Eng Int Syst (2022) 2: 115–126

Engineering
Intelligent Systems

Emergency Resource Allocation and Scheduling in Differential Distributed Storage System

Liang Chen*

Registrar's Office, Changsha Normal University, Changsha 410100, China

In order to address the problems of the high energy consumption and poor real-time performance of traditional emergency resource allocation and scheduling methods, an emergency resource allocation and scheduling method for distributed storage systems is proposed. With the proposed method, task requests are prioritized and priority queues are generated. The ant colony algorithm and the particle swarm algorithm are applied for efficient resource allocation, the selection of an appropriate scheduling strategy, and allocation of emergency resources. The experimental results show that the proposed method has strong real-time performance and can reduce energy consumption. Hence, it is highly suitable for emergency resource allocation and scheduling.

Keywords: Differential Distribution, Inventory System, Resource Allocation, Dispatching Method

1. INTRODUCTION

With the advent of the big data era, the total amount of global data being generated is showing unprecedented growth, requiring the rapid expansion of data centers. With the emergence of cloud computing and big data, a high-performance computing system is required comprising a large-scale cluster system. Its computing and communication operations are generally completed heterogeneously, and involve a heterogeneous processor, heterogeneous memory and heterogeneous communication network. With the continuous acceleration of CPU processing speed, the gap between the computing system and the computer storage system continues to widen (Chong et al., 2018; Al-Haija et al., 2019). Over the past decade, high-performance computing, cloud computing and big data have driven the development of storage systems. Through hardware data management technology and hardware acceleration technology, the performance gap among computing system, computer and storage system is narrowed, and the problem that the data generated by computer system exceeds the data processed by computer storage system is improved. Moreover, the development of high-performance multi-core computing has changed from simple large-scale parallel computing to one that has high efficiency, high reliability and low energy consumption. Therefore, a lot of research work begins with software storage management technology, combining data distribution technology and task scheduling technology to improve the efficiency and reliability of high-performance computers, and reduce the overall energy consumption of the system (Feng et al., 2018).

The common storage system stores big data by means of special servers and disk arrays. However, if the storage load of servers in the storage system is too large, this may cause a bottleneck problem for the storage system (Naidu et al., 2018; Bramasta et al., 2019). This kind of storage system cannot meet the storage requirements of big data. In order to be able to store big data, a differential distributed storage system is adopted. Unlike the network storage system, when the distributed storage system stores data, it separates the data processing from the storage process,

*Corresponding Author Email: cliang0816@163.com

and effectively stores the big data. Distributed storage system emergency resource scheduling usually refers to the allocation of emergency resources to each resource demand point after an emergency occurs. The current emergency resource scheduling method cannot efficiently schedule the resources contained in the storage system. The emergency resource task scheduling and data distribution technology of differential distributed storage system includes two parts: task scheduling and data distribution. This requires scheduling tasks on the right processor and data on the right memory, so as to improve performance, reduce latency and meet resource constraints. At present, the scale of high-performance systems is constantly increasing, and the structure scale is quite different. At the same time, the application scale is constantly expanding, its task parallel mode is different, and the data set is huge. These characteristics increase the complexity of data allocation and task scheduling technology, making it difficult to improve the throughput of the system. Moreover, the resource allocation scheduling method has the problems of high energy consumption and poor real-time performance. Many experts have studied the emergency resource scheduling method of distributed storage systems to find an approach that will enable each resource demand point to receive emergency resources quickly.

Hammuda et al. (2018) proposed to establish a multi-source and multi-objective emergency resource scheduling model. This method takes the shortest emergency disposal time as the planning goal. The sum of emergency resource transportation and emergency engineering construction time must be less than the time when pollutants diffuse to the emergency disposal space location, and the number of comprehensive dispatching resources of multiple emergency resource warehouses must meet the disposal demand as the constraint condition. When the time of emergency resource delivery and loading and unloading is fixed, The number of resources to be dispatched in each warehouse and the time required for resources to arrive at the emergency disposal space are calculated. Dijkstra algorithm is used to select the optimal path of emergency resource distribution in real time, which improves the execution efficiency of emergency management. However, this method has the problems of poor real-time resource scheduling ability and untimely scheduling of emergency resources. In reference, Xu et al., (2018) an improved resource scheduling algorithm is proposed for the cloud storage environment. In this method, the triangular fuzzy number analytic hierarchy process is used to conduct a comprehensive analysis of the factors that influence scheduling. A judgment matrix of storage nodes is obtained, which is subsequently used to construct the genetic algorithm objective function. Then, the simple genetic algorithm is innovated in terms of solution coding, cross mutation operation and lethal chromosome self-improvement, making it suitable for large-scale resource scheduling in the cloud storage environment (Dzhurik et al., 2019; Phuong et al., 2019). This method can effectively schedule emergency resources, but has poor reliability. In reference, Fang et al., (2019) a multi-objective two-stage temporary distribution center location and emergency resource scheduling model are established to minimize the total cost and total time. The relative robust optimization method is used to create the model, and the interval estimation is used to describe the uncertain factors, and the robustness and optimality of the model are introduced. The model can effectively solve the problem of emergency resource scheduling network construction when there is uncertainty of demand, and can ensure the robustness of emergency decision-making. However, this method requires high energy consumption which is not conducive to practical application.

Given the above problems considered above, this paper puts forward a proposal for an emergency resource allocation and scheduling method for a differential distributed storage system. The resource scheduling program used in this method adopts dynamic management resource allocation mechanism. According to the scheduling request, it sets the grade score for the emergency resource demand. By introducing the potential energy method to optimize the priority, it makes a reasonable arrangement for the queue data waiting for allocation. Combined with ant colony algorithm and particle swarm algorithm, the corresponding resource allocation container is selected to complete the allocation of emergency resources. The experimental results show that the method has better real-time performance, improves the execution efficiency of management in practical operation, improves the scheme of emergency resource allocation and scheduling, and is suitable for emergency resource scheduling.

