Eng Int Syst (2021) 5: 279–289
© 2021 CRL Publishing Ltd

Engineering
Intelligent Systems

Load Balance Optimization of Distributed Massive Database Information Acceptance and Processing in the Internet of Things Scenario

Jialiang Wang*

Department of Information and Engineering, Sichuan Tourism University, ChengDu 610072, China

In order to make full use of the resources of each service node in a cluster to improve the overall performance of the cluster, it is necessary to select the appropriate load balancing technology and efficient load balancing algorithm to allocate client access requests. Focusing on the load imbalance problem of Spark, this paper proposes an adaptive task scheduling strategy based on Spark cluster to achieve the load balancing optimization of Spark clusters. This strategy uses the heuristic algorithm of the ant colony simulated annealing fusion algorithm to optimize the task scheduling strategy of the Spark cluster according to the node's current load and computing resources. This achieves the appropriate allocation of tasks for the purpose of load balancing optimization, thereby improving the cluster's task-processing efficiency. In order to achieve dynamic load balancing on the Reduce side, a dynamic and lightweight division strategy is adopted. This strategy combines the load information for the dynamic design of the sampling scale and the lightweight design of the sampling method, and combines the sampling data and node performance to determine the number of Reducers. In addition, a division strategy is formulated based on the partition analysis of the sampling results and load information. After experimentation and analysis, it is concluded that the optimization technology has significantly improved the parallel computing performance of the Map Reduce cluster.

Keywords: load balancing; ant colony-simulated annealing algorithm; distributed massive database; Internet of Things (IoT) scenario

1. INTRODUCTION

There are differences in the structure and performance of a single server in a server cluster. Client access is random and bursty. The source and content of access requests are fragmented and diverse, causing some servers to bear either too little or too much load [1–3]. Unbalanced load distribution will seriously reduce the overall performance of the server cluster system and cannot convey the original meaning of the cluster. To solve this kind of cluster server imbalance problem, it is necessary to ensure that the load level of each node is consistent at all times [4]. By means of a load balancing strategy, the system distributes the received requests to the available nodes, avoiding blind allocation that leads to uneven availability of nodes and causes the load of the cluster system to tilt. Therefore, load balancing technology is vital to the cluster system and can improve the performance of the cluster [5,6].

A key-value storage database is one in which data is stored in memory and disk in the form of keys and values [7]. Key-value storage has a commonly-used simple data model: a

*E-Mail: wang20202021ei@163.com

map/dictionary that allows clients to request and push values through keys. A well-known, key-value store database is Amazon Dynamo. It is used for multiple purposes in Amazon and other databases. As one of the earliest No SQL products, it has had a great impact on subsequent No SQL databases. Redis is another well-known No SQL database developed by Salvatore Sanfilippo that offers key-value storage [8]. It provides a richer data structure comprised of list, set, hash, string, etc. and operations on these data structures. Redis loads the entire database system into memory, so it has high performance, rich data types, and all operations are automatic operations [9]. However, because most of Redis's data is in memory, its security is poor. Secondly, since the price of memory is relatively low compared with that of hard disks, the cost-effectiveness of scalability is not high. However, Ippotito suggests that a caching layer can be integrated in Redis [10]. The document database puts several pairs of key-values into a particular structure as a transition from a simple key-value storage to a more complex storage database. Apache Couch DB and Mongo DB are typical of such databases. Couch DB is written in Erlang, which is an improvement on Lotus Notes because its developer worked at IBM and referred to the former's technology. The notable feature of Couch DB is a document database that contains a number of documents with no fixed structure in a flat address space that can be accessed through a RESTful HTTP interface [11]. It allows multiple parallel versions of the same document, and the database automatically detects and merges conflicts. According to its submission system and the way it manipulates files, Couch DB can be considered to meet ACID attributes. The most famous application of Couch DB is ubuntu one, a cloud storage and replication service under the Ubuntu Linux environment. The BBC web application platform, several Facebook applications, and Apache Lucene also use Couch DB as their server.

A load balancing algorithm is the key to achieving cluster system load balancing, and there have been many advancements in this field [12–14]. Load balancing algorithms are divided into two categories: static load balancing algorithms and dynamic load balancing algorithms. Static load balancing algorithms include random algorithms, polling algorithms, and weighted polling algorithms. This type of algorithm saves the task allocation ratio of each server as a system parameter in the load balancer, regardless of real-time changes in server load. The static load balancing algorithm is an open-loop load adjustment method, but it cannot meet the performance requirements of the new generation of server clusters. Dynamic load balancing algorithms include the minimum connection number algorithm, weighted minimum connection number algorithm and dynamic weight round-robin algorithm [15]. The load balancer will periodically receive the operating status of each server, and use this as a basis for selecting the target node to process the request. Under normal circumstances, compared with static algorithms, the performance of dynamic load balancing algorithms is 30%–40% higher, but dynamic load balancing algorithms require additional data processing and transmission [16]. With the improvement of computer performance, the system resources consumed by the dynamic load balancing algorithm will basically not affect the performance of the load balancer [17]. Because of its better flexibility and efficiency, many research results have been produced by the dynamic load balancing algorithm. Scholars in the field have grouped cluster servers in the cloud computing environment according to their hardware, and grouped nodes with similar performance into one group, thereby simplifying calculations and reducing problems such as delays caused by high concurrency [18]. Researchers use the concept of entropy to divide the cluster into several uniform domains to construct a hierarchical cluster system, which prevents server communication delays caused by excessive clusters [19]. Researchers have set the distribution weights as the calculation indicators, and optimized them with the critical acceleration regression algorithm [20].

To address the problem of load imbalance in Spark, this paper proposes an improved strategy which is based on an optimized task allocation method to achieve load balancing. Because the ant colony algorithm can easily fall into a local optimal solution, this paper proposes ant colony simulated annealing fusion algorithm. The fusion algorithm introduces the notion of random disturbance in the simulated annealing algorithm into the ant colony algorithm to make up for the shortcomings of the ant colony algorithm and shorten the iteration time while avoiding the local optimum. Subsequently, the ant colony simulated annealing fusion algorithm was used to optimize the task scheduling strategy, and the task scheduling improvement strategy was proposed, and the load balancing optimization of the Spark cluster was realized by means of the task scheduling improvement strategy. Optimization measures have been taken for the three objects (intermediate data set, reducer number, and division strategy) that can affect load balancing on the Reduce side. An appropriate experimental environment is established, and the optimized Map Reduce is tested in a cluster environment with experimental data. The results show that a good optimization effect has been achieved.

