time steps to extract smooth baseline trends from historical RLP sequences. This branch captures the low-frequency inertia inherent in aerospace simulation workloads. The upper branch comprises a lightweight LSTM network whose sole purpose is to model the prediction residual, i.e., the discrepancy between the DMA trend and the actual observed load. The final forecast is obtained by elementwise addition of the DMA baseline and LSTM residual correction. This architecture mitigates the overfitting tendency of pure LSTM models on noisy data and provides accurate resource demand forecasts for future

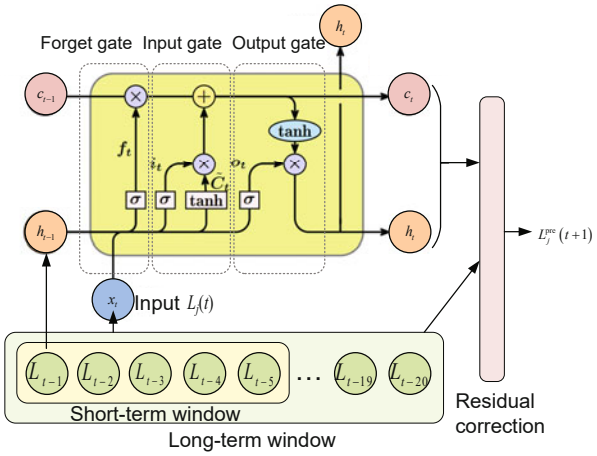


Fig. 5 Structure of the DMA-LSTM model

intervals, thereby enabling effective and dynamic adjustment of the scaling thresholds.

As shown in Fig. 5, the LSTM architecture employs three gates to regulate the information flow through the cell state. At each time step t, LSTM takes the current input x_t and the previous hidden state h_{t-1} as inputs, producing the updated hidden state h_t and cell state C_t.

The forget gate determines which information from the previous cell state C_{t-1} should be retained or discarded. Its output f_t is computed from the concatenation of h_{t-1} and x_t through a sigmoid layer:

$$f_t = \sigma\left(W_{\mathrm{f}} \cdot [h_{t-1}, x_t] + b_{\mathrm{f}}\right), \tag{2}$$

where W_{f} and b_{f} are the weight matrix and bias vector of the forget gate, respectively, and $\sigma(\cdot)$ denotes the sigmoid activation function.

To capture the recent trend of resource demand, we incorporate a one-step moving average $M_t(1)$ of RLP for node j:

$$M_t(1) = \frac{1}{n}\sum_{k=0}^{n-1} L_j(t-k), \tag{3}$$

where n is the window size (number of historical RLP values considered), and $L_j(t)$ is the RLP value of node j at time t, as defined in Eq. (1). This moving average is added to the cell state update to smooth short-term fluctuations.

The input gate controls the addition of new information to the cell state. It consists of two components: i_t, the input gate activation, which decides how much of the new candidate state to write; $\tilde{C}_t$, the candidate cell state, which creates new candidate values from the current input and previous hidden state. These are computed as

$$i_t = \sigma\left(W_{\mathrm{i}} \cdot [h_{t-1}, x_t] + b_{\mathrm{i}}\right), \tag{4}$$

$$\tilde{C}_t = \tanh\left(W_{\mathrm{c}} \cdot [h_{t-1}, x_t] + b_{\mathrm{c}}\right), \tag{5}$$

where W_{i}, W_{c} and b_{i}, b_{c} are the weight matrices and bias vectors for the input gate and candidate cell state, respectively.

The output gate filters the updated cell state to produce the final hidden state. Its activation o_t and the output h_t are

$$o_t = \sigma\left(W_{\mathrm{o}} \cdot [h_{t-1}, x_t] + b_{\mathrm{o}}\right), \tag{6}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{7}$$

$$h_t = o_t * \tanh(C_t), \tag{8}$$

where W_o and b_o are the weight matrix and bias for the output gate, and $*$ denotes element-wise multiplication. The updated cell state C_t combines the retained previous information ($f_t * C_{t-1}$) with the new candidate information ($i_t * \tilde{C}_t$), optionally augmented by the moving average $M_t(1)$ if integrated into C_t.

Finally, the next load demand $L_j^{\text{pre}}(t+1)$ is predicted by inputting the current RLP value $L_j(t)$.

In Algorithm 2, the static scaling threshold a is initially established after the node queue Q_n designed by Algorithm 1, with the dynamic scaling threshold $L_j^{\text{pre}}(t+T)$ determined based on the load demand predicted by the DMA-LSTM model. Typically, $L_j^{\text{pre}}(t+T) < a$. When the cluster load reaches this dynamic threshold, the scaling mechanism is triggered to provision new nodes. By the time the load reaches the static scaling threshold, the scaling process will have been completed, thereby reducing delays. The difference between the dynamic scaling threshold and the actual static threshold is defined as the tolerance zone, which mitigates delays in scaling initiation. Fig. S2 shows the flowchart of Algorithm 2.

Algorithm 2 Dynamic elastic expansion algorithm

Require: $N, Q_n\ (n = 1, 2, \ldots, N)$
Sum $\leftarrow 0$
for $i = 1$ to N **do**
  Predict $L_i^{\text{pre}}(t+T)$ for node i using DMA-LSTM
  Sum $\leftarrow$ Sum $+ L_i^{\text{pre}}(t+T)$
end for
if Sum $> N$ **then**
  Create a new node Q_{new}
  Add Q_{new} to the expanding queue
end if
return $\left(Q_n \cup \{Q_{\text{new}}\},\ N+1\right)$

4.3 Load migration scheduling algorithm

In a distributed microservice architecture, the challenge lies in optimally controlling the node resources. To address this issue, load-balancing strategies are employed to ensure a balanced scheduling of node resources, preventing nodes from becoming overloaded or underutilized.

Most existing load-balancing strategies assume that computational workloads and service response times follow an exponential distribution. However, this assumption is inadequate for data–model fusion tasks, where computational models are invoked on a large scale simultaneously, and frequent data interactions and read–write operations complicate the modeling of service response times.

