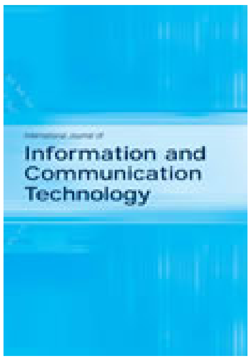


International Journal of Information and Communication Technology

ISSN online: 1741-8070 - ISSN print: 1466-6642
https://www.inderscience.com/ijict

Optimisation of gradient descent algorithm and resource scheduling in big data environment

Zhendong Ji

DOI: 10.1504/IJICT.2025.10071986

Article History:

Received:	06 May 2025
Last revised:	23 May 2025
Accepted:	23 May 2025
Published online:	10 July 2025

Optimisation of gradient descent algorithm and resource scheduling in big data environment

Zhendong Ji

School of Data Science and Computer Science,
Shandong Women's University,
Jinan 250300, China
Email: jzd@sdwu.edu.cn

Abstract: With the rapid development of big data technology, traditional gradient descent algorithms face problems such as low computational efficiency, slow convergence speed, and uneven resource allocation. This article proposes a collaborative framework that integrates dynamic resource scheduling and adaptive gradient descent optimisation for distributed machine learning scenarios in big data environments. Firstly, an asynchronous gradient descent algorithm based on hierarchical batch sampling (HB-ASGD) was designed, which dynamically adjusts the local batch size and global synchronisation frequency to balance the load differences between computing nodes and reduce communication overhead. Secondly, the resource aware elastic scheduling (RAES) model is introduced to dynamically predict task computation using reinforcement learning, and combined with containerisation technology to achieve fine-grained allocation of CPU/GPU resources, prioritising the protection of computing resources for critical iterative tasks. The experiment shows that this study effectively solves the efficiency bottleneck problem in massive data iteration.

Keywords: big data; gradient descent algorithm; resource scheduling; reinforcement learning; resource aware elastic scheduling; RAES.

Reference to this paper should be made as follows: Ji, Z. (2025) 'Optimisation of gradient descent algorithm and resource scheduling in big data environment', *Int. J. Information and Communication Technology*, Vol. 26, No. 25, pp.19–32.

Biographical notes: Zhendong Ji received his Master's in Software Engineering from the Shandong University in 2011. He is now working at the Shandong Women's University. His research interests are data mining (DM) and mathematical modelling.

1 Introduction

In recent years, with the rapid development of the internet of things, social networks and industrial internet, the global data scale shows exponential growth. IDC predicts that the global data volume will reach 175ZB by 2025, with over 80% of the data having high-dimensional, high noise, and unstructured features. In this context, machine learning model training faces two core challenges: on the one hand, traditional gradient descent (GD) algorithms suffer from problems such as decreased iterative convergence speed and local optimal traps when processing TB level data (Haji and Abdulazeez, 2021); on the other hand, the static allocation strategy of hardware resources (such as GPU memory and network bandwidth) in distributed computing clusters is difficult to adapt to dynamically changing computing loads, resulting in low resource utilisation. How to achieve collaborative optimisation of algorithm efficiency and resource efficiency has become a key issue in the field of big data machine learning (Tian et al., 2023).

In terms of optimising GD algorithm, researchers mainly improve it from two dimensions: one is to improve computational efficiency through data parallelisation, and the other is to accelerate convergence through gradient update strategy. The DistBelief system proposed by Dean et al. (2012) achieved large-scale asynchronous stochastic GD for the first time, decoupling computing nodes from parameter updates through a parameter server architecture, resulting in a training speed increase of over ten times. However, its fixed batch size design can easily lead to gradient delay differences between nodes. In response to this issue, Zhang et al. (2015) proposed elastic average SGD, which allows each node to dynamically adjust the local batch processing volume, but does not consider the correlation between communication overhead and computational load. In terms of convergence optimisation, Yi et al. (2020) constructed an effective non-convex cost function optimisation method. This method solves the problem of getting stuck in local minima by adding a cost function to the parameter update rules of the ADAM method. Through numerical comparison with gradient descent (GD, ADAM, and AdaMax), the convergence of the sequences generated by the proposed method and the superiority of the proposed method have been demonstrated.

In the field of resource scheduling, existing research mainly focuses on two directions: static resource allocation and dynamic task orchestration. Dakić et al. (2024) proposed a different architecture based on seamless hardware integration and user-friendly UI. It also provides dynamic workload placement based on real-time performance analysis and prediction, as well as machine learning based scheduling. Recently, Deng et al. (2023) used deep reinforcement learning (DRL) to optimise the deployment of LRA class containers. The proposed non-generic model can customise specialised models for each container group, providing high-quality placement and low training complexity; meanwhile, the proposed batch deployment scheme can optimise various scheduling objectives that are not directly supported by existing constraint based schedulers, such as minimising SLO violations. However, the above methods often treat algorithm execution and resource management as independent problems, lacking cross layer collaborative optimisation mechanisms.