The rest of this paper is organized as follows. Section 2 analyzes the architecture of the distributed massive database system in the IoT scenario. In Section 3, a distributed mass database load balancing optimization algorithm based on ant colony-simulated annealing is designed. In Section 4, experiments and analysis are conducted. Section 5 concludes the paper with a summary of this study.

2. ARCHITECTURE OF A DISTRIBUTED MASSIVE DATABASE SYSTEM IN THE IOT SCENARIO

2.1 Architecture of a Distributed Massive Database System

The distributed mass database system in the IoT scenario is an organic combination of distribution and unification, and is an extension of the centralized database system. The centralized database system divides the database into an external model and an internal model. However, the model structure of a distributed massive database in the IoT scenario is more complicated because its structure can usually be divided into global external mode, global conceptual mode, sharding

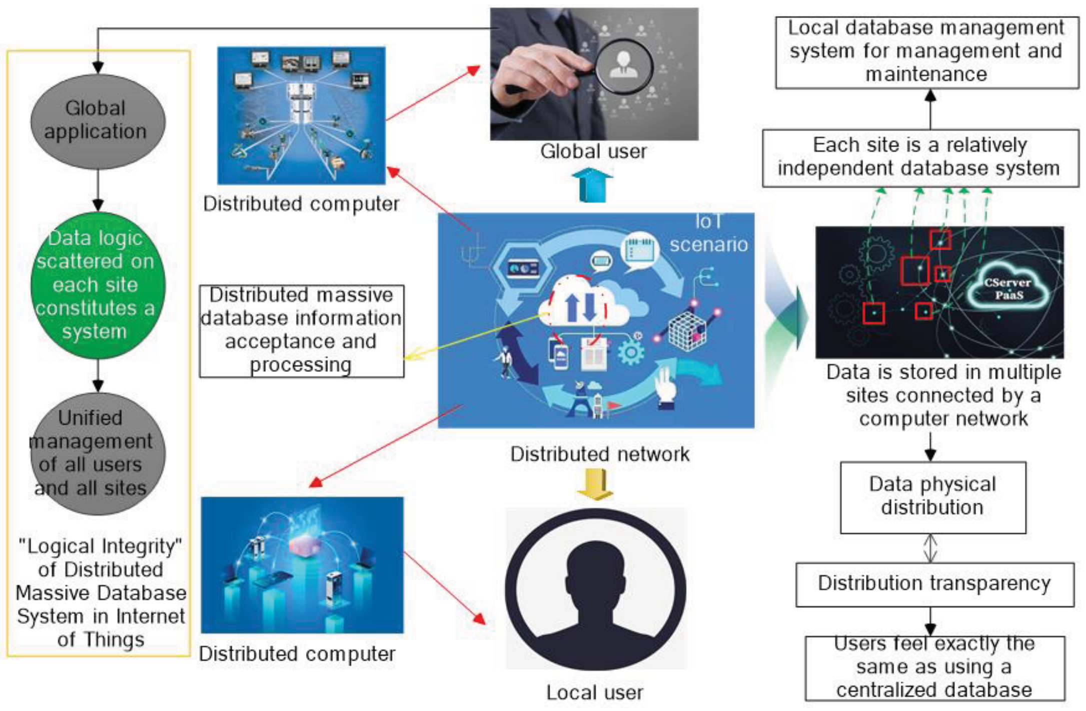
Global application
Data logic scattered on each site constitutes a system
Unified management of all users and all sites
"Logical Integrity" of Distributed Massive Database System in Internet of Things
Distributed computer
Global user
Local database management system for management and maintenance
Each site is a relatively independent database system
IoT scenario
Distributed massive database information acceptance and processing
CServer PaaS
Data is stored in multiple sites connected by a computer network
Distributed network
Data physical distribution
Distribution transparency
Distributed computer
Local user
Users feel exactly the same as using a centralized database

Figure 1 Distributed Massive Database Management System Architecture in the IoT Scenario.

mode, distributed mode, local conceptual mode, and local internal mode: a total of six modes. The conversion between the various modes is similar to that of a centralized database system, which is implemented by multiple images provided by the global and local database management systems.

The centralized database consists of a database, a database management system, and a database administrator. The distributed massive database system architecture in the IoT is also expanded on the basis of the centralized database system. This distributed massive database system divides the database, the database management system, and the database administrator into local and global. Of course, the data dictionary in the database is also indispensable. In the IoT, the distributed mass database system divides the data dictionary into a local data dictionary and a global data dictionary. In addition, according to the requested data, the users of the database are also divided into local users and global users. Figure 1 shows the architecture of the distributed massive database system in the IoT scenario.

In the IoT scenario, the data in the distributed massive database system is scattered and stored on multiple sites connected by a computer network, and such scattered users cannot feel it. This feature is called the physical distribution of data in a distributed massive database system in the IoT scenario, also known as distributed transparency. Distribution transparency means that when using the database, users do not need to care about where the required data is stored, how the data is distributed, or which system each site server uses, and which database management system the database uses. When a user uses a data system, the distributed mass database system of the IoT will process the data according to the required distribution. This process the user feels and uses a centralized database is exactly the same, that is, completely transparent.

The data in the distributed massive database system of the IoT is scattered on various sites in the network, and each site is a relatively independent database system that is managed and maintained by the local database management system. However, the data on each site can be used by users of other sites in addition to local users, that is, global applications. The distributed mass database system of the IoT constitutes a system of data that is scattered on each site. All users can use this distributed mass database system in the IoT. All users and all sites are distributed in the IoT scenario. The unified management of the distributed massive database management system is the “logical integrity” of the distributed massive database system of the IoT. Figure 2 depicts the distributed massive database model in the IoT scenario.