2. DESIGN OF EMERGENCY RESOURCE ALLOCATION AND SCHEDULING METHOD FOR DIFFERENTIAL DISTRIBUTED STORAGE SYSTEM

Most of the differential distributed storage systems are based on the development and operation of the Hadoop platform. For emergency resource allocation scheduling, we can refer to the resource scheduler instead of the original task scheduler to achieve the unified allocation of emergency resources.

2.1 Resource Scheduler Reference

The differential distribution storage system supports the management and allocation of two different resource types: memory and CPU. The resource scheduler adopts the Dynamic management of resource allocation mechanism to submit requests for resource allocation by processing applications.

When Node Manager (NM) registers with Resource Manager (RM) through the Resource Tracker protocol to report node status, the total amount of resources available in the node is submitted. In the differential distributed system, the default memory in each NM is 8 GB and has 8 virtual CPU cores. The NM periodic landlord moves to RM to issue a request and report resource information, while the application requests resources from RM via Resource Request, a request containing the following fields:

(1) Priority: Represents a resource priority (O priority is the largest, and the greater the number, the smaller the priority).

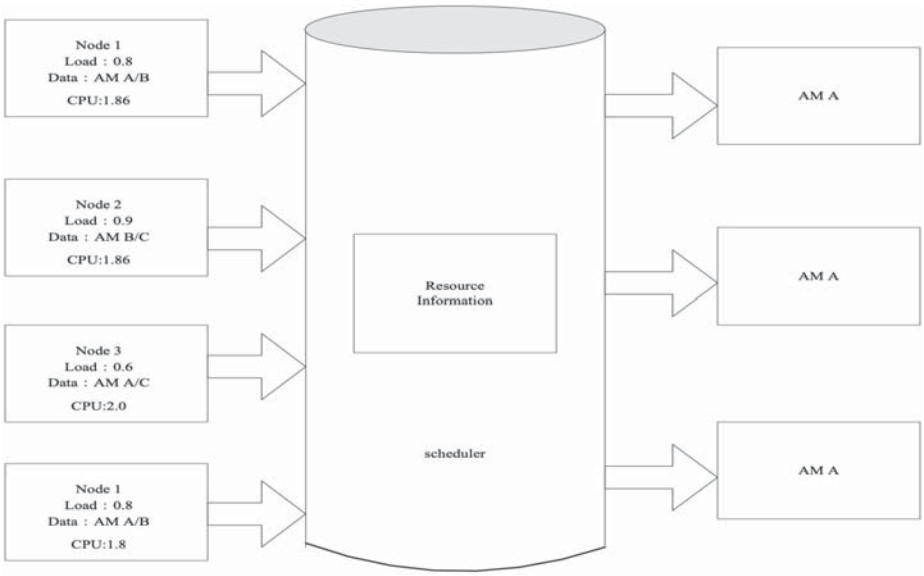


Figure 1 Resource scheduler allocation process.

(2) Resource Name: Represents the location of the requested resource (*: indicates an arbitrary location resource; /Rack-name: Applies for a resource at any node on this rack; /Host-name: Applies for a resource at a node).

(3) Capability: Specification of the resource, including the amount of the two resources mentioned above (e.g.: 2 GB memory, a virtual CPU core).

(4) Num-container: The number of Containers requested to meet the above resource specification description (Liang et al., 2019).

(5) Relax_ locality: Whether to force locality.

Submits multiple resource requests based on the number of copies of data and responds according to the order of local jobs and arbitrary locations when responding to Resource Request (Heinonen et al., 2019). In this way, a response will result in a greater number of requests being withdrawn. The scheduler will schedule according to a series of conditions, starting with the selection of queues. These must be considered by the scheduler to determine the queue because the queue has two attributes:

(1) Minimum capacity: The scheduler needs to work hard to meet the value of the minimum capacity. In a multi-queue environment, the scheduler selects the queue with the largest difference between actual and minimum capacity of the queue, assigning resources to these queues to ensure the efficiency of job execution for these queues. When a queue is idling, its resources are allocated to other queues for other queue applications until this queue again submits a request and applies for resources (Amine, 2019).

(2) Maximum capacity: The queue also sets a maximum capacity when resources are tight. The maximum capacity value is the queue hardness requirement value. At no time can the queue's share of resources exceed this value. In general, this value is not set to improve the efficiency of the queue, so that a busier queue is able to fully use the cluster resource (occupying 100% of the resource) when the other queues are not working, and then release it stepwise when the other queues are applied.

Resource manager is hereinafter referred to as RM, node manager is hereinafter referred to as NM, and application master is hereinafter referred to as AM.

In a queue, the queue user submits the application, and the scheduler responds to the resource request depending on the resource situation and the applicant's request (Fu et al., 2019). The new resource scheduler implements a two-tier scheduling mechanism. In the first layer, the above-mentioned Resource Request is submitted to the RM by the application, and the scheduler's only job is to determine whether to respond to the Resource Request submitted by the application and the resource situation of each node, and to initiate one or more than one Container on the node if there is no response.

In a differential distributed storage system, the allocation of resources is done with the following steps:

Step 1: NM uses heartbeat information to report node status to RM.
Step 2: The Resource Tracker service in RM returns a response that contains information to start and release Container.
Step 3: The service forwards the NODE_UPDATE event to the scheduler.
Step 4: The scheduler relabels the released resources as available, and then assigns the resources on the node to the application according to certain policies.
Step 5: AM sends heartbeat messages to RM.
Step 6: RM responds to AM by sending the allocated Container.
Step 7: AM assigns Container to its own internal tasks.

The resource allocation strategy is shown in Figure 1.