Despite significant progress in research, there are still key issues in practical industrial level big data scenarios: existing GD optimisation methods often assume unlimited hardware resource supply and ignore the impact of memory limitations and communication bottlenecks on convergence speed (Chaudhary et al., 2022). For example, synchronous SGD in heterogeneous clusters often leads to a decrease in overall efficiency due to ‘slow node’ issues. Traditional resource schedulers (such as YARN and Mesos) adopt a periodic resource allocation strategy, which makes it difficult to respond in real-time to sudden load fluctuations in iterative calculations (Cao and Su, 2023). The current system design generally separates the algorithm layer (such as batch processing strategy) from the resource layer (such as container orchestration), failing to establish a joint optimisation objective function, resulting in potential performance loss (Rivas et al., 2024).

In response to the above challenges, this article proposes an algorithm and resource collaborative optimisation framework, with core innovations including:

1 Design a HB-ASGD that dynamically adjusts the local gradient computation and global synchronisation frequency to achieve Pareto optimality in communication overhead and convergence rate.

2 Build a resource aware elastic scheduling (RAES), combine LSTM network to predict the computational requirements of iterative tasks, and design a dynamic scaling strategy for containerised resources to achieve fine-grained allocation of CPU/GPU resources.

2 Relevant technologies

2.1 GD algorithm

The GD algorithm, as the core method for optimising machine learning models (Ahn et al., 2023), aims to minimise the parameter $\theta \in \mathbb{R}^d$ of the objective function $J(\theta)$ through iterative search. The basic idea is to gradually adjust the parameters along the negative gradient direction by calculating the gradient direction of the objective function. The parameter update equation is:

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) \tag{1}$$

where θ_t represents the parameter vector at the tth iteration, α is the learning rate (controlling the update step size), and $\nabla J(\theta_t)$ is the gradient of the objective function at θ_t, that is, the vector composed of partial derivatives of each dimension. When initialising the algorithm, randomly select θ_0 and calculate the gradient and update the parameters in each iteration until the gradient norm $\|\nabla J(\theta_t)\|$ is less than the preset threshold or reaches the maximum number of iterations.

With the expansion of data scale, various variants of classical GD have emerged to balance computational efficiency and convergence stability. Batch gradient descent (BGD) uses all N samples to calculate the average gradient, and its update equation is:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla J_i(\theta_t) \tag{2}$$

where $J_i(\theta_t)$ represents the loss function of the ith sample. The gradient estimation of BGD is unbiased and the convergence direction is stable, but each iteration requires traversing the entire data, resulting in a time complexity of up to $O(Nd)$, making it difficult to cope with big data scenarios. For this purpose, SGD adopts a single sample random sampling strategy (Chen et al., 2021), and the updated formula is simplified as:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla J_{i_t}(\theta_t) \tag{3}$$

where i_t is a randomly selected sample index. SGD reduces the complexity of a single computation to $O(d)$, but the gradient estimation variance is large, which may cause parameter update oscillations. To balance efficiency and stability, mini batch gradient descent (MBGD) introduces a batch size of *m*, and randomly selects m samples (referred to as set B_t) from the dataset each time to calculate the average gradient:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{m} \sum_{i \in B_t} \nabla J_i(\theta_t) \tag{4}$$

MBGD has become the mainstream choice for distributed training by adjusting the gradient variance of *m* and utilising GPU parallel computing acceleration.

The convergence of GD algorithm depends on the properties of the objective function and the learning rate scheduling strategy. For convex functions that satisfy L-Lipschitz continuous gradient, if the fixed learning rate is set to $\alpha = 1/L$, the upper bound of the error of BGD after *T* iterations is:

$$J(\theta_T) - J(\theta^*) \le \frac{L\|\theta_0 - \theta^*\|^2}{2T} \tag{5}$$

where θ^* is the global optimal solution, indicating that its convergence rate is $O(1/T)$. However, actual machine learning models often involve non-convex optimisation, and in this case, a decay learning rate is required to meet the Robbins Monroe condition. Under these conditions, the gradient norm of SGD converges to zero with probability:

$$\liminf_{t \to \infty} E\left[\|\nabla J(\theta_t)\|^2\right] = 0 \tag{6}$$

Although the convergence rate has decreased to $O(1/\sqrt{T})$, the actual training efficiency can be significantly improved by dynamically adjusting the learning rate or adaptive methods. The collaborative design of learning rate scheduling mechanism and gradient estimation method has become a key direction in the current research of large-scale optimisation algorithms (Li et al., 2023).

2.2 Reinforcement learning

Reinforcement learning is an important branch of machine learning, whose core idea is to enable agents to learn optimal decision strategies through dynamic interaction with the environment (Matsuo et al., 2022). This process simulates the learning mode of organisms adapting to the environment through trial and error mechanisms, and is widely used in complex decision-making scenarios such as robot control, game AI, and resource scheduling. Unlike supervised learning that relies on static labelled data, the core challenge of reinforcement learning is how to gradually approach strategies that can maximise long-term returns through trial and error exploration and experience accumulation in an environment without prior knowledge.