2.2 Distributed Massive Database System Functions in the IoT Scenario

Distributed massive database systems in the IoT scenario are divided into heterogeneous DDBS and isomorphic DDBS or into centralized DDBS and decentralized DDBS. Therefore, in addition to managing local databases, they also contain many more complicated functions. However, their basic function is to realize the establishment, query, update, replication, and maintenance of distributed massive databases in the IoT scenario. It also includes data distribution, query optimization, centralized control, data consistency, concurrency control, and update synchronization and globalization. The distributed mass database management system in the IoT scenario includes four basic functional modules:

(1) Query processing module

The query processing module is responsible for query analysis and optimization processing. In distributed

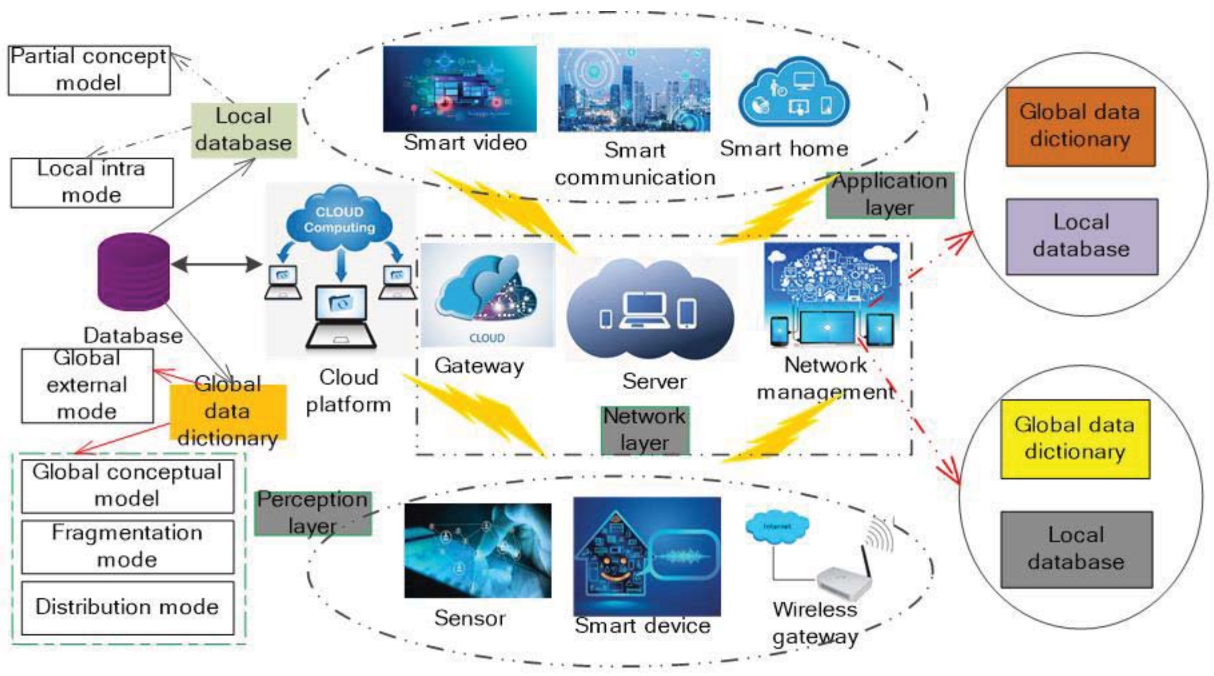

Figure 2 Mode Structure of Distributed Massive Database System in IoT Scenario.

data query processing, data communication is often generated, especially in wide area networks, where the communication cost is very high. Therefore, query optimization is very important in the processing of distributed massive databases in the IoT scenario, which is also the core issue of this paper.

(2) Integrity processing module

The integrity processing module is responsible for maintaining the integrity and consistency of the database. Due to the redundancy of the data, the module and the query processing module will decide to use the data copy of that site. In addition, this module is responsible for maintaining the integrity of the database and improving the concurrency control mechanism.

(3) Scheduling processing module

The scheduling processing module is responsible for the coordination and scheduling of data transmission. The module issues commands to the site where the data is requested, and the database management system of the site executes these commands. In order to complete the distributed query, the necessary data transmission between the sites is required to obtain the final result.

(4) Reliability processing module

The reliability processing module is responsible for system failure recovery and data consistency. This module continuously monitors the operating status of each site of the system. Once a site fails, it will be drained from the system, and the data on this site can be obtained from other sites. After the fault is repaired, the site can again be included in the system to maintain the consistency of the database.

3. LOAD BALANCING OPTIMIZATION ALGORITHM FOR DISTRIBUTED MASSIVE DATABASES BASED ON ANT COLONY-SIMULATED ANNEALING

3.1 Analysis of Load Balancing Optimization Strategy

Since the task scheduling follows the data locality priority scheduling method, the actual situation of each computing node is ignored, this causes load imbalance. This paper optimizes Spark cluster load balancing from the perspective of task scheduling, introduces heuristic algorithms to improve task scheduling strategies, and appropriately allocates tasks to be executed to the computing nodes in the Spark cluster to achieve task load balancing among computing nodes.

The task scheduling problem is described as follows: using the task scheduling scheme, n mutually independent tasks are assigned to m computing nodes for execution, as shown in Figure 3. The goal of the distribution is that after these tasks are allocated to the execution nodes, the task load of each node is relatively balanced. By balancing the task load of each node in the calculation process, the load balance of the entire cluster is realized.

3.2 Ant Colony Algorithm and Simulated Annealing Algorithm

1) Ant Colony Algorithm

The ant colony algorithm was originally used to solve the TSP problem. In the calculation, the different walking paths of different ants represent multiple feasible solutions to the problem, and these path sets constitute

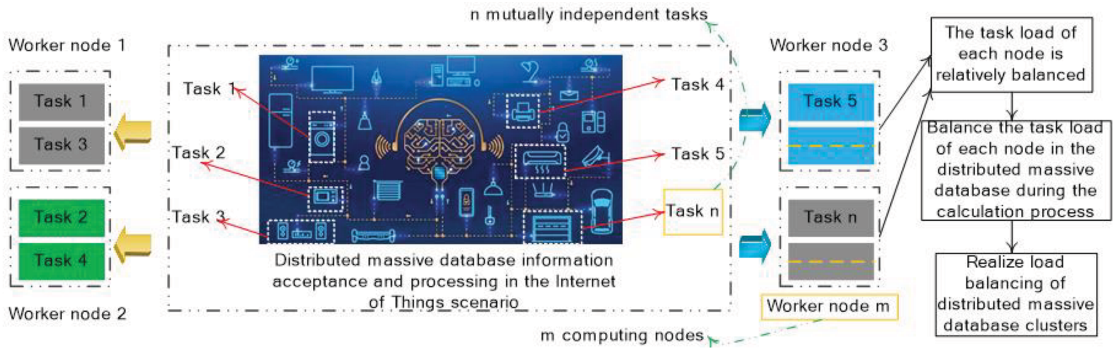
n mutually independent tasks
Worker node 1
Task 1
Task 3
Task 2
Task 4
Worker node 2
Task 1
Task 2
Task 3
Task 4
Task 5
Task n
Distributed massive database information acceptance and processing in the Internet of Things scenario
m computing nodes
Worker node 3
Task 5
Task n
Worker node m
The task load of each node is relatively balanced
Balance the task load of each node in the distributed massive database during the calculation process
Realize load balancing of distributed massive database clusters