Thus, we propose a load migration model. Prior to describing this model in detail, we assume that the status information returned by node j comprises a tuple $\langle Q_i, T_j \rangle$ with $i, j \in \{1, 2, \ldots, N\}$, where Q_i denotes the length of the task queue of node j and T_j denotes the number of idle threads in its thread pool. The load-balancing time $\bar{t}_{i,j}$ for each node is proportional to the task queue length and inversely proportional to the number of concurrent threads processing the tasks. This relationship is expressed as follows:

$$\bar{t}_{i,j} = \eta E\left[\Delta L_j(t)\right] \Big/ \langle Q_i, T_j \rangle, \tag{9}$$

$$\langle Q_i, T_j \rangle_{1-\text{busy}_{i,j}} \propto \frac{1}{\bar{t}_{i,j}}, \tag{10}$$

where the subscript $1 - \text{busy}_{i,j}$ denotes the idle capacity ratio of node j when serving task i, i.e., $1 - \rho_{i,j}$, with $\rho_{i,j}$ being the busy ratio (e.g., CPU utilization or thread occupancy) of node j during the execution of task i. This ratio is inversely proportional to the node's load-balancing time. The parameter η is the transmission attenuation coefficient, a constant determined by the server's hardware performance. In addition, $E[\Delta L_j(t)]$ denotes the mathematical expectation of the change in the node's load.

To accurately assess the load status of cloud nodes, we define the task scheduler weight (TSW) as the priority assigned by a node to the current computational task, which is directly proportional to the node's idle capacity. A lower TSW value indicates that the task is neglected.

$$W_{i,j}(t) = h_{i,j}(t-1)\xi_i \left(L_j(t) - L_j(t-1)\right) \Big/ Q_i. \tag{11}$$

Here, ξ_i denotes the task complexity, and $h_{i,j}(t-1)$ is the coefficient indicating whether the node is involved in load migration during the previous sampling period. If no load migration has occurred, this coefficient is set to 1. If the node undergoes load migration, the coefficient is set to 0.5.

The load migration model optimizes the following objectives:

1. Constrain the total load transfer (TLT) oscillations. TLT represents the total amount of load migration between nodes and idle threads, serving as an indicator of the overall system overhead.

$$U(t) = \sum_{i=1}^{N} \sum_{j=1}^{N} u_{i,j}(t), \tag{12}$$

where $u_{i,j}(t)$ represents the load transferred in a single migration.

2. Minimize the node load-balancing time. Combining Eqs. (3) and (5), the node load-balancing time can be expressed as follows:

$$\bar{t}_{i,j} = \frac{\eta U(t)}{Q_i T_j}. \tag{13}$$

Thus, the following loss function is formulated:

$$\text{Loss} = E_{i,j}\left\{\bar{t}_{i,j}(t) + \left[W_{i,j}(t)\left(U(t) - U(t-1)\right)\right]^2\right\} \tag{14}$$

$$= E_{i,j}\Bigg\{\left[W_{i,j}(t)U(t)\right]^2 - 2\left(W_{i,j}(t)\right)^2 U(t)U(t-1) + \left[W_{i,j}(t)U(t-1)\right]^2\Bigg\} + E_{i,j}\left\{\frac{\eta U(t)}{Q_i T_j}\right\}. \tag{15}$$

Here, $E_{i,j}$ represents the mathematical expectation. When the loss function reaches its minimum, the derivative with respect to $U(t)$ is zero:

$$\frac{\partial \text{Loss}}{\partial U} = 2W(t)U(t) - 2W(t)U(t-1) + \eta / \left[Q(t)T(t)\right] = 0. \tag{16}$$

Solving this equation yields the optimal control for load

transfer at the t^{th} sampling time:

$$U^*(t) = U(t-1) - \eta / \left[2W(t)Q(t)T(t)\right]. \tag{17}$$

Moreover, during load migration between nodes, potential issues arising from task surges leading to node overload, thereby triggering additional load migration events, should be addressed to prevent oscillatory migration phenomena. To mitigate this issue, we propose a dynamic load migration scheduling algorithm (Algorithm 3) based on the load migration model.

Algorithm 3 Load migration scheduling

Require: N, $L_n(t)$, $Q_n(t)$, and $T_n(t)$ $(n = 1, 2, \ldots, N)$
for all $0 \le t \le T$ **do**
  $u_{\text{thr}}(t) = \text{mean}\{1 - L_{i,j}(t)\},\ i, j = 1, 2, \ldots, N$
  $\Phi\{\text{desc}\} \rightarrow \text{sort}\ L_{i,j}(t)$
  for all $i, j = 1, 2, ..., N$ **do**
    $u_{i,j}(t) = (L_{i,j}(t) - L_{i,j}(t-1)) / (Q_i(t)T_j(t))$
    while $\Phi\{\text{inx}\}$ by $1, 2, \ldots, N$ **do**
      if $L_{i,j}(t) + u_{i,j}(t) \ge u_{\text{thr}}(t)$ **then**
        $u_{i,j}(t) = 0,\ h_{i,j}(t) = 1.0$
        $\rightarrow$ transfer to $u_{i+1,j+1}(t)$ by $\Phi\{\text{inx} + 1\}$
      else
        $h_{i,j}(t) = 0.5$
      end if
    end while
    $U(t)$ += $u_{i,j}(t)$
  end for
  $U^*(t) \leftarrow Q_i(t), L_{i,j}(t), T_j(t)$
end for
return $U^*(t)$

Following the node queue design of Algorithms 1 and 2, Algorithm 3 controls the load migration behavior and prevents load-balancing oscillations by selecting nodes with lower TSW as migration targets and marking them for an increased selection probability in the subsequent sampling period. Fig. S3 shows the flowchart of Algorithm 3.

5 Experimental analysis

5.1 Experimental setup

The proposed MP-DMC framework employs a distributed container cloud architecture with centralized orchestration management. The hardware environment adopts a cluster of 10 Zhongke Tenglong TL550F-B servers (two 64-core 1.1 GHz S5000C CPUs, 64 GB RAM, 5 Gb/s network bandwidth, and 2.4 TB storage capacity). Fig. S4 illustrates the hardware deployment. These instances are run on the China Galaxy Kirin Advanced Server Operating System V10, and the data are stored in China Shenzhou General Database V7.0.

The framework comprises two distinct functional categories of cloud nodes: (1) core service nodes that handle message queuing clusters, distributed data processing, model version control, and persistent storage subsystems; (2) a computational cluster organized in a leader–follower configuration with five computing nodes. In addition, it provides human-in-the-loop interaction through a web-based client interface, as illustrated in Fig. S5.