The mathematical foundation of reinforcement learning problems is Markov Decision Process (MDP). MDP assumes that the state transitions of the environment have Markovian properties, meaning that future states depend only on the current state and actions, and are independent of history. A standard MDP is defined by the following elements:

1 State space: the collection of all environmental states that an intelligent agent may perceive, such as vehicle position, speed, and other information in autonomous driving (Shakya et al., 2023).

2 Action space: a collection of actions that an intelligent agent can perform, such as the direction of movement of robot joints or key operations in games.

3 State transition probability: the probability distribution of transitioning to state s after executing action a, reflecting the dynamic uncertainty of the environment.

4 Reward function: real-time feedback from the environment on the actions of the agent, such as an increase in scores or a decrease in energy consumption in the game.

5 Discount factor: used to balance the importance of current rewards and future returns, avoiding the divergence problem of accumulating rewards over an infinite period of time.

The goal of the intelligent agent is to find a strategy $\pi(a|\ s)$, which is to select the probability distribution of action a in each state s, so as to maximise the expected cumulative discount reward from the initial state:

$$J(\pi) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, a_t, s_{t+1}\right)\right] \tag{7}$$

This objective function embodies the core optimisation direction of reinforcement learning: seeking a balance between immediate benefits and long-term value.

To quantify the advantages and disadvantages of strategies, reinforcement learning introduces state value function $V^\pi(s)$ and action value function $Q^\pi(s, a)$. The state value function represents the expected cumulative reward that can be obtained by following policy π starting from state s, while the action value function is further refined to the long-term value after executing a specific action a in state s. The mathematical definitions of the two are:

$$V^\pi(s) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R\left(s_{t+k}, a_{t+k}, s_{t+k+1}\right)|s_t = s\right] \tag{8}$$

$$q^\pi(s) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R\left(s_{t+k}, a_{t+k}, s_{t+k+1}\right)|s_t = s, a_t = a\right] \tag{9}$$

According to Markov property, the value function can be recursively decomposed through the Bellman equation, decomposing the long-term reward into the sum of the discounted value of the current reward and subsequent states (Shinn et al., 2023). For example, the Bellman equation for the state value equation is:

$$V^\pi(s) = \sum_{a\in A} \pi(a\mid s) \sum_{s'\in S} P\left(s'|s, a\right)\left[R(s, a, s') + \gamma V^\pi(s')\right] \tag{10}$$

This equation reveals the core idea of dynamic programming: to solve the global optimal solution through the recursive relationship of local optimal substructures. However, in real environments, the state transition probability P and reward function R are often unknown and need to be estimated through data-driven methods.

Q-learning is a classic model free reinforcement learning algorithm, whose core is to iteratively update the action value function $Q(s, a)$ to approximate the optimal strategy.

This algorithm adopts the concept of temporal difference and updates the Q-value by combining the current reward with the optimal estimate of the next state:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[R(s_t, a_t, s_{t+1}) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right] \tag{11}$$

where α represents the learning rate, and $\max_{a'} Q(s_{t+1}, a')$ represents the greedy selection of the optimal action value for the next state. The key feature of Q-learning lies in its off policy property, which allows the use of historical empirical data during the update process without strictly following the actions generated by the current policy, thereby improving data utilisation efficiency.

In high-dimensional or continuous action spaces, methods based on value functions face the problem of exploding complexity in action selection. The policy gradient method directly parameterises policy $\pi_\theta(s, a)$ and optimises policy parameter θ through gradient ascent to maximise the expected return. According to the policy gradient theorem, the gradient of objective function $J(\theta)$ can be expressed as:

$$\nabla_\theta J(\theta) = E_{\pi\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot Q^\pi(s_t | a_t)\right] \tag{12}$$

This formula indicates that the direction of strategy optimisation is determined by the advantage of the action (i.e., the degree to which the value brought by the action is higher than the average level). To reduce variance, a baseline function is often introduced, such as the state value function $V^\pi(s_t)$, to construct an advantage function:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{13}$$

Gradient update is simplified as:

$$\nabla_\theta J(\theta) = E_{\pi\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot A^\pi(s_t | a_t)\right] \tag{14}$$

The actor critic algorithm further combines' policy gradients with value function estimation to form a dual network architecture: Actor network (policy network): responsible for generating actions and adjusting policy parameters based on the advantage signals provided by critic. Critic network (value function network): evaluate the value of the current strategy, calculate the dominance function to guide actor updates (Elguea-Aguinaco et al., 2023).

Critic optimises the value function estimation by minimising the temporal differential error:

$$L(w) = E\left(R(s_t, a_t, s_{t+1}) + \gamma V_w(s_{t+1}) - V_w(s_t)\right)^2\Big] \tag{15}$$

Actor updates their strategy based on the advantages provided by critic:

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot A_w(s_t, a_t) \tag{16}$$

With the development of deep learning, traditional reinforcement learning algorithms have been combined with deep neural networks to form DRL. For example, deep Q-Network solves the representation problem of high-dimensional state spaces (such as image inputs) by fitting Q-value functions through convolutional neural networks; the strategy gradient method achieves direct optimisation of continuous actions (such as robotic arm torque control) by parameterising the strategy function through neural networks. These extensions significantly enhance the applicability of reinforcement learning in real-world scenarios, but also introduce new challenges such as training stability and sample efficiency.