Figure 3 Description of Task Allocation Problem.

the feasible solution set for the problem. The shorter the path, the higher the concentration, the more ants on the path. After multiple iterations, this positive feedback effect will make the ants concentrate on the optimal path, and the optimal solution of the problem is obtained at this time.

Assuming that the number of cities is n, the number of ants in the ant colony is m, the distance between cities is denoted as d_{ij}, and the pheromone concentration of the path between the two cities at time t is represented by $\mathrm{d}_{ij}(t)$, and the initial pheromone concentration on the path is d_0. The number of ants is k, and the probability of ant k transferring from city i to city j at time t is:

$$P_{ij}^{K}(t) = \begin{cases} \frac{\delta_{ij}(t)\cdot\theta_{ij}(t)\cdot\beta}{\sum_{s\in\xi}\delta_{ij}(t)\cdot\theta_{ij}(t)\cdot\beta} & S \in \xi \\ 0 & S \notin \xi \end{cases} \quad (1)$$

ξ represents the next set of cities for ants to choose, α is the pheromone heuristic factor, β is the expected heuristic factor, and θ is the heuristic function.

In the ant colony algorithm, in order to avoid the situation where the enlightening information has been covered, the algorithm simulates the pheromone volatilization mechanism in real life and introduces the pheromone update idea, that is, the pheromone with fewer selected paths will decrease, which makes most ants not select the path again. When all the ants complete a week cycle, the pheromone update is:

$$\delta_{ij}(t+1) = \Delta\delta_{ij} + (1-p)\cdot\delta_{ij(t)} \quad 0 < p \leq 1 \quad (2)$$

$\Delta\delta_{ij}$ is the total concentration of pheromone increase on the path (i, j), and its calculation method is:

$$\Delta\delta_{ij} = \sum_{k=0}^{m-1}\Delta\delta_{ij}^{k} \quad (3)$$

2) Simulated annealing algorithm

The simulated annealing algorithm originated from the metal cooling process during smelting, where the metal is first heated, which causes drastic changes to the internal particles, and then the internal particles are cooled to stabilize them. When this idea is applied to the optimal solution problem, the objective function is used to simulate the degree of change of the particles, and the parameters are used to represent the temperature, which is the main idea of the simulated annealing algorithm.

The mathematical model of the simulated annealing algorithm is: when the initial temperature is t, the objective function value is calculated for the initial solution i and the randomly generated new solution j. If the objective function value of the new solution j is better than i, you replace i with j, and perform this process on j. If the objective function value of the new solution j is worse than i, the solution is replaced with a certain probability, and the probability decreases as the temperature parameter value t decreases. After multiple iterations the above process, the final algorithm will obtain a stable solution or the temperature parameter value is reduced to a pre-set threshold. The solution at this time is the approximate optimal solution obtained by the simulated annealing algorithm.

In the simulated annealing algorithm, it is very important to accept the new solution formed after the random perturbation of the initial solution. After the simulated annealing algorithm obtains the initial solution, the current solution is exchanged randomly through permutation rules. If the fitness function value of the new solution is lower than the original solution, the new solution is accepted; otherwise, the new solution is accepted according to the probability of being accepted.

Simulated annealing algorithm has a better effect in searching complex regions, because its solving process makes hasssssss the characteristics of parallelism. In addition, since the search process of the simulated annealing algorithm is random, the flexibility of the algorithm is enhanced, so that the algorithm can better search for the global optimal solution.

3.3 Load Balancing Optimization Based on Ant Colony-Simulated Annealing

1) Ant colony-simulated annealing fusion algorithm

Since the ant colony algorithm can easily fall into a local optimal solution, in order to solve the problems of the ant colony algorithm, this paper introduces the random disturbance principle of the simulated annealing algorithm. When the ant colony algorithm completes a single population calculation and produces the optimal solution for the current sub-group, we perform local

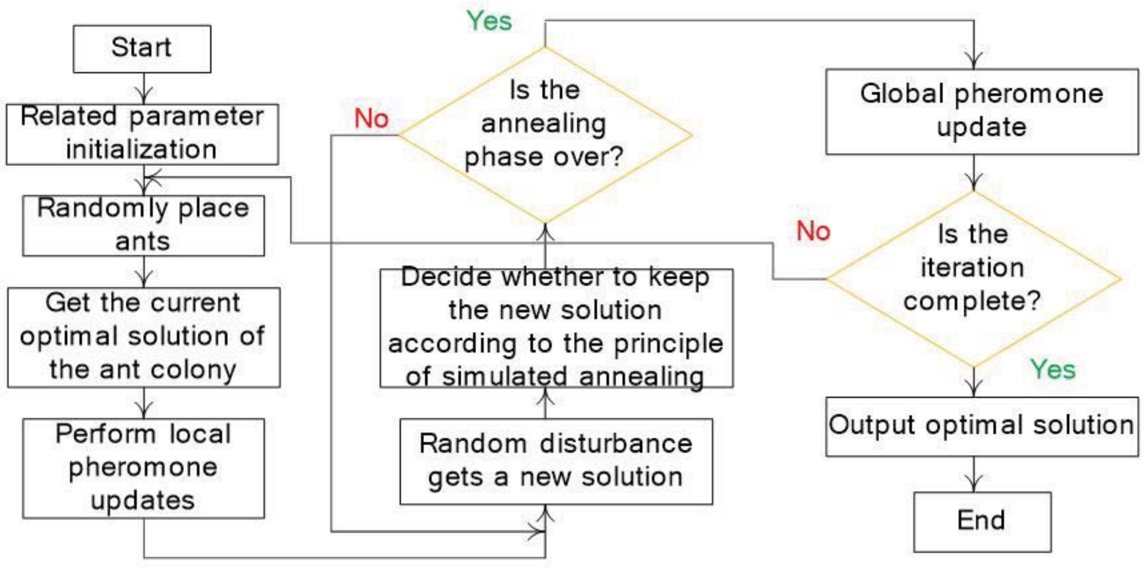
Start
Related parameter initialization
Randomly place ants
Get the current optimal solution of the ant colony
Perform local pheromone updates
Yes
No
Is the annealing phase over?
Decide whether to keep the new solution according to the principle of simulated annealing
Random disturbance gets a new solution
Global pheromone update
No
Is the iteration complete?
Yes
Output optimal solution
End