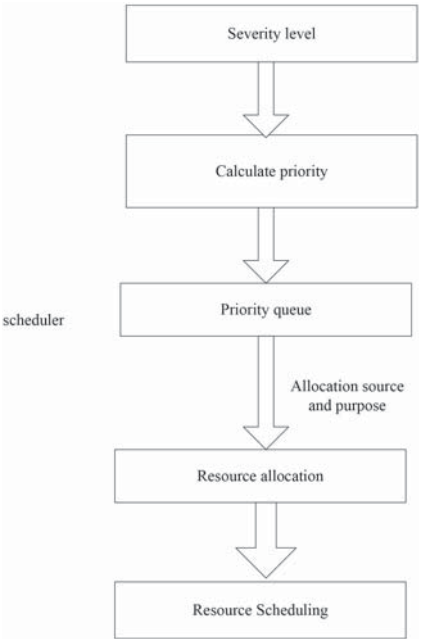


Figure 2 Resource scheduling process.

Figure 1 depicts the resource allocation strategy for the current resource scheduler. When AMA applies to the Scheduler component in RM for Container, assuming that the current location node Node2 and the resources available in Node2 meet the those resources required for AMA; the scheduler allocates the resources in Node2 to AMA. Compared to Node2, Node3 has a smaller resource load and faster CPU processing speed, so it is more reasonable to allocate resources to AMA in Node3.

2.2 Allocation of Priority

As the implementer of the emergency resource allocation scheduling method of the differential distributed storage system, the resource scheduler is used depending on the priority of resource allocation. The workflow of the resource scheduler is shown in Figure 2.

Firstly, after the scheduler obtains the request from different resource levels, the priority scheduling module produces a priority queue. Then, the resource allocation module selects the appropriate allocation source and allocation purpose for the resources in the queue. Finally, the scheduling module completes the emergency resource scheduling work for multiple emergency resources according to different levels. Therefore, the prioritization of resource allocation is important. It is very common for multiple potential failures to occur simultaneously in different distributed storage systems, so how to allocate emergency resources appropriately becomes a key problem. A more reasonable approach is to allocate more resources to tasks with greater urgency according to the crisis level (Yang et al., 2019). When allocating resources, it is necessary to establish the relevant severity threshold to determine whether a failure has occurred, and the severity of the failure. This score enables critical tasks to be prioritized according to the level of severity, in order to facilitate subsequent resource allocation.

The percentage of resource allocation is also an important factor in scoring the severity of failure, in addition to the severity level as the evaluation standard. If the severity of the failure is high, the score of the task is high, and more resources need to be allocated, then the remaining resources will be reduced. The relevant severity score was calculated as follows:

$$\gamma(i) = \frac{1 - x(i)}{K} \tag{1}$$

In formula (1), $x(i)$ represents the percentage of resources allocated, K Indicates that the failure is the severity of the event, $\gamma(i)$ represents related severity scores.

After setting the relevant severity score for the execution task, the system can distinguish the different degrees of crisis according to the different scores. Without taking into account the percentage allocation, the higher the severity, the greater the scores assigned to the higher the risky tasks. In order to prioritize the allocation of resources to higher-level missions, priority teams are used to prioritize the allocation of work. The queue takes into account not only the task-related severity score, but also the crisis status of other tasks. The task queues are sorted in situations where the overall priority processing scores are higher and other crisis states are more severe.

The concept of a priority queue is introduced for the purpose of allocating emergency resources more efficiently. There are two kinds of tasks in differential distributed storage system: one is the normal read and write task; the other is the emergency task. The user's normal read-write tasks are added to the task queue in the original "first-come-first-served" manner, while the emergency task needs to be added to the priority queue in order of priority and then to the work queue in the same order as in the priority queue. Using the on-duty process to assign queue processes, the system is regularly scanned for abnormal data, multiple tasks are added to the priority queue according to the priority level, and finally the priority queue is followed. The work queue is joined in order

to wait for the resource scheduler to schedule. In the priority queue method, the potential energy method is used to parse the priority queue data entry time. This method is usually used to determine the complexity of the allocation, and in the emergency resource allocation method, the task is divided into general task and emergency task. The potential energy method is introduced to optimize the entry time of field priority queue data to improve operational efficiency.

Assume that a priority queue data needs to perform n operations, for each $i(i = 1, 2 \ldots n)$, make a_i the actual priority of the i task level. The priority queue operates the potential energy ϕ_{i-1} and ϕ_i by potential energy methods. Therefore, the operation return time of the potential energy function i is represented as follows:

$$b_i = a_i + \phi_i - \phi_{i-1} (i = 1, 2 \ldots n) \tag{2}$$

According to Equation (2), the total time of the n operation is:

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (b_i - \phi_i + \phi_{i-1}) = \sum_{i=1}^{n} b_i - \phi_n + \phi_0 \tag{3}$$

In formula, ϕ_0 generally represents 0, and $\phi_i \geq 0$, thus:

$$\sum_{i=1}^{n} a_i \leq \sum_{i=1}^{n} b_i (i = 1, 2 \ldots n) \tag{4}$$

2.3 Emergency Resource Scheduling Based on Ant Colony Algorithm and Particle Swarm Algorithm

2.3.1 Resource Scheduling Based on Ant Colony Algorithm

Ant represents the scheduler in the ant algorithm and is responsible for allocating resources to the container of the applied resources to obtain the resource allocation scheme (Zhang et al., 2019).

Firstly, the pheromone is initialized, and in the differential distributed storage system, whether it is Map or Reduce, factors such as CPU rate, memory volume and load of the node have a crucial effect on the execution of the task (Zhang et al., 2020). In general, tasks are not assigned to nodes with low CPU execution speed, small memory, and heavy load, but should begin with high performance nodes, such as high CPU rate, low load, and low corresponding job failure rate, so as to reduce the task execution time. In the differential distributed storage system resource scheduling framework, AM is responsible for job start-up and monitoring, RM is responsible for the management and allocation of all resources in the cluster, and monitors, manages the status of AM on each node, communication between AM and RM through a heartbeat transmission mechanism. This paper refers to the resource scheduler, through which the node CPU rate, job failure record, memory capacity and load information are obtained from NM, the execution ability of each container on the node is calculated, and the result is stored in the pheromone matrix in the form of pheromone value, complete the initialization of pheromone matrix $Q_{n \times m} = (q_{in})_m$.