The theoretical framework of reinforcement learning provides a universal framework for solving complex decision-making problems by formally defining interactive learning processes. Its core value lies in the ability to autonomously optimise strategies through environmental feedback without the need for pre-labelled data; by iterating discount factors and value functions, global optimisation of multi-step decision-making is achieved; it can be extended to heterogeneous task scenarios by combining deep learning, meta learning, and other technologies. Currently, the successful application of reinforcement learning in fields such as autonomous driving, energy management, and medical decision-making has validated the powerful potential of its theoretical methods. However, how to improve learning efficiency and ensure security in large-scale distributed environments remains an important direction for future research (Gronauer and Diepold, 2022).

3 GD optimisation algorithm based on dynamic resource collaboration

This chapter proposes a collaborative optimisation framework that integrates dynamic resource scheduling and adaptive GD, as shown in Figure 1. Aiming to solve the core problem of the imbalance between algorithm convergence efficiency and resource utilisation in large-scale data parallel training. This method achieves load balancing for gradient updates through a hierarchical batch sampling mechanism, while dynamically allocating computing resources through a RAES, forming a closed-loop feedback optimisation system between the algorithm layer and the resource layer. The following discussion will focus on three aspects: gradient update strategy, resource scheduling mechanism, and collaborative optimisation methods.

3.1 Hierarchical adaptive GD algorithm

HB-ASGD introduces a two-stage optimisation mechanism of dynamic batch sampling and asynchronous gradient aggregation. In the local computing stage, each working node dynamically adjusts the batch processing size 1 based on real-time computing load, and the adjustment rule is:

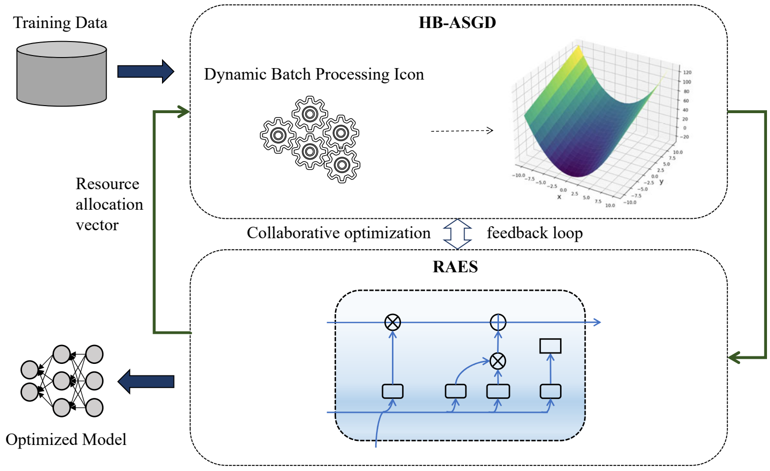
$$m_i^{(t)} = \left\lfloor m_{base} \cdot \frac{T_{avg}}{T_i^{(t-1)}} \right\rfloor \tag{17}$$

where m_{base} is the benchmark batch size, $T_i^{(t-1)}$ represents the computation time of node i in the previous iteration, and T_{avg} is the average cluster time. By dynamically expanding the batch processing capacity of faster nodes and reducing the computational load of slower nodes, the difference in computation time between nodes can be effectively balanced. In the gradient aggregation stage, a delay tolerant asynchronous update strategy is adopted. After receiving the gradient $\nabla J_i(\theta)$ from each node, the parameter server updates the global parameters by weighting the gradient contributions within the time window τ:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \cdot \sum_{k=0}^{\tau} \beta^k \nabla J_{i_{t-k}} \left(\theta^{(t-k)}\right) \tag{18}$$

where $\beta \in (0, 1)$ represents the delay gradient attenuation coefficient, which is used to reduce the impact of expired gradients on the current update. This mechanism reduces synchronous waiting time while suppressing gradient bias introduced by asynchronous updates.

Figure 1 Method framework diagram (see online version for colours)


Training Data
HB-ASGD
Dynamic Batch Processing Icon
Resource allocation vector
Collaborative optimization
feedback loop
RAES
Optimized Model

3.2 RAES model

To improve the utilisation of cluster resources, the RAES model predicts the computational requirements of iterative tasks based on LSTM network and drives the dynamic scaling of containerised resources (Zha et al., 2022). Firstly, a task computation predictor is constructed, taking historical resource usage sequence $\{(c_{t-\Delta}, m_{t-\Delta}), \ldots, (c_{t-1}, m_{t-1})\}$ as input (c_t represents CPU usage, m_t represents memory usage), and outputting the predicted computation $\hat{d}_t$ for the next time step through an LSTM network.

$$h_t = LSTM\left(W_h h_{t-1} + W_x \left[c_t, m_t\right] + b_h\right) \tag{19}$$

$$\hat{d}_t = \sigma\left(W_d h_t + b_d\right) \tag{20}$$

where h_t is the hidden state of LSTM, w_h, w_x, w_d are weight matrices, b_h, b_d are bias terms, and σ is the Sigmoid activation function. Based on the predicted results, the resource scheduler constructs a mixed integer programming model with the goal of minimising task completion time and resource idle costs:

$$\min \sum_{i=1}^{N} \left(\lambda_1 T_i^{exec} + \lambda_2 \sum_{r \in R} u_{i,r} \right) \tag{21}$$

where T_i^{exec} represents the execution time of task *i*, $u_{i,r}$ represents the usage of resource type r (such as CPU core, GPU memory) by task *i*, and λ_1 and λ_2 are trade-off coefficients. This model uses heuristic search to generate a dynamic allocation scheme for container resources.