Figure 4 Algorithm Flow Chart.

random perturbation on the solution to produce a new solution, and calculate the fitness function value of the new solution at the same time. If the new solution is better than the solution prior to the perturbation, we accept the new solution as the optimal solution for the current subgroup, and perform the global enhancement of excellent path pheromone to speed up the overall convergence speed of the algorithm, while preventing the ant colony algorithm from falling into the local optimal solution. The flow of the fusion algorithm is shown in Figure 4.

2) Load evaluation index

To solve the load balancing optimization problem, we use the task node CPU usage C_m and the node task volume as the standard to measure the current node load. The calculation method is:

$$Load_i = M \cdot C_{m_i} + N \cdot task\,point_i \Big/ \sum_{i=0}^{m-1} taskpoint_i \quad (4)$$

$Load_i$ represents the load condition measurement index of the i-th node, $taskpoint_i$ represents the number of tasks allocated to the node, and M and N are adjustable parameters.

After all tasks have been allocated to the nodes, the expected value H of the node to achieve load balancing is calculated with:

$$H = \frac{1}{m} \sum_{i=0}^{m-1} Load_i \quad (5)$$

The calculation method for node load deviation error is as follows:

$$Error = \frac{1}{m} \left[\sum_{i=0}^{m-1} (H - Load_i)^2 \right]^{\frac{1}{2}} \quad (6)$$

3) Pheromone update mechanism

This study uses a combination of local and global updates to update the pheromone of each node; that is, first, the node pheromone of each ant is updated locally, and after all ants have completed a single optimization, high-quality information is added to the better node, thereby achieving global update and accelerating the convergence speed.

Given the load balancing problem to be solved in this paper, the larger the node load, the higher the current resource occupancy rate of the node, the less available resources, and the weaker computing power of the node. Therefore, the calculation method of the ant k pheromone increment is as follows:

$$\Delta\delta_{ij}^{k} = Q/Load_i \quad (7)$$

After the ant completes a one-week cycle, the solution is randomly disturbed, and a decision is made whether to retain the new solution according to the Metropolis criterion, and after the end of the current iteration, additional pheromone is added to the optimal solution.

4) Ant colony-simulated annealing task allocation strategy

This paper applies the fusion algorithm of the ant colony and simulated annealing algorithm to the optimization of the Spark strategy for task allocation. The steps are as follows:

Step 1: The user submits a Spark job to the cluster and initializes the fusion algorithm parameter values.

Step 2: The cluster resource manager obtains the specific situation of each node.

Step 3: After obtaining the current operating status, health status and load information of each node, the path is selected and the local pheromone is updated.

Step 4: After the ant colony completes the calculation, the current local optimal solution is perturbed, and a decision is made whether to form a new solution according to the Metropolis criterion.

Step 5: After completing a single round of calculation, an additional pheromone is included in the high-quality solution to complete the global update.

Step 6: When the algorithm reaches the termination criterion, the global optimal position is output, and the task allocation sequence corresponding to the solution is the optimal scheduling plan for this time; otherwise, iterative calculation continues.

Step 7: The cluster allocates tasks according to the obtained task allocation plan.

4. EXPERIMENT AND ANALYSIS

4.1 Description of the Specific Experiment Scheme

Based on a series of processes for load balancing optimization, this part of the experiment consists of two phases: the sampling optimization test experiment and the partition strategy optimization experiment.

The prerequisite for the optimization of the division strategy is the ability to obtain more accurate sampling results. The sampling accuracy has a greater impact on the optimization of the division strategy. The experiment is carried out to evaluate the sampling method proposed in this paper and compare it with approximate sampling and general random sampling. Through the analysis of the sample S and the corresponding sampling of the data set under different slope conditions, the sampling results are compared, and the sampling accuracy is evaluated by the root mean square error RMSE of the real data.

In addition, the optimization experiment conducted to evaluate the division strategy applies the method proposed in this paper, the default Hash strategy, the dynamic division strategy based on approximate sampling, and the XP strategy comparison that does not consider particularly inclined sampling and division processing. The word count experiment is performed on the above, and the task execution time found by determining the number of Reducers in this paper is compared with the number of Reducers under other settings.

In this paper, both the sampling optimization and the final partition strategy optimization take into account the extremely high degree of data tilt, so in both parts of the experiment, a specifically tilted data set needs to be used for verification. First, we consider the data resources shared by Sogou Labs externally, because it is easy to obtain and simple to operate its statistics, so we choose Sogou T-Link from the corpus data in the data resources. It includes a list of link relationships corresponding to all documents in the Internet corpus, and each row of data is relatively simple.

The full version of the above Sogou T-Link data set 2008 version is 25.9G. In the experiment, because of the limited performance of the physical machine, the 10G data set was selected. At the same time, the data set was counted through the Word Count program in Hadoop, and it was found the cardinality distribution of this data set is seriously skewed. Among them, the number of keys based on the cardinality 0 to 8 is 36,972, and the number after the cardinality 40 is only a single-digit size. Therefore, this data set is selected as a special tilt data set for verification.

In addition, in order to verify the effects of different tilt degrees on sampling optimization and partitioning strategy optimization, this study simulates the data set according to the conditions satisfied by the Zipf distribution.

4.2 Experimental Analysis of Sampling Efficiency

After processing the experimental data, a data set comprising 1 million data elements is generated. During the sampling test, it is divided into slices, and each slice uses the sampling method proposed in this paper, approximate sampling, and general random sampling at a sampling rate of 0.002 determined by the sampling rate analysis in this paper (combined with high frequency and general frequency analysis). Sampling is performed, and the sampling results of the three methods under different slope conditions are evaluated and analyzed by the standard error RMSE.