The proposed MP-DMC framework implements three situational awareness simulation modalities. As shown in Fig. 6, these modalities employ the following heterogeneous data–model fusion paradigms:

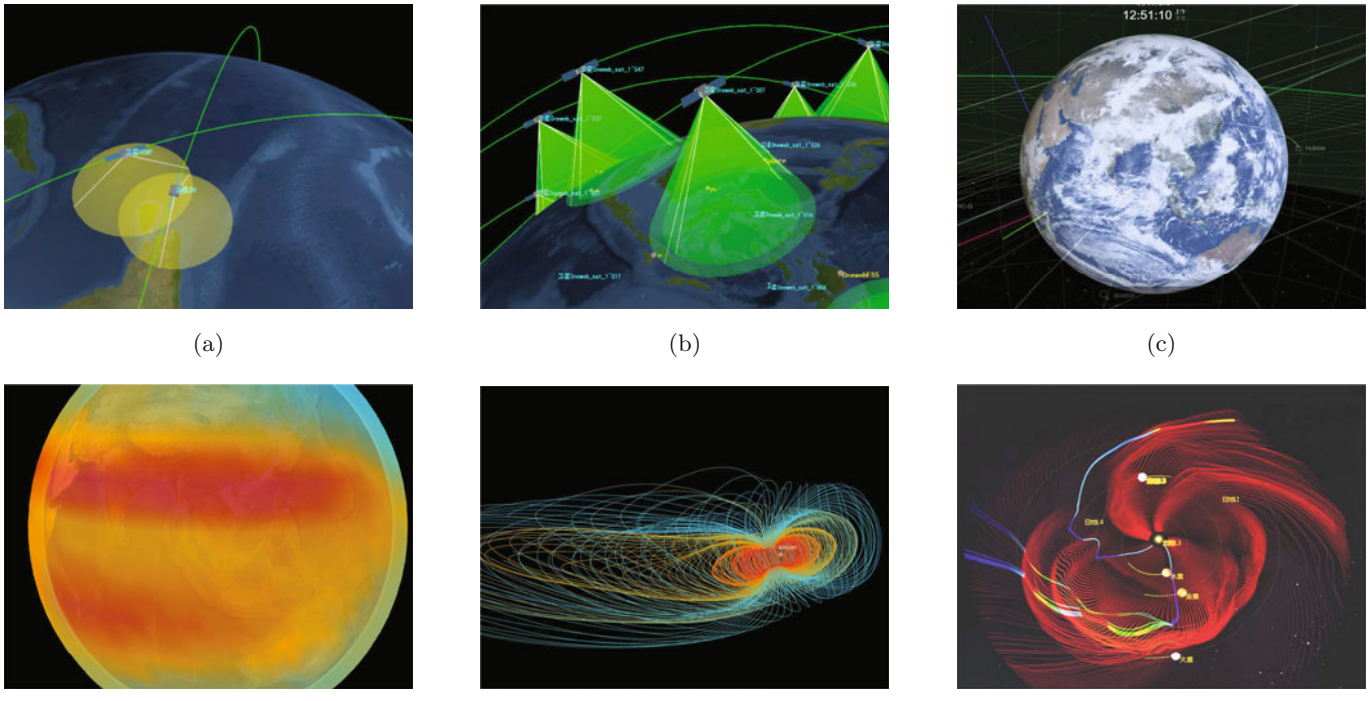


Fig. 6 Three-dimensional display results of data–model fusion simulation tasks: (a) spacecraft close-in; (b) spacecraft detection; (c) near-Earth atmosphere; (d) near-Earth ionosphere; (e) Earth–Moon magnetic field; (f) Earth–Moon solar wind. The computational complexity increases gradually from (a) to (f)

1. Spacecraft close-in simulation. It couples orbital prediction models with real-time telemetry streams from target spacecraft.

2. Spacecraft detection simulation. It integrates satellite sensor detection models with real-time telemetry streams from target spacecraft.

3. Space environment simulation. It synthesizes physics-based models (near-Earth atmosphere, near-Earth ionosphere, Earth–Moon magnetic, and Earth–Moon solar wind) with space weather indices (Ap/Kp/F10.7/sunspot).

These simulation tasks, while emulated computationally, are designed to replicate the characteristics of real-world spacecraft operations. The computational complexity increases progressively from models (a) to (f) in Fig. 6 by incrementally incorporating increasingly demanding processes, such as multisensor data fusion, collision probability forecasting, and high-fidelity orbital dynamics. This allows us to evaluate the scalability and robustness of the proposed MP-DMC framework under a controlled but representative set of load conditions.

5.2 Dynamic election analysis

This experiment is conducted to evaluate the performance differences between the leader–follower election algorithm in the MP-DMC framework and mainstream election algorithms, such as the Paxos, Raft, and PBFT protocols, through 100 election cycles across five nodes. Table S1 in the supplementary materials lists the configuration of the election algorithm parameters.

Fig. 7 shows that the conventional algorithms distribute leadership probabilistically, while the proposed MP-DMC framework maintains node 1 as the primary leader with 97% stability. This deterministic election strategy reduces topology reconfiguration overhead, thereby improving election efficiency and system availability.

To validate the efficacy of the proposed election algorithm in enhancing system reliability, a controlled experiment is conducted under equivalent workload (10^3 task units) stress testing conditions. Here, the system outage frequency induced by node failures is evaluated and compared among configurations employing this election algorithm and baseline scenarios without our election algorithm. In addition, to evaluate scalability beyond the physical infrastructure, we perform additional tests in a large-scale server cluster environment. Note that we are allocated only five physical cloud nodes. To test at a larger scale, we create 100 virtual nodes through containerization to benchmark the performance of the dynamic election algorithm. The experimental results are presented in Table 1.

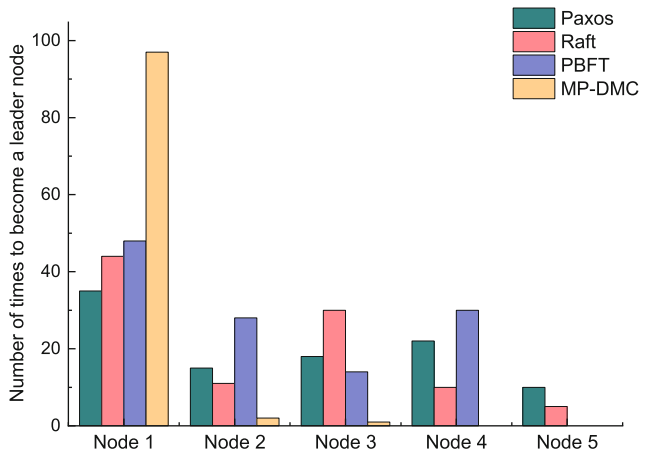


Fig. 7 Comparison of election algorithms in terms of leader election stability

Table 1 lists the number of service interruptions caused by system crashes. The results demonstrate that our leader–follower election algorithm provides a crucial fault tolerance mechanism by swiftly reelecting and replacing the main control node, thereby preventing system failures under thousand-level task loads. In addition, the results of scalability tests conducted on a large-scale virtualized cluster of 100 nodes confirm that the election algorithm maintains its effectiveness in even more complex environments, enhancing overall system availability and operational stability. Note that while virtualization enables flexible and scalable deployment, it may introduce differences in terms of resource isolation and network overhead compared with physical environments. However, the consistent performance observed across the virtualized nodes suggests that the algorithm's core mechanisms are robust to such environmental variations, supporting its applicability in real-world physical deployments. Furthermore, the performance advantages of the proposed MP-DMC framework over baseline algorithms can be attributed to its optimized election process, which reduces the time required for failure detection and leader reelection, thereby minimizing service disruptions.