3.3 Algorithm resource collaborative optimisation mechanism

HB-ASGD and RAES achieve cross layer collaboration through bidirectional feedback: HB-ASGD real-time calculates the gradient variance $Var(\nabla J_i)$ and update delay δ_i of each node, using them as input features for RAES, and prioritises allocating computing resources to high variance or high delay nodes to accelerate the convergence of critical gradient paths. RAES passes the current resource allocation vector u_t to the parameter server, and HB-ASGD adjusts the global learning rate α accordingly:

$$\min \sum_{i=1}^{N} \left(\lambda_1 T_i^{exec} + \lambda_2 \sum_{r \in R} u_{i,r} \right) \tag{22}$$

When GPU resources are sufficient, increase the learning rate to accelerate convergence, and when resources are tight, decrease the learning rate to avoid divergence. This collaborative mechanism is implemented through the Spark on Kubernetes platform, where Spark is responsible for distributed gradient computing and Kubernetes dynamically adjusts container resource quotas based on RAES instructions.

This method achieves joint optimisation of gradient update efficiency and resource utilisation through deep coupling between the algorithm layer and the resource layer.

4 Experiment and result analysis

4.1 Experimental setup and dataset

The experiment is based on a Spark on Kubernetes cluster environment, consisting of ten computing nodes (Intel Xeon Gold 6248R CPU, NVIDIA A100 GPU, 256 GB memory), with a network bandwidth of 100 Gbps.

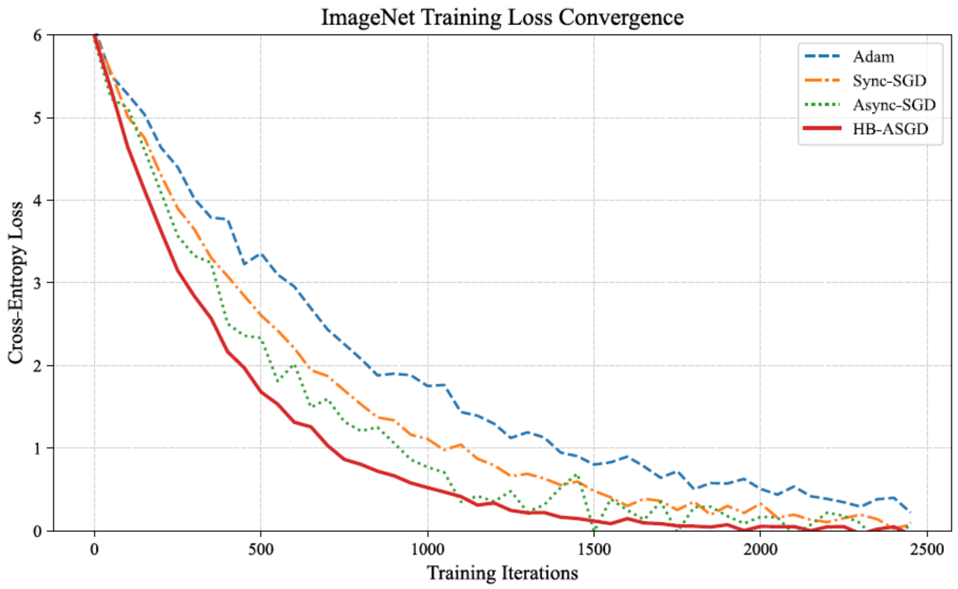
Two types of datasets were selected to validate the generalisation of the method: ImageNet-1K, which contains 1.2 million images and 1,000 categories, and the ResNet-50 model was trained. Industrial grade user behaviour data, consisting of 210 million multimodal interaction logs from an e-commerce platform (covering behaviours such as clicks, add ons, payments, etc.), is used to construct a DeepFM model for click through rate prediction. This dataset has high-dimensional sparse features (over 5,000 dimensions) and real-time requirements, which can verify the practicality of the method in industrial scale scenarios. The baseline comparison includes Adam optimiser (Reyad et al., 2023), sync SGD (Hu et al., 2021), asynchronous SGD (Yu et al., 2023), Kubeflow (Zhang et al., 2021), Optimus (Chiang et al., 2023) scheduling strategies.

4.2 Convergence efficiency verification

This experiment runs HB-ASGD and baseline methods separately in ImageNet and CTR tasks, and records the number of iterations and time required to achieve the target loss (ImageNet cross entropy loss $\leqslant$ 2.0, CTR LogLoss $\leqslant$ 0.35). The HB-ASGD parameters are set to a baseline batch size of 512, a delay gradient attenuation coefficient of 0.8, and an initial learning rate of 0.1.