Combined with the root mean square error, the results of calculation and analysis of the data in the proposed sampling, the approximate sampling, and the general random sampling during the experiment are shown in Figure 5. It can be seen from Figure 5 that this paper's sampling method produces better accuracy compared to the other two kinds of sampling.

In addition, the experiment compared the time efficiency of the three sampling methods using the same order of magnitude (1G) and four different inclination levels, and the time efficiency of different orders of magnitude data on the same inclination level (this time uses Sogou T-Link data, the order of magnitude is 2G$\sim$10G). The comparison results of the two cases are shown in Figure 6 and Figure 7.

The data results in Figure 6 indicate that when the data set is at different degrees of inclination, the sampling method proposed in this paper is more efficient in terms of time compared with the other two sampling methods when dealing with the intermediate data set sample generated in the big data environment. From the analysis of the data results in Figure 6 and Figure 7, it can be seen that the sampling method proposed in this paper can improve time efficiency, reducing the overall time overhead.

4.3 Comparison of Performance Results of Partitioning Strategies

In order to comprehensively verify the rationality of the sampling rate 0.002 and the advantages of the division strategy proposed in this paper, the Sogou T-Link data set is sampled; the sampling rate is 0.002 with 0.0005 intervals. The experimental results and comparison are shown in Figure 8.

It can be concluded from Figure 8 that the division strategy in this paper has advantages in terms of the efficiency of task execution. In addition, the sampling rate obtained from the

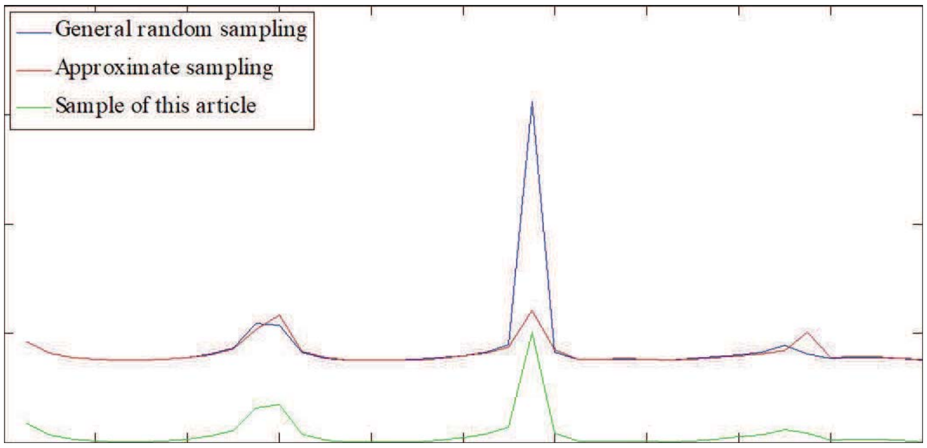

Figure 5 Data Analysis Results for Three Sampling Methods.

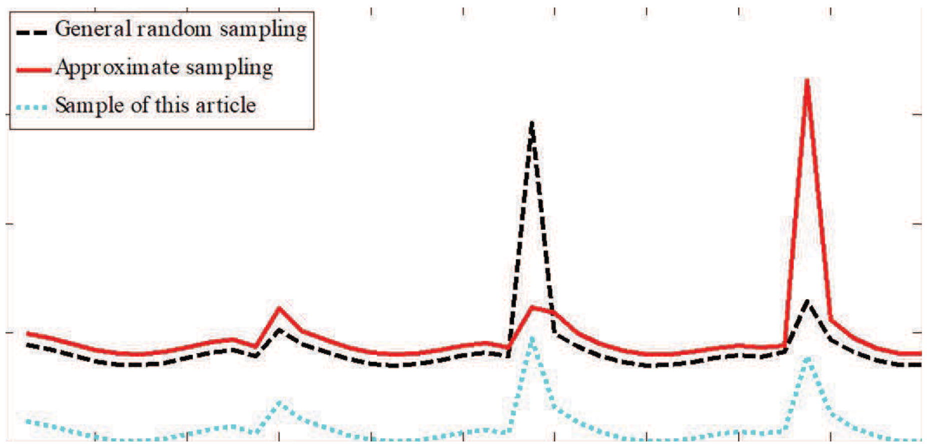

Figure 6 The Task Completion Time of the Three Methods With Different Inclination Degrees Under the Same Order of Magnitude.

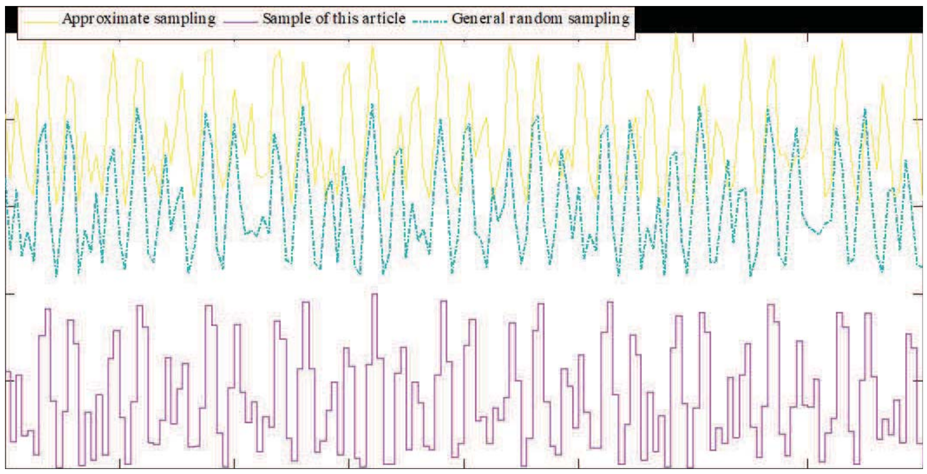

Figure 7 Different Orders of Magnitude Task Completion Time for the Three Methods at the Same Tilt.

analysis of sample S optimizes the load balancing technology and also improves the efficiency of cluster operations.