Suppose q_{in} represents the execution capability of container E_n on resources r_i. It is calculated by the CPU rate of the node, the memory, the relative weight of the job failure records of the container, and the resource r_i remaining utilization, the expression is:

$$q_{in} = \left(\frac{cpuV_i}{\sum_{i=1}^{m} \frac{cpuV_i}{m}} + \frac{W_i}{\sum_{i=1}^{m} \frac{W_i}{m}} + \frac{\alpha}{fai \ln ote_{in}} \right) * (1 - L_i) \tag{5}$$

$$L_i = \frac{W_i - resW_i}{W_i} \tag{6}$$

In the formula, $fai \ln ote_{in}$ show E_n failing record of the job on the r_i node, $cpuV_i$ represents the CPU rate, W represents memory capacity, α represents the weight value of the job failure record to which the container belongs. After the initialization of the pheromone is completed, the state transfer probability needs to be calculated. In an iterative process, the allocation of resources is accomplished by multiple "node container" operations. The details are as follows:

Suppose the ant $a_k (0 \leq k \leq q)$, k indicates the number of ants. Random selection of node resources $r_i (0 \leq i \leq m)$. The update state transition probability matrix of a_k is $D_{m \times a}$. Select the container with probability $d_{in} (0 \leq n \leq a)$. Multiple choices may be made if the termination condition is not met. For example, suppose the container selected by the current node resource R is E_n, and $(r_i, E_n) > 0$, then r_i allocates resources for E_n and modifies matrix elements at the same time $x_{in} = 1$. Conversely, the container is re-selected until the constraints are met. The transfer probability for a_k to select container E_n at time t is:

$$d_{in}(t+1) = \begin{cases} \frac{q_{in}(t)|E_n r_i|}{\sum_{x,y=1} q_{in}[|E_n r_i| - D_a]} & if\ y_{in} = 1 \\ 0, else \end{cases} \tag{7}$$

In the formula, x, y represents a matrix element, the constraints are:

$$\sum_{i=0}^{n} \sum_{n=0}^{i} x_{in} = t \tag{8}$$

$$\sum_{i=0}^{n} \sum_{n=0}^{i} y_{in} = 0 \tag{9}$$

Of which, $\sum_{i=0}^{n} y_{in} = 0$, when the above conditions are met, the allocation of resources r_i has been completed.

Every time a_k selects a container, if container E_n and resource r_i, the larger the corresponding pheromone is q_{in}, the greater the probability that the container is selected, which to some extent reduces the progress of clustered job sets.

When the constraints are met, the ants a_k end one iteration. According to the following formula, the objective function value of the current solution is:

$$F_k = \sum_{i=0}^{n} [\vartheta_i * g_i] \tag{10}$$

$$\varphi_i = \frac{\frac{resWh_i}{resTotalh_i}}{\sum_{i=0}^{n} \frac{resWh_i}{resTotalh_i}} \tag{11}$$

In the formula, F_k represents an objective function, the global optimal solution for emergency resource allocation, g_i represents operational progress ϑ_i is the task h_i progress weight value, which is calculated by Equation (11).

The current solution is compared with both the local optimal solution and the global optimal solution. If the current solution is superior to the local optimal solution and the global optimal solution, the latter two solutions are updated with the current solution; otherwise, they are not updated. When the number of iterations is less than 8, the volatilization coefficient is set to a fixed value of 0.3. When the number of iterations is greater than or equal to 8, the adaptive pheromone update mechanism is adopted. If the change of the global optimal solution in 8 consecutive iterations tends to be stable, the pheromone volatilization coefficient is appropriately increased, whereas the pheromone volatilization coefficient is reduced. These updates are only for the resource allocation included in the solution obtained from this iteration.

After obtaining 8 consecutive job progress F_k, the standard deviation for calculating the objective function ε is:

$$\varepsilon = \sqrt{\frac{1}{n-1}\sum_{q=1}^{n}\left[F\left(d_k\right)_q - \overline{F\left(d_k\right)_q}\right]^2} \tag{12}$$

Substituting standard deviation ε into Equation (13), the calculated dynamic evaporation coefficient ξ_a is:

$$\xi_a = \frac{\arctan\left(\varepsilon\right)}{\frac{\pi}{2}} \tag{13}$$

When the standard deviation of the calculated objective function is small, the algorithm's solution finding process tends to be stable. When the Volatilization Coefficient ξ_a of dynamic pheromone is large ($0 < \xi_a < 1$), the search for solutions fluctuates. The adaptive pheromone update mechanism is used to improve the ants' global search ability by adjusting the volatile amount of pheromone to prevent the ants from falling into the local optimal solution.

The volatile coefficient of dynamic information is used to update the pheromone, namely:

$$q_{in}(t+1) = \begin{cases} (1-\xi_a)q_{in}(t),\ flag_n = P \\ (1-\xi_a)q_{in}(t) + \triangle\ q_{in}(t),\ flag_n = Z \end{cases} \tag{14}$$

$$q_{in} = \frac{resWE_n}{\sum_{i=0}^{n} resW_i} \tag{15}$$

In the formula, $flag_n$ indicates the state of container E_n, and q_{in} indicates the increment of pheromone, P represents the state of the container's requested resource of zero represents the state of the container's requested resource of zero. Conversely, the state of the container is Z, if $flag_n = Z$, the update of pheromone includes not only the volatilization of pheromone, but also the increase of pheromone accumulation.

Ant colony algorithms tend to choose large containers when allocating resources. When allocating cluster resources, if the free resources are allocated to the small container, a large amount of small-capacity resource fragments will be generated, and the large capacity container cannot find adequate resources to meet the conditions, resulting in a waste of resources. The pheromone update process shown in Equation (14) and Equation (15) can avoid the above problems, so that the resources are allocated to the container to the maximum extent possible by the cluster performance to implement the emergency resource scheduling.