As shown in Figure 2, HB-ASGD can converge in ImageNet task with only 1,420 iterations, a decrease of 36.9% compared to Adam (2,250 iterations) and a decrease of 27.2% compared to synchronous SGD (1,950 iterations). Its advantage lies in the dynamic batch processing strategy: nodes with fast computing speed are batch expanded to 768 (1.5 times the baseline), while slow nodes are reduced to 384. Load balancing reduces the average iteration time of the cluster by 18%. Meanwhile, the delay gradient attenuation coefficient ($\beta = 0.8$) suppresses the noise interference of expired gradients by weighted aggregation of historical gradients. In the CTR task, HB-ASGD took 6.2 hours, significantly better than asynchronous SGD's 10.5 hours, indicating its efficient adaptability to sparse data.

Figure 2 ImageNet task loss convergence curve (see online version for colours)



4.3 Resource utilisation optimisation

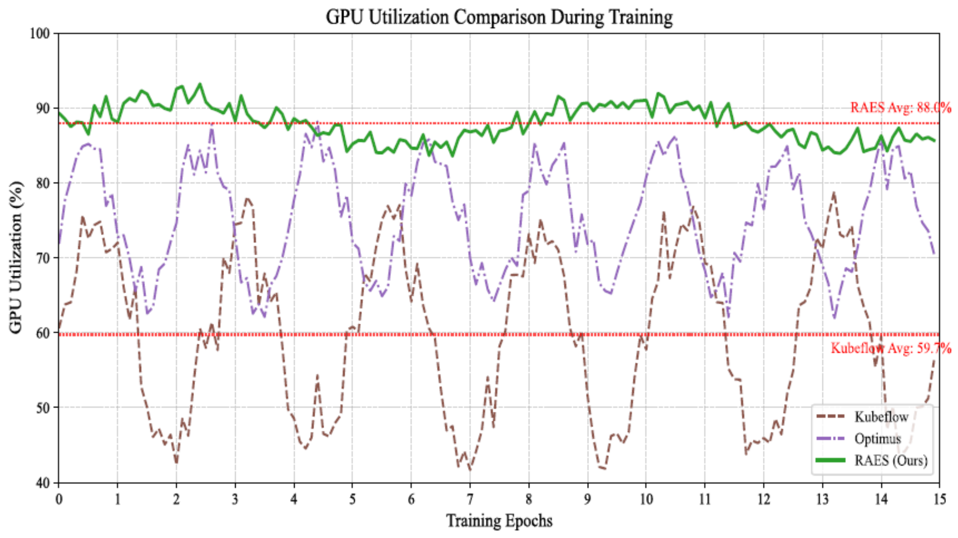
Static resource allocation (such as Kubeflow) can easily lead to resource idleness, while periodic scheduling (such as Optimus) lags behind in responding to sudden loads. The purpose of this experiment is to verify whether the RAES model can improve resource utilisation through LSTM prediction and dynamic scaling. This experiment deployed RAES and baseline scheduler in the ImageNet task to monitor GPU utilisation and CPU memory fragmentation rate. The LSTM predictor of RAES takes the historical resource usage sequence (CPU, memory, GPU memory) from 10 rounds and outputs the next round of computation requirements; The elastic scaling strategy is based on a mixed integer programming model, with the goal of minimising task completion time and resource idle costs.

As shown in Table 1, RAES resulted in an average GPU utilisation rate of 89.4%, an increase of 28.2 percentage points compared to Kubeflow (61.2%), and a decrease in fragmentation rate from 34.7% to 6.5%. As shown in Figure 3, during the mid training period (500–1,200 iterations), the accuracy of the LSTM predictor exceeded 92%. RAES dynamically adjusted the GPU memory allocation (from 16GB to 48GB) to stabilise the utilisation rate at over 85%; however, Optimus has a fixed five-minute adjustment cycle, resulting in response delays when the load suddenly increases, causing GPU utilisation to fluctuate between 60% and 80%. In addition, RAES reduces the execution time of critical path tasks by 23% through priority scheduling, such as prioritising GPU resources for high gradient variance nodes.

Table 1 Comparison of resource utilisation rates

Index	*Kubeflow*	*Optimus*	*RAES*
GPU utilisation rate (%)	61.2	75.3	89.4
Fragmentation rate (%)	34.7	18.9	6.5

Figure 3 GPU utilisation rate (see online version for colours)