In order to verify the optimization of the number of Reducers proposed in this paper for load balancing, a simulated data set (capacity 4G, gradient 0.2) is selected for experimentation, and the number of Reducers is artificially set at 2–15. Figure 9 shows the comparison of the task completion time of the experiment. It can be seen from Figure 9 that the determination of the number of Reducers in this article is effective for the load balancing effect, which has great advantages over artificially setting the number of Reducers.

In order to verify the superiority of the proposed division strategy, a simulation data set (3G) is used to compare and analyze the job completion time data of the four division strategies in the cluster environment. In this round of experiments, the value of different data tilt degrees range from 0 to 1; the statistical results are shown in Figure 10. The data results presented in Figure 10 show that the proposed division

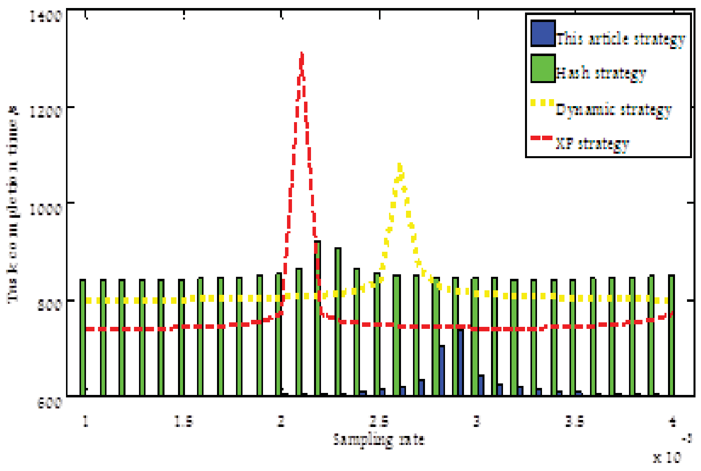

Figure 8 Task Completion Time of Four Division Strategies at Different Sampling Rates.

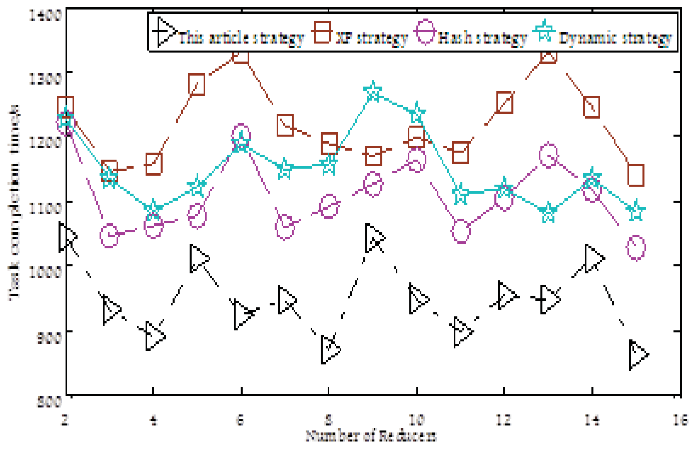

Figure 9 Task Completion Time of The Four Division Strategies Under Different Reducer Number Settings.

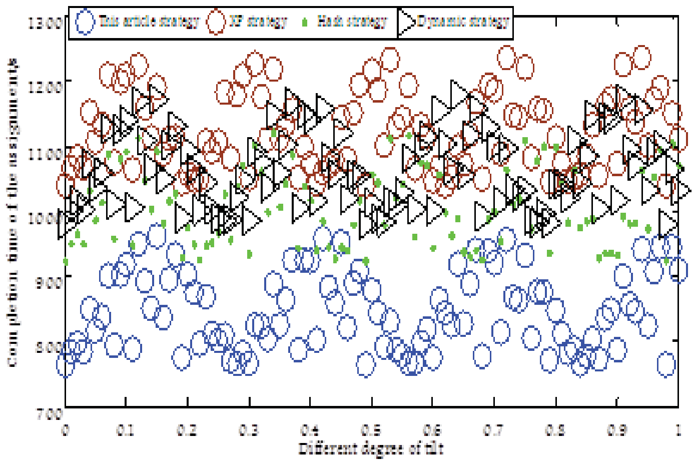

Figure 10 Job Completion Time of The Four Division Strategies Under The Condition of Different Inclination Data Sets.

strategy has obvious advantages in terms of time efficiency compared with other division strategies.

Using data slopes of 0.1 and 0.9, and the sampling of the data scale from small to large, the task completion times of the four division strategies are compared. These four strategies are the default hash algorithm, the dynamic partition strategy based on approximate sampling, the XP strategy without considering the specifically inclined sampling partition, and the dynamic partition algorithm based on sampling and load feedback optimization presented in this paper. The results are shown in Figure 11 and Figure 12 for comparison. The results presented in these figures indicate the significant advantages of the division strategy.

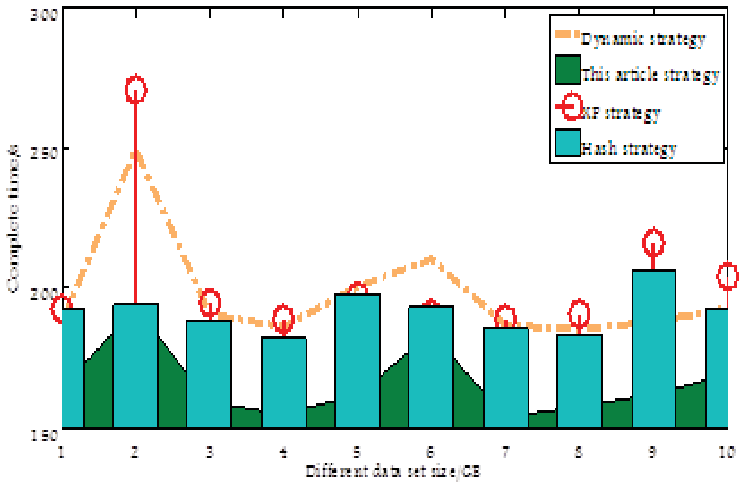

Figure 11 Completion Time of The Four Division Strategies for Different Data Sets (Inclination of 0.15).