5.3 Elastic expansion performance

5.3.1 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 is a dual-branch hybrid, where the lower branch captures smooth baseline trends via DMA, and the upper LSTM branch

Table 1 Reliability enhancement effects of leader–follower election algorithms after different running times, from 24 h to 168 h

Node number	System	Number of service interruptions						
		24	48	72	96	120	144	168
5	MP-DMC w/o election	0	0	1	2	2	3	3
	MP-DMC*	0	0	0	0	0	0	0
20	MP-DMC w/o election	0	0	1	1	2	2	3
	MP-DMC*	0	0	0	0	0	0	0
50	MP-DMC w/o election	0	0	2	2	2	4	5
	MP-DMC*	0	0	0	0	0	0	1
100	MP-DMC w/o election	0	1	1	2	4	5	6
	MP-DMC*	0	0	0	0	1	1	1

w/o: without. MP-DMC* denotes the proposed framework with the election algorithm

exclusively corrects short-term prediction residuals. In this evaluation, the model is 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. Here, each node's load (CPU, memory, and I/O) is 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 is split into training, validation, and test sets at a ratio of 7:2:1, and all features are 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.

The prediction accuracy of the DMA-LSTM model is evaluated in terms of the mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean square error (RMSE). The metrics are calculated as follows:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \times 100\%, \tag{18}$$

$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%, \tag{19}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \times 100\%. \tag{20}$$

Here, y_i and $\hat{y}_i$ denote the true value and the predicted value, respectively.

As summarized in Table 2, the proposed DMA-LSTM model achieves superior predictive accuracy, outperforming all baseline models across all evaluation metrics. Here, all error metrics are reported as percentages relative to the normalized load range [0, 100] to facilitate an effective comparison. Specifically, the DMA-LSTM model reduces RMSE by 1.39 percentage points (PPs), MAE by 0.47 PPs, and MAPE by 1.15 PPs compared with the standard LSTM model. These improvements stem from the DMA branch/mechanism's ability to selectively focus on relevant patterns, e.g., recurring workload trends and abrupt load spikes. By emphasizing these salient features, DMA-LSTM reduces prediction errors at challenging points, thereby improving the overall accuracy.

Table 2 Load prediction accuracy of models

Model	RMSE (%)	MAE (%)	MAPE (%)
Historical mean	12.56	9.45	13.23
SVM	6.84	5.22	7.41
LSTM	5.21	4.15	5.66
DMA	7.01	6.71	7.92
DMA-LSTM	3.82 ± 0.16	3.68 ± 0.11	4.51 ± 0.23

For the proposed DMA-LSTM model, the results are reported as the mean±standard deviation over five independent runs with different random initializations. The baseline results are reported as the best scores obtained during hyperparameter tuning (consistent with standard practice in time-series forecasting benchmarks)

The comparison in Fig. 8 demonstrates that the DMA-LSTM model more efficiently captures sudden load spikes and fluctuations, which are frequently missed or smoothed out by baselines. This superior predictive capability ensures that the subsequent elastic scaling algorithm can adjust resources proactively and accurately, forming the foundation for the robust performance of the proposed MP-DMC framework.

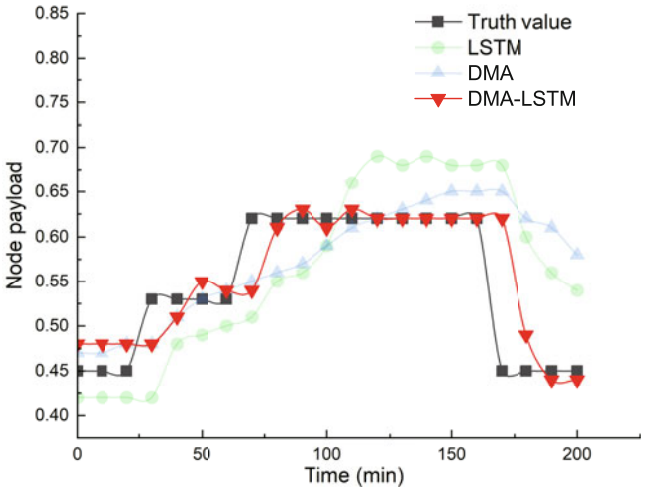


Fig. 8 Comparison of DMA-LSTM and baseline models in terms of load spike prediction accuracy

5.3.2 Dynamic elastic expansion analysis

A comparative analysis of the performance differences between the node dynamic elastic expansion algorithm with the DMA-LSTM model in the proposed MP-DMC framework and mainstream scaling algorithms, including HPA, DMA-HPA, ProSmart HPA, and RL, is conducted under varying task loads. Here, the performance metrics include expansion efficiency and effectiveness, with expansion effectiveness evaluated using the cluster load balance (CLB) index.

CLB refers to the degree of load distribution across cloud nodes. A smaller CLB value indicates a more evenly distributed allocation of resources.

$$C = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N}\left(\Delta L_i - \overline{\Delta L}\right)^2\right)}. \tag{21}$$

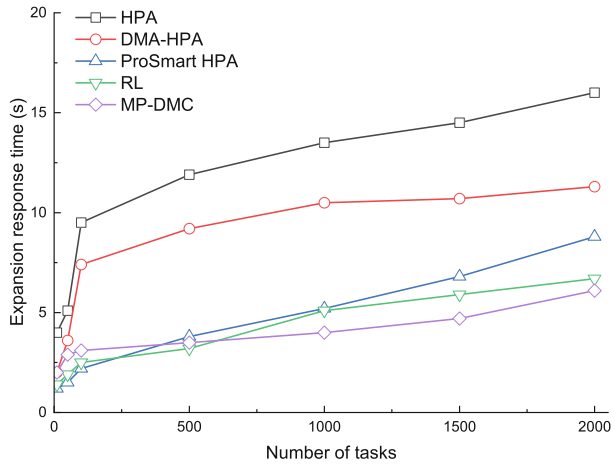
Here, N denotes the number of nodes, ΔL_i denotes the change in RLP for the i^{th} cloud node before and after receiving the computational task, and ΔL_i denotes the average change in the load proportion across all cloud nodes.