4.4 End to end efficiency and collaborative gain verification

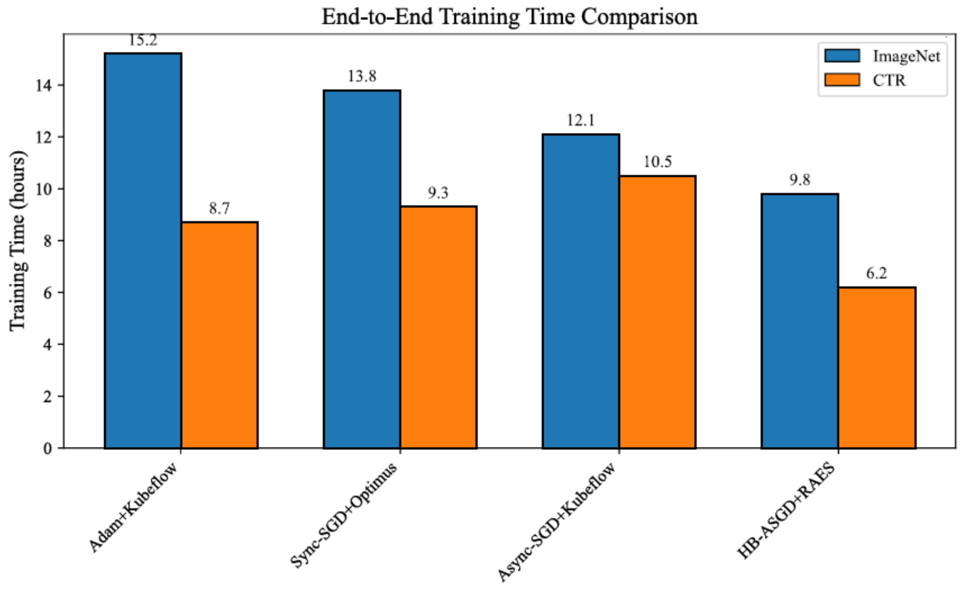
A single optimisation algorithm or resource scheduling may lead to performance bottlenecks due to interlayer coupling issues. The purpose of this experiment is to verify whether cross layer collaboration mechanisms can significantly improve end-to-end efficiency. This section analyses the synergistic effect contribution by comparing the total training time of the complete framework (HB-ASGD+RAES) with the baseline combination in ImageNet and CTR tasks.

As shown in Figure 4 and Table 2, the complete framework takes a total of 9.8 hours in the ImageNet task, which is 35.5% less than Adam+Kubeflow (15.2 hours); took 6.2 hours in the CTR task, a decrease of 40.9% compared to asynchronous SGD+Optimus (10.5 hours). The collaborative mechanism achieves efficiency doubling through bidirectional feedback: when RAES detects that the GPU utilisation rate of a node is below 70%, it triggers HB-ASGD to dynamically expand its batch processing volume (such as increasing from 512 to 768), and at the same time, the global learning rate is adjusted from 0.1 to 0.15 with resource adequacy to avoid iteration stagnation caused by insufficient resources. Experiments have shown that when HB-ASGD or RAES are used separately, the ImageNet task takes 12.3 hours and 11.9 hours, respectively, while the collaborative framework is further compressed to 9.8 hours, demonstrating the necessity of cross layer optimisation.

Table 2 End to end training time

Method	*ImageNet*	*CTR*
Adam + Kubeflow	15.2	8.7
Sync-SGD + Optimus	13.8	9.3
Async-SGD + Kubeflow	12.1	10.5
HB-ASGD + RAES	9.8	6.2

Figure 4 Comparison of end-to-end training time (see online version for colours)



Experiments have shown that the collaborative framework of HB-ASGD and RAES significantly outperforms traditional methods in terms of convergence speed, resource utilisation, and end-to-end efficiency. Its core advantage lies in: HB-ASGD compensates for node heterogeneity through adaptive batch processing, reducing slow node blocking; RAES utilises LSTM to capture load temporal characteristics and achieve fine-grained elastic scaling; resource allocation and algorithm updates form a closed-loop optimisation to avoid local optima.

5 Conclusions

This article proposes a collaborative framework that integrates dynamic GD optimisation and elastic resource scheduling to address the core issue of the imbalance between algorithm efficiency and resource utilisation in distributed machine learning tasks in the big data environment. By designing HB-ASGD and RAES, deep coupling optimisation between the algorithm layer and the resource layer has been achieved. HB-ASGD effectively alleviates the slow node blocking problem in heterogeneous clusters by dynamically adjusting the node batch processing scale and delay gradient weighted aggregation. Future work will revolve around exploring collaborative optimisation strategies for heterogeneous hardware (GPU/TPU hybrid clusters); combining the federated learning framework, study the balance mechanism between cross domain data privacy protection and training efficiency; develop a global optimisation theoretical model for adaptive load sensing to further enhance the generalisation ability of the method. This study provides a new technological path for distributed machine learning training in the big data environment, and its engineering implementation and theoretical achievements have important reference value for promoting the development of efficient AI computing infrastructure.

Declarations

All authors declare that they have no conflicts of interest.

References

Ahn, K., Cheng, X., Daneshmand, H. and Sra, S. (2023) 'Transformers learn to implement preconditioned gradient descent for in-context learning', *Advances in Neural Information Processing Systems*, Vol. 36, No. 3, pp.45614–45650.

Cao, Y. and Su, S. (2023) 'Fractional gradient descent algorithms for systems with outliers: a matrix fractional derivative or a scalar fractional derivative', *Chaos, Solitons & Fractals*, Vol. 174, No. 12, p. 113881.

Chaudhary, N.I., Raja, M.A.Z., Khan, Z.A., Mehmood, A. and Shah, S.M. (2022) 'Design of fractional hierarchical gradient descent algorithm for parameter estimation of nonlinear control autoregressive systems', *Chaos, Solitons & Fractals*, Vol. 157, No. 19, p.111913.