2.3.2 Optimization of Resource Allocation Based on PSO

The particle swarm algorithm is derived from random unraveling to find the optimal solution by iteration. In this paper, based on the ant algorithm, the particle swarm optimization algorithm is used (Zheng, 2019).

Suppose there are m particles in the emergency resource allocation group searching in the c-dimensional space. In step v, the position vector of the j particle in the c-dimensional space is $x_j^v = \{x_{j_1}^v, x_{j_2}^v, \ldots, x_{j_c}^v\}$, $j = 1, 2, \ldots m$, adaptation function defines emergency time according to relevant requirements $T_j^v = T(X_j^v)$, the position of the best solution found by the j-th particle itself is $Q_j^v = \{q_{j_1}^v, q_{j_2}^v, \ldots, q_{jm}^v\}$, $U^v = \{u_{j_1}^v, u_{j_2}^v, \ldots, u_{j_m}^v\}$ for the best possible solution for the entire population, the velocity corresponding to each particle is $S_j^v = \{s_{j_1}^v, s_{j_1}^v, \ldots, s_{j_m}^v\}$. The updated velocity and position of the particle are:

$$S_j^{v+1} = \varpi s_{j_1}^v + d_1\psi(Q_j^v - X_j^v) + d_2\lambda(U_j^v - X_j^v) \tag{16}$$

$$X_j^{v+1} = X_j^v + \tau v_j^{v+1} \tag{17}$$

In the formula, ϖ is inertial factors, d_1, d_2 represents group cognitive coefficient, ψ, λ represents the uniform distribution coefficient of random numbers in an interval. The PSO calculation flow is shown in Figure 3.

3. SIMULATION EXPERIMENTAL RESEARCH

3.1 Experimental Scheme

In order to verify the effectiveness of this method, simulation experiments are carried out. The method is evaluated by the benchmark program of dspstone. First, the benchmark program is compiled by GCC, and then the task graph and read / write data set are extracted from GCC. The compilation and extraction process comprises three stages: first, compile the original code with - fprofile generate; second, execute the binary data set generated after the compilation of the corresponding case; finally, the source code is optimized by profile guided to a feasible absint (- fprofile use fabsinth) for recompilation. Pass - absinth - BBS traverses all RTL expressions in each basic block. For each expression, pass absinth BBS determines whether it is an instruction, and performs an initial execution for each instruction. Then, the task graph and access data set are put into the simulator. The number of tasks, dependent sides, and data sets for the benchmark are shown in Table 1.

When extracting task graph from GCC to build MDFG of weight, it is necessary to design the parameters shown in

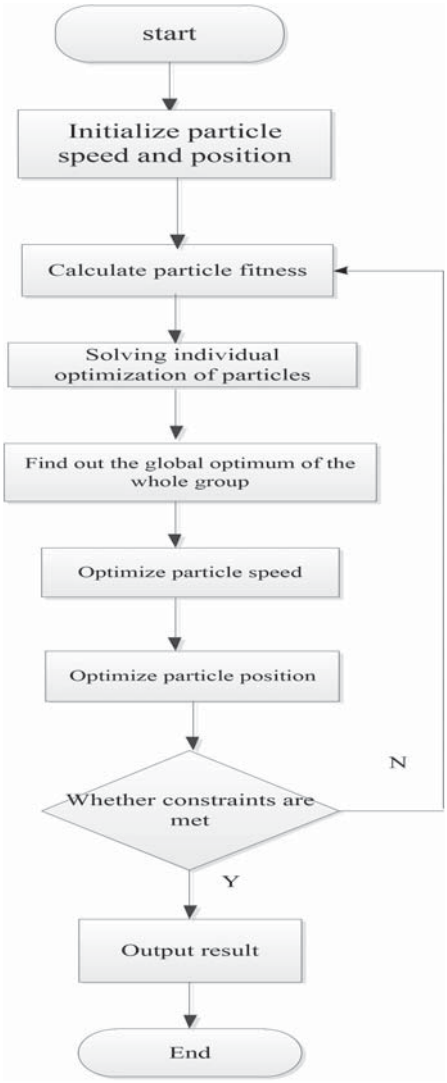


Figure 3 Flowchart of particle swarm algorithm.

Table 1 DSP stone benchmark test procedure.

Numbering	1	2	3	4	5
Number of tasks	8	13	17	29	178
Number of edges	7	15	21	38	146
data	12	24	32	48	180
$\bar{C}_d$	6 KB	8 KB	4 KB	5 KB	7 KB
ETR	1.1	1.25	0.95	1.75	1.2
β	0.2	0.5	0.8	0.45	1.0

Table 1 as required. Task execution time: this is a basic heterogeneous parameter based on processor speed. The higher the β value is, the more time a task is executed on different processors. Setting $\bar{T}_d$ is the average time to map tasks, in resource allocation scheduling. Then the average execution time $\bar{T}_i$ of task q_i is randomly selected from mean and distribution $\left[0, 2\bar{T}_d\right]$. Accordingly, the execution time of each task on q_i processor W_j must conform to the expression range of the formula given below:

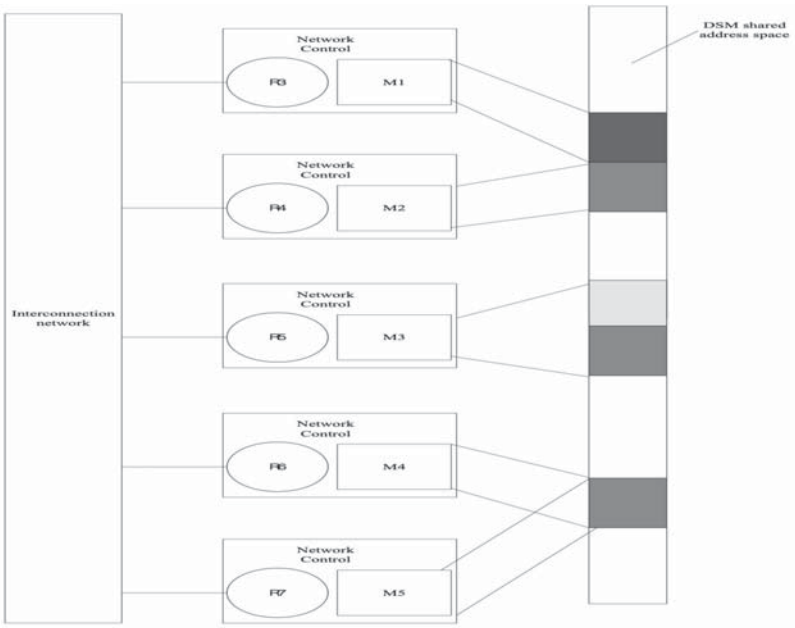
$$\left(1-\frac{\beta}{2}\right)\bar{T}_i \le U(q_i, W_j) \le \left(1+\frac{\beta}{2}\right)\bar{T}_i \tag{18}$$

In the formula, $U(q_i, W_j)$ is the uniform random variable with the mean value of $\bar{T}_i$ and the variance of $\beta\bar{T}_i$, i and j are constants.

The table ETR indicates the energy consumption time ratio, which is the ratio of the average energy consumption and the average execution time of the MDFG. Set $\bar{B}_j$ to represent the scaling of processor W_j is the energy consumption. Then the energy consumption of task q_i on processor W_j is:

$$B(q_i, W_j) = \bar{B}_j \times U(q_i, W_j) \times ETR \tag{19}$$

Suppose the number of tasks in MDFG is Q, the number of dependent edges in MDFG is V, then the data required to perform a given task diagram is:



(a) Structural model

Memory	P1		P2		P3		P4		P5	
	AT1	AE1	AT2	AE2	AT3	AE3	AT4	AE4	AT5	AE5
M1	1	2	3	4	4	6	5	8	2	3
M2	4	3	1	2	4	7	5	9	3	3
M3	3	3	4	5	1	1	6	8	2	2
M4	4	6	6	7	7	10	2	2	4	5
M5	2	2	3	3	4	9	5	13	1	1

(b) Model parameters

Figure 4 Experimental structure model.

$$M_d = \chi \times \sqrt{Q} \times \sqrt{V} \tag{20}$$

In the formula, χ represents a data parameter.

$\bar{C}_d$ in the table represents the average number of each required in MDFG, and is randomly given in the resource allocation scheduling method. For a data r, the data size parameter is randomly selected from the uniform distribution [0, 2]. For a data, the data size parameter is randomly selected from a uniform distribution. Therefore, the data r size of each data is:

$$d\,(r) = \eta \times \bar{C}_d \tag{21}$$

3.2 Experimental Environment and Parameter Setting

In the experiment conducted to test the performance of various emergency resource allocation scheduling methods for differential distributed storage systems, the time, energy consumption, real-time, load equilibrium of resource scheduling are factors used as performance standards in the experiment which compares the reference [5] method, the reference [6] method and the reference [7] method. Based on the performance criteria for comparison, two sets of experiments are required, in which all the benchmark test programs are run according to the experimental structure model shown in Figure 4.

The structure model shown in the figure consists of five heterogeneous processors. The parameter set of the structure model is obtained from ARM7 and MSP430 using the CACTI tool. The specific parameter settings are shown in Table 2.

In Table 2 and the structural model in Figure 3, the experimental results for access time and access energy consumption of the model unit data in different processors have been given. All experiments are performed on the computer of the Linux system, which has a memory of 2 GB.

Table 2 Parameters of experimental structure model.

Parameter	Frequency (MHz)	Local Memory Size (KB)	Prolonged (ms)	Energy Consumption (mJ)
Core1	64	128	1.25	1.39
Core2	30	64	2.36	1.22
Core3	15	32	2.11	1.66
Core4	12	16	1.25	0.73
Core5	8	8	0.93	2.18

Table 3 Comparison of energy consumption results obtained by different methods used for resource allocation and scheduling.

Method	Experiment Number	Constraint Time / min	Energy Consumption / J	Energy Consumption Ratio / %
Literature [5]	1	20	196	2.3
	2	25	236	2.6
	3	30	275	1.5
	4	35	294	3.1
	5	40	-	-
Literature [6]	1	20	177	3.5
	2	25	196	2.3
	3	30	225	2.7
	4	35	263	1.9
	5	40	298	3.4
Literature [7]	1	20	185	2.2
	2	25	199	2.5
	3	30	220	2.6
	4	35	241	1.8
	5	40	287	2.9
The method of this paper	1	20	112	7.6
	2	25	93	8.3
	3	30	132	6.9
	4	35	89	8.9
	5	40	97	7.7

3.3 Analysis of Experimental Results

3.3.1 Energy Consumption Analysis of Emergency Resource Allocation for Different Methods

To verify the feasibility of the proposed method, the experiments compared the energy consumption of the reference [5] method, the reference [6] method, and the reference [7] method in carrying out the emergency resource allocation. The lower the energy consumption, the more efficient is the method. The results are shown in Table 3.

The energy consumption rate shown in the table is the reduction value of the initial energy consumption compared with the stable energy consumption of the resource allocation method, which indicates that the method has no optimal solution of resource allocation scheduling under the time constraint. According to the data results in the table. In Phuong (2019) related research, the energy consumption is 196 at least, 294 at most, 3.1% at most, and 1.5% at least. Five experiments have unresolved problems, that is, resource allocation and scheduling failure. In Dzhurik and Belov (2019) related research, the energy consumption is at least 177, the most 298, the highest 3.5%, and the lowest 1.9%; In Yang (2019) related research, the energy consumption is at least 185, the maximum is 287, the maximum energy consumption ratio is 2.9%, and the minimum is 1.9%. The maximum energy consumption of the proposed method is 132, at least 89, the maximum energy consumption is 8.9%, and the minimum energy consumption is 6.9%. In conclusion, the maximum and minimum energy consumption of this method are higher than the other three methods, which proves the feasibility of this method.