To ensure a fair and reproducible comparison, we configure the baseline algorithms as follows. For all HPA-based baselines (i.e., HPA, DMA-HPA, and ProSmart HPA), common Kubernetes scaling parameters are set uniformly. The target CPU utilization percentage is 75%, the stabilization window for scaling down and scaling up is 300 s and 60 s, respectively, to prevent thrashing, and the minimum and maximum numbers of pod replicas are set to 1 and 10, respectively. Beyond these common settings, each algorithm is configured with its own unique parameters, as detailed in Table S3.

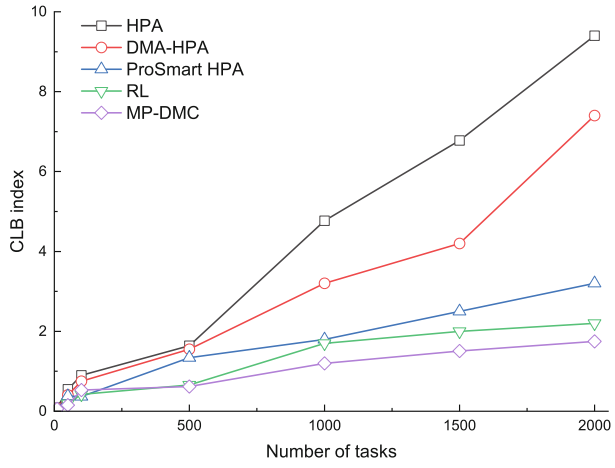
The RL-based baseline is implemented using a tabular Q-learning approach, following the threshold adjustment policy proposed by Rossi et al. (2020). Unlike deep Q-networks (DQNs), which use neural networks for function approximation, this implementation employs a discretized state space mapped to a Q-table.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

operational environment. The key learning parameters of the tabular Q-learning agent are provided in Table S4.

The experimental results are shown in Fig. 9. The results demonstrate that the different algorithms exhibit distinct elastic expansion characteristics under varying workloads. For example, as the task load increases, both HPA and DMA-HPA exhibit growth patterns in elastic expansion response time and CLB. This is primarily because higher workloads introduce greater scheduling complexity and resource contention, inevitably prolonging decision-making and adjustment processes. Specifically, experimental analysis reveals a critical workload threshold at 100 tasks, beyond which all resource elastic expansion algorithms demonstrate more stable growth patterns in expansion response time. This plateauing behavior indicates their enhanced suitability for large-scale scheduling scenarios compared with smaller task volumes.



(a)



(b)

Fig. 9 Comparison of elastic expansion performance under varying task workloads: (a) expansion response time; (b) CLB index

Remarkably, the proposed MP-DMC framework demonstrates superior scalability with its response time remaining consistently below 6 s across all tested scales despite increasing data volumes. This advantage stems from the predictive capabilities of the DMA-LSTM model, which forecasts workload spikes accurately and allows the elastic scaling algorithm to proactively provision resources before demand surges. Consequently, the MP-DMC framework avoids the reactive delays that cause the compared algorithms' response times to escalate under heavy load conditions. Under small-scale task loads, ProSmart HPA and RL outperform the MP-DMC framework. However, beyond 500 tasks, the MP-DMC framework's CLB value increases at a lower rate compared with the other algorithms, and it achieves a shorter elastic expansion response time. These results indicate that the MP-DMC framework is more effective in suppressing cluster load imbalance under heavier workloads. In contrast, the baseline algorithms frequently rely on threshold-based or reactive policies that can trigger overprovisioning or underprovisioning, thereby exacerbating cluster imbalance. These quantitative comparisons highlight the MP-DMC framework's architectural advantages in terms of maintaining system stability during workload fluctuations.

5.4 Migration scheduling effectiveness

We also perform a comprehensive comparative analysis of scheduling algorithms across three critical situational awareness simulation scenarios, i.e., spacecraft close-in, spacecraft detection, and space environment modeling. Here, we benchmark the load migration algorithm against the classic round-robin, purely random, and weighted random scheduling strategies.

The configuration of baseline scheduling algorithms is detailed in Table S5. The performance is evaluated in terms of the task response time and node load-balancing time. The latter metric is operationally defined as the mean maximum duration required for RLP stabilization across compute nodes during scheduling transitions.

The experimental results are shown in Fig. 10. In the near-Earth ionosphere simulations, the MP-DMC framework achieves an average response time of 45.2 ms in 10–10^3 task units, with an average node load-balancing time of 166.34 s. For Earth–Moon solar wind modeling, the proposed framework obtains an average latency of 52.4 ms and an average load stabilization time of 390.20 s in the same task units. In this scenario, the increased stabilization time reflects the greater difficulty of balancing workloads under more volatile task characteristics. However, the MP-DMC framework maintains relatively stable response times, highlighting the robustness of its predictive scheduling mechanism. Compared with the best-performing weighted random scheduling algorithm, the MP-DMC framework reduces the task response time to the 12.8–21.4 ms range for each task type. In addition, it reduces the node load-balancing time to the 13.0–78.34 s range, demonstrating faster stabilization. These improvements are primarily attributable to the DMA-LSTM model's ability to forecast workload fluctuations with higher accuracy, coupled with an effective load migration strategy that enables proactive (rather than reactive) resource allocation. In contrast, weighted random scheduling relies on static probabilities or heuristic weights, which cannot adapt to real-time changes in the system state, frequently resulting in a suboptimal distribution and longer convergence times. To offer a clearer overview, the average results from