Chen, C., Lee, B., Li, N-N., Chae, M., Wang, D., Wang, Q-H. and Lee, B. (2021) 'Multi-depth hologram generation using stochastic gradient descent algorithm with complex loss function', *Optics Express*, Vol. 29, No. 10, pp.15089–15103.

Chiang, M-C., Zhang, L-W., Chou, Y-M. and Chou, J. (2023) 'Dynamic resource management for machine learning pipeline workloads', *SN Computer Science*, Vol. 4, No. 5, p.665.

Dakić, V., Kovač, M. and Slovinac, J. (2024) 'Evolving high-performance computing data centers with Kubernetes, performance analysis, and dynamic workload placement based on machine learning scheduling', *Electronics*, Vol. 13, No. 13, p.2651.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M.A., Senior, A., Tucker, P. and Yang, K. (2012) 'Large scale distributed deep networks', *Advances in Neural Information Processing Systems*, Vol. 25, No. 2, p.876.

Deng, L., Wang, Z., Sun, H., Li, B. and Yang, X. (2023) 'A deep reinforcement learning-based optimization method for long-running applications container deployment', *International Journal of Computers Communications & Control*, Vol. 18, No. 4, p.776.

Elguea-Aguinaco, Í., Serrano-Muñoz, A., Chrysostomou, D., Inziarte-Hidalgo, I., Bøgh, S. and Arana-Arexolaleiba, N. (2023) 'A review on reinforcement learning for contact-rich robotic manipulation tasks', *Robotics and Computer-Integrated Manufacturing*, Vol. 81, No. 32, p.102517.

Gronauer, S. and Diepold, K. (2022) 'Multi-agent deep reinforcement learning: a survey', *Artificial Intelligence Review*, Vol. 55, No. 2, pp.895–943.

Haji, S.H. and Abdulazeez, A.M. (2021) 'Comparison of optimization techniques based on gradient descent algorithm: a review', *PalArch's Journal of Archaeology of Egypt/Egyptology*, Vol. 18, No. 4, pp.2715–2743.

Hu, S., Chen, X., Ni, W., Hossain, E. and Wang, X. (2021) 'Distributed machine learning for wireless communication networks: techniques, architectures, and applications', *IEEE Communications Surveys & Tutorials*, Vol. 23, No. 3, pp.1458–1493.

Li, X., Liu, Y. and Liu, Z. (2023) 'Physics-informed neural network based on a new adaptive gradient descent algorithm for solving partial differential equations of flow problems', *Physics of Fluids*, Vol. 35, No. 6, p.9887.

Matsuo, Y., LeCun, Y., Sahani, M., Precup, D., Silver, D., Sugiyama, M., Uchibe, E. and Morimoto, J. (2022) 'Deep learning, reinforcement learning, and world models', *Neural Networks*, Vol. 152, No. 2, pp.267–275.

Reyad, M., Sarhan, A.M. and Arafa, M. (2023) 'A modified Adam algorithm for deep neural network optimization', *Neural Computing and Applications*, Vol. 35, No. 23, pp.17095–17112.

Rivas, J.M., Gutiérrez, J.J., Guasque, A. and Balbastre, P. (2024) 'Gradient descent algorithm for the optimization of fixed priorities in real-time systems', *Journal of Systems Architecture*, Vol. 153, No. 8, p.103198.

Shakya, A.K., Pillai, G. and Chakrabarty, S. (2023) 'Reinforcement learning algorithms: a brief survey', *Expert Systems with Applications*, Vol. 231, No. 9, p.120495.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. and Yao, S. (2023) 'Reflexion: language agents with verbal reinforcement learning', *Advances in Neural Information Processing Systems*, Vol. 36, No. 8, pp.8634–8652.

Tian, Y., Zhang, Y. and Zhang, H. (2023) 'Recent advances in stochastic gradient descent in deep learning', *Mathematics*, Vol. 11, No. 3, p.682.

Yi, D., Ahn, J. and Ji, S. (2020) 'An effective optimization method for machine learning based on ADAM', *Applied Sciences*, Vol. 10, No. 3, p.1073.

Yu, E., Dong, D. and Liao, X. (2023) 'Communication optimization algorithms for distributed deep learning systems: a survey', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 34, No. 12, pp.3294–3308.

Zha, W., Liu, Y., Wan, Y., Luo, R., Li, D., Yang, S. and Xu, Y. (2022) 'Forecasting monthly gas field production based on the CNN-LSTM model', *Energy*, Vol. 260, No. 2, p.124889.

Zhang, S., Choromanska, A.E. and LeCun, Y. (2015) 'Deep learning with elastic averaging SGD', *Advances in Neural Information Processing Systems*, Vol. 28, No. 7, pp.223–234.

Zhang, X., Li, L., Wang, Y., Chen, E. and Shou, L. (2021) 'Zeus: Improving resource efficiency via workload collocation for massive kubernetes clusters', *IEEE Access*, Vol. 9, No. 4, pp.105192–105204.