3.3.2 Analysis of Emergency Resource Allocation Time for Different Methods

To verify the working efficiency of the method in this paper, the experimental analysis of the method in this paper, the method of reference [5], the method of reference [6] and the method of reference [7] for emergency resource allocation need to be compared in terms of time; the shorter the allocation time, the higher is its working efficiency. The results are shown in Figure 5.

From the analysis of Figure 5, it can be seen that under the same conditions, the time taken to allocate emergency resources is different for each of the four methods. When the number of experiments is 2, Phuong (2019) method takes about 8 minutes to deploy, Dzhurik and Belov (2019) method

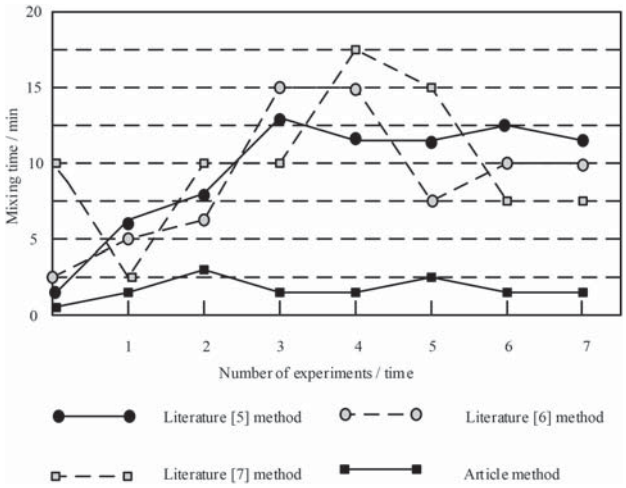


Figure 5 Comparison of time allocation for different resource allocation scheduling methods.

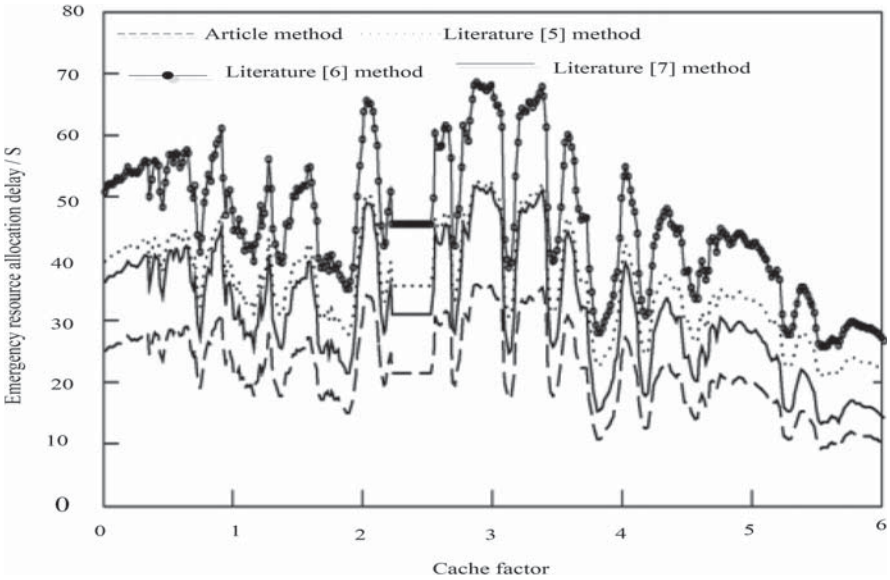


Figure 6 Comparison of emergency resource allocation delays for different methods.

takes about 6 minutes to deploy, Yang (2019) method takes about 10 minutes to deploy, the deployment time of this method is about 3 min; When the number of experiments is 7, Phuong (2019) method takes about 12 minutes to deploy, Dzhurik and Belov (2019) method takes about 10 minutes to deploy, Yang (2019) method takes about 7.5 minutes to deploy, and that of this method is about 2 minutes. Through comparison, it can be seen that this method is short in emergency resource deployment, high in work efficiency, and has certain reliability.

3.3.3 Analysis of Emergency Resource Scheduling Delays in Different Methods

In order to verify the real time performance of this method, compared with the Phuong (2019) method, Dzhurik and Belov (2019) method and Yang (2019) method, to calculate the delay time of each method, the shorter the delay time, the higher the feasibility. The results are shown in Figure 6.

It can be seen from Figure 6 that the four methods change with the change of cache coefficient when the emergency resource is allocated. When the cache coefficient is 2, the allocation time of reference [5] method is about 29 s, the allocation time of reference [6] method is about 39 s, the allocation time of reference [7] method is about 23 s, and the allocation time taken by this method is about 17 s. For 10 s, the curve shows that the method has the shortest time delay and quickest reflection, which proves that the proposed method has better real-time performance.

3.3.4 Load Equilibrium for Deployment of Emergency Resources by Different Methods

In order to verify the advantages of this method, the load balance of the four methods is compared experimentally. The load balance is constructed on the network structure to enhance the scheduling performance of emergency resources, and the larger the value of load equilibrium is, the more balanced is the load of the emergency resource scheduling node, and the utilization ratio of emergency resources in the storage system is improved. The results are shown in Figure 7.