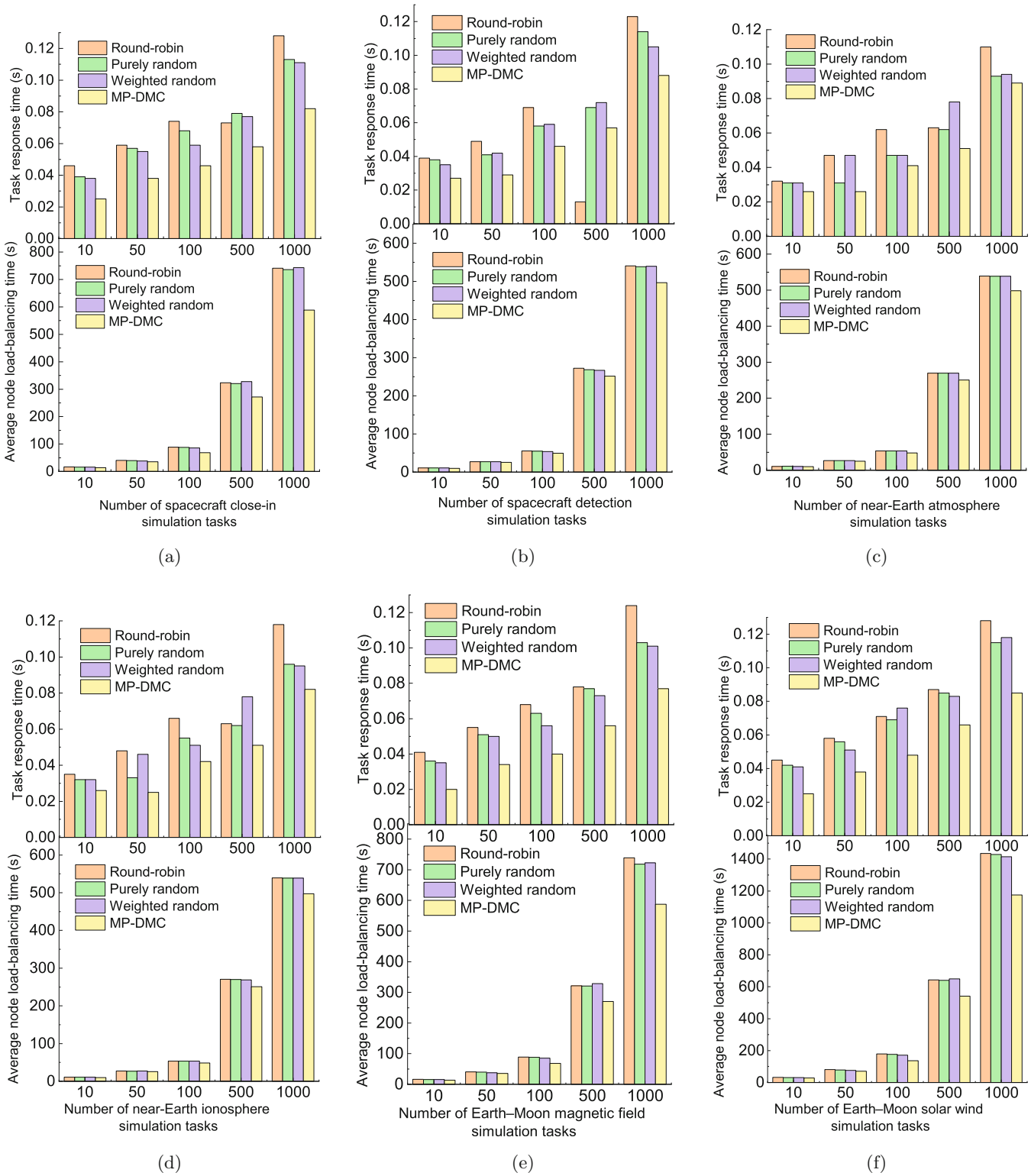


Fig. 10 Performance comparison of task scheduling algorithms. Each graph compares the task response time and node load-balancing time of the scheduling algorithms under different simulation tasks: (a) spacecraft close-in; (b) spacecraft detection; (c) near-Earth atmosphere; (d) near-Earth ionosphere; (e) Earth–Moon magnetic field; (f) Earth–Moon solar wind

Table 3 Task response time and load-balancing time for different numbers of simulation tasks

Algorithm	Average task response time (s)					Average node load-balancing time (s)				
	10	50	100	500	1000	10	50	100	500	1000
Round-robin	0.039	0.053	0.068	0.063	0.121	16.170	40.410	86.450	350.083	755.897
Purely random	0.036	0.045	0.060	0.072	0.106	15.873	39.815	85.790	348.258	749.950
Weighted random	0.035	0.049	0.058	0.077	0.104	15.998	38.953	83.922	351.658	749.688
MP-DMC*	0.025	0.032	0.048	0.054	0.084	14.121	36.488	69.892	305.875	640.472

The values reported for MP-DMC* represent the mean over five independent runs. The standard deviations for the response time and load-balancing time of MP-DMC* range from ±0.002 s to ±0.007 s and ±0.7 s to ±3.2 s, respectively (increasing slightly with the task count)

the six simulation tasks are summarized in Table 3. The results consistently demonstrate the scheduling advantages of the proposed MP-DMC framework.

Finally, ablation experiments are conducted to isolate and evaluate the individual contributions of the three core components in the proposed MP-DMC framework, i.e., the election-optimized leader–follower mechanism, the DMA-LSTM-based predictive model for dynamic elastic scaling, and the load migration strategy. Here, we construct three ablated variants: (1) –E, which replaces the leader–follower election mechanism with a static master node without automatic reelection; (2) –S, which employs a fixed number of cluster nodes; (3) –M, which disables the dynamic task migration policy and falls back to a static scheduling strategy. All variants are evaluated under the same simulation tasks, and the results are presented in Table 4.

Table 4 Contributions of core MP-DMC components for different numbers of tasks

Configuration	Response time (s)			Load-balancing time (s)		
	10	100	1000	10	100	1000
Full	0.03	0.05	0.08	14.12	69.89	640.47
–E	0.03	0.05	0.11	16.15	84.27	752.13
–S	0.02	0.05	0.09	19.41	105.34	925.21
–M	0.02	0.05	0.11	24.39	132.52	1180.88

All values represent the mean over five independent runs. The standard deviations for response time and load-balancing time range from ±0.001 s to ±0.004 s and ±0.5 s to ±4.2 s, respectively (increasing proportionally with the task count)

The results demonstrate that each component plays a critical and distinct role in achieving the overall performance of the proposed MP-DMC framework. For example, removing the leader–follower election mechanism results in a consistent increase in both response time and load-balancing time across all task scales. At 1000 tasks, the response time increases from 0.08 s to 0.11 s, while the load-balancing time increases by approximately 17.4%. This degradation occurs because, without automatic reelection, any failure or bottleneck at the master node cannot be recovered quickly, leading to prolonged scheduling delays and imbalanced load distribution. In addition, disabling autoscaling leads to an even more pronounced impact, particularly on load-balancing time. This confirms that the DMA-LSTM model's elastic scaling mechanism is essential for dynamically provisioning resources in response to predicted load spikes, thereby preventing severe imbalance under heavy workloads. Note that the "w/o autoscaling" configuration disables the entire scaling pipeline. The isolated contribution of the DMA-LSTM predictor over simpler forecasting baselines is quantified separately in Table 2, where DMA-LSTM achieves a reduction of 1.39 PPs in RMSE compared with the standard LSTM. The most significant degradation is observed when the migration strategy is removed. At 1000 tasks, the load-balancing time surges to 1180.88 s, nearly double that of the full MP-DMC framework. Without migration, tasks remain on overloaded nodes, and the scheduler cannot actively rebalance the system. This highlights the importance of migration as a fine-grained adjustment mechanism that complements predictive scaling.