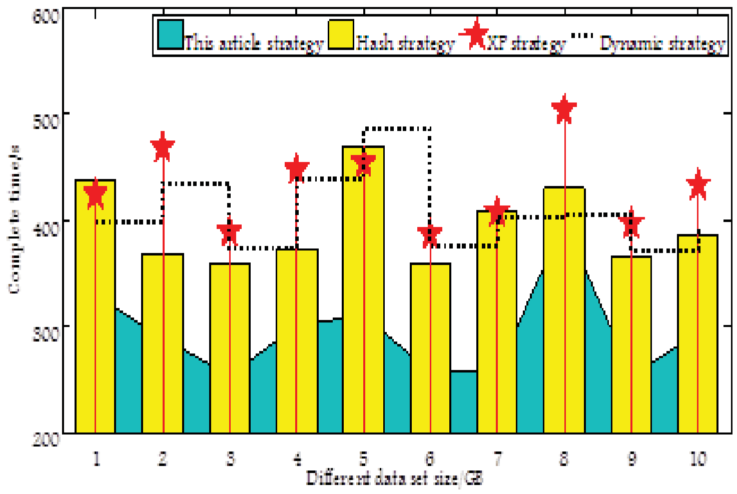

Figure 12 Completion Time of The Four Division Strategies for Different Data Sets (Inclination 0.95).

5. CONCLUSION

Spark is currently one of the mainstream frameworks for distributed computing, the performance of which affects the efficiency of big data processing and analysis. The paper examines and analyzes the load imbalance problem of Spark clusters, and proposes an optimized strategy for Spark task scheduling. This strategy has a demonstrable optimization effect on the load balancing of the Spark cluster, while shortening the overall running time of the task and improving the computing efficiency. Analysis results show that load balancing optimization on the Reduce side requires a series of processes to complete; in particular, the intermediate data generated on the Map side must be rationalized to improve the load balancing efficiency on the Reduce side. In addition, the load balancing on the Reduce side is also related to itself, and the determination of the number of task nodes plays a vital role in load balancing. For the processing of intermediate data, this paper combines sampling scale analysis and optimization with sampling method design optimization. The number of Reducers on the Reduce side is optimized according to real-time data and node performance. The optimization work of the two aspects is combined, and the optimal mapping between the intermediate data and the Reducer is established through the formulation of a dynamic division strategy which ensures the optimal realization of load balancing. The experimental results show that the load balancing strategy significantly improves the parallel computing power of Map Reduce in a cluster environment.

ACKNOWLEDGMENTS

This work was supported by the Sichuan Provincial Department of Education Foundation under grant no. JG2018-911, Higher education talent training quality and teaching reform project “Exploration and practice of innovative and entrepreneurial talent training mode for college students based on studio task driven”.

REFERENCES

1. Zhang Y., Liu S. A real-time distributed cluster storage optimization for massive data in internet of multimedia things. Multimedia Tools and Applications, 2019, 78(5): 5479–5492.

2. Tu L., Liu S., Wang Y., et al. An optimized cluster storage method for real-time big data in Internet of Things. The Journal of Supercomputing, 2020, 76(7): 5175–5191.
3. Hu W., Li H., Yao W., et al. Energy optimization for WSN in ubiquitous power internet of things. International Journal of Computers Communications & Control, 2019, 14(4): 503–517.
4. Zhang K., Zhu Y., Maharjan S., et al. Edge intelligence and blockchain empowered 5G beyond for the industrial Internet of Things. IEEE Network, 2019, 33(5): 12–19.
5. Liang W., Tang M., Long J., et al. A secure fabric blockchain-based data transmission technique for industrial Internet-of-Things. IEEE Transactions on Industrial Informatics, 2019, 15(6): 3582–3592.
6. Qiu T., Chi J., Zhou X., et al. Edge computing in industrial internet of things: Architecture, advances and challenges. IEEE Communications Surveys & Tutorials, 2020, 22(4): 2462–2488.
7. Savazzi S., Nicoli M., Rampa V. Federated learning with cooperating devices: A consensus approach for massive IoT networks. IEEE Internet of Things Journal, 2020, 7(5): 4641–4654.
8. Cao B., Li Y., Zhang L., et al. When Internet of Things meets blockchain: Challenges in distributed consensus. IEEE Network, 2019, 33(6): 133–139.
9. Savari G. F., Krishnasamy V., Sathik J., et al. Internet of Things based real-time electric vehicle load forecasting and charging station recommendation. ISA transactions, 2020, 97: 431–447.
10. Liu Y. H., Zhang S. Information security and storage of Internet of Things based on block chains. Future Generation Computer Systems, 2020, 106: 296–303.
11. Cheng L., Kotoulas S., Liu Q., et al. Load-balancing distributed outer joins through operator decomposition. Journal of Parallel and Distributed Computing, 2019, 132: 21–35.
12. Liu M., Yu F. R., Teng Y., et al. Performance optimization for blockchain-enabled industrial Internet of Things (IIoT) systems: A deep reinforcement learning approach. IEEE Transactions on Industrial Informatics, 2019, 15(6): 3559–3570.
13. ElHalawany B. M., Hashad O., Wu K., et al. Uplink resource allocation for multi-cluster internet-of-things deployment underlaying cellular networks. Mobile Networks and Applications, 2020, 25(1): 300–313.
14. Lyu Y., Yin P. Internet of Things transmission and network reliability in complex environment. Computer Communications, 2020, 150: 757–763.
15. Haji L. M., Ahmad O. M., Zeebaree S R. M., et al. Impact of cloud computing and internet of things on the future internet. Technology Reports of Kansai University, 2020, 62(5): 2179–2190.
16. Djellabi B., Younis M., Amad M. Effective peer-to-peer design for supporting range query in Internet of Things applications. Computer Communications, 2020, 150: 506–518.
17. Qadri Y. A., Nauman A., Zikria Y. B., et al. The Future of Healthcare Internet of Things: A Survey of Emerging Technologies. IEEE Communications Surveys & Tutorials, 2020, 22(2): 1121–1167.
18. Awin F. A., Alginahi Y. M., Abdel-Raheem E., et al. Technical issues on cognitive radio-based Internet of Things systems: A survey. IEEE Access, 2019, 7: 97887–97908.
19. Mei G., Xu N., Qin J., et al. A Survey of Internet of Things (IoT) for Geohazard Prevention: Applications, Technologies, and Challenges. IEEE Internet of Things Journal, 2019, 7(5): 4371–4386.
20. Xu Y., Xu W., Wang Z., et al. Load balancing for ultradense networks: A deep reinforcement learning-based approach. IEEE Internet of Things Journal, 2019, 6(6): 9399–9412.