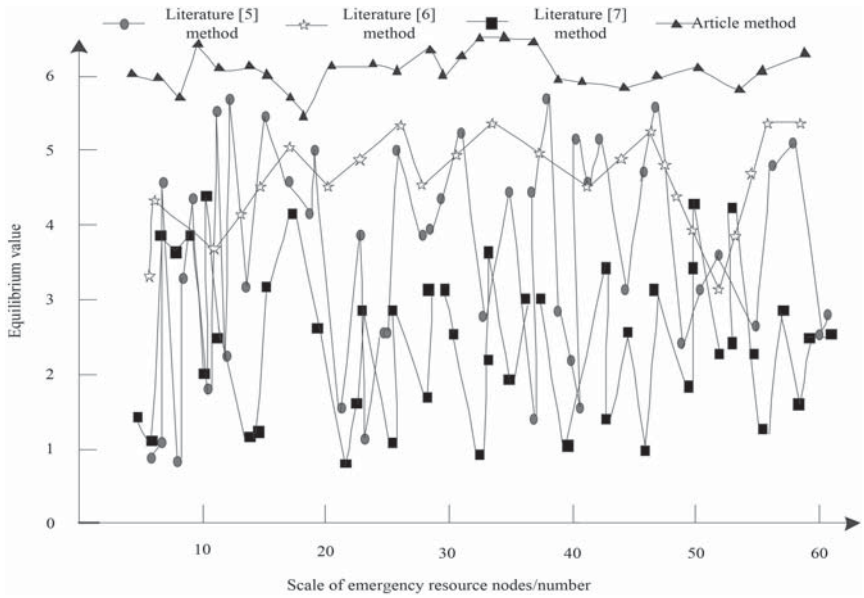


Figure 7 Comparison of emergency resource scheduling load balance for different methods.

Figure 7 shows that the equilibrium value of this method is higher than that of the other three methods as the scale of the computing emergency resource node increases. This is because this method determines the priority level in the scheduling of resources, which reduces the time required to allocate resources, and improves the efficiency of emergency resource allocation.

4. CONCLUSIONS

With the advent of the era of big data, the current emergency resource scheduling method cannot meet the users' demands for real-time emergency resource scheduling and low energy consumption. In order to meet these demands, this paper proposes a differential distributed storage system emergency resource allocation and scheduling method. Here, the resource allocator is introduced to schedule tasks in order of priority. In the priority queue, the potential energy method is used to analyze the data-entering time of the priority queue. This paper introduces the adaptive resource scheduling algorithm based on the ant colony algorithm and the particle swarm algorithm, resulting in an emergency resource allocation and scheduling method for different distributed storage systems. The experimental results show that this method can: effectively schedule the emergency resources of different distributed storage systems, achieve real-time success, and reduce energy consumption, all of which are of great practical significance.

REFERENCES

1. Al-Haija Q.A., Asad M.M., Marouf I., Bakhuraibah A., Enshasy H. 2019. FPGA Synthesis and Validation of Parallel Prefix Adders. *Acta Electronica Malaysia*, 3(2), 31–36.
2. Amine, K. 2019. E-fuel system: A conceptual breakthrough for energy storage. *Science Bulletin*, 64(4), 227–228.
3. Bramasta D., Irawan D. 2019. Tourism Object Mapping Based on Geographic Information System in Baturraden District, Regency of Banyumas. *Acta Informatica Malaysia, 3*(2), 14–18.
4. Chong, L.W., Wong, Y.W., Rajkumar, R.K., Isa, D. 2018. An adaptive learning control strategy for standalone PV system with battery-supercapacitor hybrid energy storage system. *Journal of Power Sources*, 394(15), 35–49.
5. Dzhurik, A.S., Belov, A.M. 2019. The integrated data-acquisition system of the T-11M tokamak. *Instruments and Experimental Techniques*, 62(1), 18–21.
6. Fang, W.W., Ding, S., Li, Y.Y., et al. 2019. OKRA: Optimal task and resource allocation for energy minimization in mobile edge computing systems. *Wireless Networks*, 25(5), 2851–2867.
7. Feng, C., Liao, H.Y., Tian, X.Q., et al. 2018. Model and algorithm for lean principle based deploying emergency resources. *China Safety Science Journal*, 28(6), 185–191.
8. Fu, D.Q., Chen, Z.H., Jian, J., et al. 2019. Research of resource location & allocation model in regional emergency joint action. *Mathematics in Practice and Theory*, 49(6), 30–41.
9. Hammouda, M., Vegni, A.M., Peissig, J., et al. 2018. Resource allocation in a multi-color DS-OCDMA VLC cellular architecture. *Optics Express*, 26(5), 5940–5961.
10. Heinonen, J.S., Luttinen, A.V., Spera, F.J. 2019. Deep open storage and shallow closed transport system for a continental flood basalt sequence revealed with Magma Chamber Simulator. *Contributions to Mineralogy and Petrology*, 174(11), 1–18.
11. Liang, X., Yun, J.F., Wang, Y., et al. 2019. A new high-capacity and safe energy storage system: Lithium-ion sulfur batteries. *Nanoscale*, 11(41), 19140–19157.
12. Naidu, K., Battula, R.B. 2018. Quick resource allocation in heterogeneous networks. *Wireless Networks*, 24(8), 3171–3188.
13. Phuong N.H. 2019. A Short Communication on Reverse Logistics Role in the Supply Chain. *Information Management and Computer Science*, 2(1), 10–14.
14. Xu, J.P., Li, X., Zhao, X.F.2019. A Resource scheduling Improvement Algorithm in Cloud Storage Environment . *Computer Application Research*, 36(7), 2015–2019
15. Yang, L.Q., Chen, Y.Y. 2019. Resource-efficiency improvement based on BBU/RRH associated scheduling for C-RAN. *Wireless Networks*, 25(5), 2805–2815.

16. Yang, Q., Chen, J.M., Liu, J., et al. 2019. Task allocation of emergency rescue workers based on bilateral matching decision. *China Safety Science Journal*, 29(1), 180–186.
17. Zhang, Q., Li, G. 2019. 1A predictive energy management system for hybrid energy storage systems in electric vehicles. *Electrical Engineering*, 01(3), 759–770.
18. Zhang, X.Y., Guo, D.X., An, K., et al. 2020. Secure transmission and power allocation in multiuser distributed massive MIMO systems. *Wireless Networks*, 26(2), 941–954.
19. Zheng, K. 2019. Simulation of emergency resource optimization scheduling for differential distributed storage systems. *Computer Simulation*, 36(7), 415–418.