A further analysis of the results in Table 4 reveals that the relative importance of these three components is highly dependent on the scale of workload. Under a light load (10 tasks), the performance gaps among the different configurations are negligible, which indicates that the system operates efficiently even without a dynamic election mechanism. As the workload increases to a moderate level (100 tasks), the absence of autoscaling begins to impose an observable penalty; i.e., the load-balancing time increases by approximately 50.7% compared with the full MP-DMC framework, whereas removing the election mechanism causes an increase of only 20.6%. These results demonstrate that predictive autoscaling (driven by the DMA-LSTM model) is the dominant performance enabler as the workload accumulates by proactively provisioning resources before congestion occurs. In the high-load scenario (1000 tasks), the "w/o migration" configuration suffers from the most severe penalty. Here, the load-balancing time exhibits an 84% increase over the full MP-DMC framework, far exceeding the degradation observed in the "w/o autoscaling" (44.5%) and "w/o election" (17.4%) configurations. This stark contrast confirms that dynamic task migration is indispensable. In summary, the results of the ablation study validate the complementary nature of the MP-DMC framework's three core mechanisms. Specifically, the election mechanism offers lightweight fault tolerance across all scales, autoscaling delivers proactive capacity expansion under moderate growth, and migration acts as the critical safety valve during extreme load surges.

6 Conclusions

This paper has proposed the distributed MP-DMC framework for data–model fusion computation in space situational awareness simulations. The proposed framework addresses three critical challenges in cloud-based simulation systems. First, inefficient node orchestration is resolved through a

priority-based dynamic election algorithm that reduces leader-node changes by 97% compared with the Paxos and Raft algorithms. Second, imbalanced resource allocation under high concurrency is mitigated by the DMA-LSTM model's elastic expansion strategy, achieving response times below 6 s for 2000 tasks, outperforming existing algorithms, including HPA, DMA-HPA, ProSmart HPA, and RL. Third, load-balancing oscillations during data–model fusion are suppressed via TSW-prioritized migration, achieving faster stabilization compared with round-robin, purely random, weighted random, and other traditional task scheduling approaches.

Beyond the aerospace situational awareness simulation context, the proposed MP-DMC framework demonstrates broader applicability due to its stable leader selection, proactive resource allocation based on workload forecasting, and efficient load balancing under dynamic conditions. In particular, the core architecture of the DMA-LSTM model, which decouples trend prediction from residual learning, addresses a class of time-series forecasting problems characterized by non-stationarity. This design is not dependent on domain-specific assumptions, which makes it suitable for other scenarios with similar data patterns, including Industrial Internet of Things platforms that require reliable node coordination and prediction and cloud data centers that rely on predictive resource scaling for cost efficiency.

Future work will proceed with a focus on two complementary directions. First, we plan to extend the MP-DMC framework to deep space scenarios, e.g., digital simulation of Earth–Moon space and deep space exploration mission planning. These environments impose even greater demands on node coordination, resource elasticity, and load stability due to longer communication delays, highly volatile workloads, and elevated mission criticality. The proposed MP-DMC framework's demonstrated capabilities in addressing analogous challenges within the current context provide a strong foundation for such extensions. Second, we intend to extend the evaluation of the leader–follower election mechanism by quantifying leader reelection latency and task recovery time during sudden node failures. In addition, we plan to explore predictive failure models and finer-grained component-level validation under fault and stress conditions to enable proactive leader handover and enhance system stability for longer task cycles in these extreme environments. Collectively, we expect that these efforts will harden the MP-DMC framework for the unprecedented coordination challenges posed by next-generation space exploration architectures.

Supplementary information

The online version contains supplementary materials available at https://doi.org/10.1631/ENG.ITEE.2025.0154.

Acknowledgments

This work was supported by the Key Deployment Project of Chinese Academy of Sciences (No. KGFZD-145-24-33).

Author contributions

Runnan QIN conceived the method and algorithms. Runnan QIN and Zeyu FAN wrote the software code. Runnan QIN and Wenming XIE designed and performed experiments. Xiao ZHENG implemented and optimized the algorithms. Runnan QIN and Jingyi REN drafted the paper. Runnan QIN revised the paper. Xiaodong PENG and Zhen YANG supervised the research. All the authors approved the final paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declaration on the use of generative AI tools

During the preparation of this work, the authors used DeepSeek to improve language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

Ahmad H, Treude C, Wagner M, et al., 2025. Towards resource-efficient reactive and proactive auto-scaling for microservice architectures. *J Syst Softw*, 225:112390. https://doi.org/10.1016/j.jss.2025.112390

Cui HT, Zhang C, Ding X, et al., 2021. Evaluation framework for development organizations' adaptability to micro-services architecture. *J Softw*, 32(5):1256-1283 (in Chinese). https://doi.org/10.13328/j.cnki.jos.006232

Dayal J, Bratcher D, Eisenhauer G, et al., 2014. Flexpath: type-based publish/subscribe system for large-scale science analytics. 14th IEEE/ACM Int Symp on Cluster, Cloud and Grid Computing, p.246-255. https://doi.org/10.1109/CCGrid.2014.104

Ding TC, Qian SY, Zhu WD, et al., 2020. Comat: an effective composite matching framework for content-based pub/sub systems. IEEE Int Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, p.236-243. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00055

Ding WH, Ding XH, Xu ZP, et al., 2025. Simulation technology for large-scale satellite cloud network with the integration of digital and reality. *Space Integr Ground Inform Netw*, 6(1):64-77 (in Chinese). https://doi.org/10.11959/j.issn.2096-8930.2025008

Dragoni N, Lanese I, Larsen ST, et al., 2017. Microservices: how to make your application scale. http://arxiv.org/abs/1702.07149

Feng ZY, Xu YW, Xue X, et al., 2020. Review on the development of microservice architecture. *J Comput Res Dev*, 57(5):1103-1122 (in Chinese). https://doi.org/10.7544/issn1000-1239.2020.20190460

Guo W, 2019. DolphinScheduler. https://dolphinscheduler.apache.org/zh-cn/ [Accessed on Oct. 5, 2025].

Lamport L, 1998. The part-time parliament. *ACM Trans Comput Syst*, 16(2):133-169. https://doi.org/10.1145/279227.279229

Larsson M, 2021. Microservices with Spring Boot and Spring Cloud: Build Resilient and Scalable Microservices Using Spring Cloud, Istio, and Kubernetes (2nd Ed.). Packt Publishing, Birmingham, UK.

Lei X, Yang LM, Zhu YY, et al., 2024. Digital-analog hybrid simulation platform construction and characteristic research of Baihetan-Jiangsu hybrid cascaded UHVDC project. *Power Syst Technol*, 48(1):434-442 (in Chinese). https://doi.org/10.13335/j.1000-3673.pst.2023.0579

Li L, Li HZ, Li T, 2024. Multi-primary-node Byzantine fault-tolerant consensus mechanism based on Raft. *J Guangxi Norm Univ (Nat Sci Ed)*, 42(3):122-130 (in Chinese). https://doi.org/10.16088/j.issn.1001-6600.2023100805

Liang F, 2017. Apache Dubbo. https://github.com/apache/dubbo [Accessed on Oct. 5, 2025].

Liu F, Weissman JB, 2015. Elastic job bundling: an adaptive resource request strategy for large-scale parallel applications. Proc Int Conf for High Performance Computing, Networking, Storage and Analysis, Article 33. https://doi.org/10.1145/2807591.2807610

Liu WG, Cui JH, Li TT, et al., 2023. A space-efficient fair cache scheme based on machine learning for NVMe SSDs. *IEEE Trans Parall Distrib Syst*, 34(1):383-399. https://doi.org/10.1109/TPDS.2022.3221410

Luo H, Jiang W, Liu MW, et al., 2022. Multi resource load balancing optimization method based on microservice architecture. *Sci Technol Eng*, 22(5):1965-1971 (in Chinese). https://doi.org/10.3969/j.issn.1671-1815.2022.05.030

Ma HC, Si L, Tan C, 2023. Research on VR system of shearer based on fusion of sensor data and model. *Coal Mine Mach*, 44(6):200-204 (in Chinese). https://doi.org/10.13436/j.mkjx.202306061

Ma Y, Yan RY, Sun JW, et al., 2017. A MQTT protocol message push server based on RocketMQ. 10th Int Conf on Intelligent Computation Technology and Automation, p.295-298. https://doi.org/10.1109/ICICTA.2017.72

Mesbahi MR, Rahmani AM, Hosseinzadeh M, 2019. Dependability analysis for characterizing Google cluster reliability. *Int J Commun Syst*, 32(16):e4127. https://doi.org/10.1002/dac.4127

Nguyen TT, Yeom YJ, Kim T, et al., 2020. Horizontal pod autoscaling in Kubernetes for elastic container orchestration. *Sensors*, 20(16):4621. https://doi.org/10.3390/s20164621

Pathak A, Kalaiarasan C, 2021. RabbitMQ queuing mechanism of publish subscribe model for better throughput and response. 4th Int Conf on Electrical, Computer and Communication Technologies, p.1-7. https://doi.org/10.1109/ICECCT52121.2021.9616722

Pivotal Software, Inc., 2018. Spring Cloud Scheduler. https://github.com/spring-attic/spring-cloud-scheduler [Accessed on Oct. 5, 2025].

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

Shona M, Sharma R, 2023. Implementation and comparative analysis of static and dynamic load balancing algorithms in SDN. Int Conf for Advancement in Technology, p.1-7. https://doi.org/10.1109/ICONAT57137.2023.10080430

Shrimal M, Sharma AK, Patel M, 2024. A novel hybrid load balancing method to achieve quality of service (QoS) in cloud-based environments. 3rd Int Conf on Sentiment Analysis and Deep Learning, p.325-328. https://doi.org/10.1109/ICSADL61749.2024.00059

Tsenos M, Kalogeraki V, 2021. Dynamic rate control for topic-based pub/sub systems. 22nd IEEE Int Conf on Mobile Data Management, p.272-273. https://doi.org/10.1109/MDM52706.2021.00057

Ullah MH, Park SS, No J, et al., 2013. A collaboration mechanism between wireless sensor network and a cloud through a pub/sub-based middleware service. 5th Int Conf on Evolving Internet, p.38-42.

Wang YZ, Yu JQ, Yu ZB, 2023. Resource scheduling techniques in cloud from a view of coordination: a holistic survey. *Front Inform Technol Electron Eng*, 24(1):1-40. https://doi.org/10.1631/FITEE.2100298

Wei MD, Li YF, Wang YY, et al., 2025. Optimizing real-time e-commerce data streams: a comparative analysis of serialization and topic design patterns in Apache Kafka. Int Conf on Computing, Networking and Communications, p.334-339. https://doi.org/10.1109/ICNC64010.2025.10993561

Xu XL, 2015. XXL-JOB, a Distributed Task Scheduling Framework. https://github.com/xuxueli/xxl-job [Accessed on Oct. 5, 2025].

Xu ZC, Zhao DP, Liang WF, et al., 2023. HierFedML: aggregator placement and UE assignment for hierarchical federated learning in mobile edge computing. *IEEE Trans Parall Distrib Syst*, 34(1):328-345. https://doi.org/10.1109/TPDS.2022.3218807

Yang T, Zhou XS, Lin Y, 2004. A publish–subscribe system based on CORBA. *Comput Eng*, 30(10):77-78 (in Chinese). https://doi.org/10.3969/j.issn.1000-3428.2004.10.029

Yin JB, He SJ, Zhao F, et al., 2015. Design and implementation of intelligent load-balancing heterogeneous data source middleware based on ActiveMQ and XML. Int Conf on Industrial Informatics—Computing Technology, Intelligent Technology, Industrial Information Integration, p.255-258. https://doi.org/10.1109/ICIICII.2015.145

Zhang CY, Chen WY, Chen Q, et al., 2021. Distributed intelligent reflecting surfaces-aided device-to-device communications system. *J Commun Inform Netw*, 6(3):197-207. https://doi.org/10.23919/jcin.2021.9549117

Zhang Y, 2023. Research on Fault Diagnosis of Piston Pump Based on Digital Analog Fusion. MS Thesis, North University of China, Taiyuan, China (in Chinese). https://doi.org/10.27470/d.cnki.ghbgc.2023.000897

Zheng BY, Gao YH, Guo XK, 2025. BE-Raft: a Raft consensus algorithm with enhanced election determinism. *Comput Simul*, 42(11):326-330, 396 (in Chinese). https://doi.org/10.3969/j.issn.1006-9348.2025.11.063