Resource Scheduling Strategy for Energy Consumption Optimization of Big Data

Yetong Xu

**Note: It's part of my work. It was in Chinese, and I translated it into English.*

ABSTRACT

With the advent of the big data era , the data types and data volume have all exploded. Big data technology brings convenience to daily life , but also poses challenges to resource consumption and environmental protection. Under the context of Chinese goal of " achieving carbon neutral by 2060 " and promoting the " green recovery " of the world economy after the epidemic, it is of great significance and urgent to improve the energy use efficiency of big data system and achieve " green computing ". Most past studies have used big data systems with node isomorphism, which is not consistent with the realistic. Therefore, this paper takes the big data system of heterogeneous nodes as the optimization object. According to the different computing power of each node and the different amount of remaining available resources, and the different demand for resources for different tasks, an ODTC Dynamic Task Combination algorithm is proposed to improve the energy efficiency of big data system. Taking the MapReduce parallel computing model as the experimental scenario, it is verified that the ODTC algorithm has more advantages than the FIFO algorithm in performing tasks efficiently and saving energy.

Keywords: Green computing, energy efficiency, ODTC algorithms, MapReduce parallel computing models

Table of Contents

I. Introduction

(I) Background and Significance

Today, with the rapid development of computer technology, data has exploded in both variety and quantity. According to estimates by IDC[1], global data volume is expected to grow by more than 40% over the next decade, predicting a total of 35ZB by 2020, which is about 30 times the data volume in 2010. At the same time, statistics show that the amount of data generated by humans in the last two years is equivalent to all the data generated before.[2] Alongside the explosive growth in data, data center scales have become increasingly massive, leading to higher energy consumption. The soaring energy consumption of data centers has increased operational costs, and the increasing electricity use and associated carbon emissions contribute to environmental pollution. Data indicates that the energy used by all servers worldwide in 2012 equaled Mexico's national energy consumption in 2007, and China's IT energy consumption accounts for 50% of the country's annual government energy expenditure of 80 billion yuan, with data center energy consumption accounting for 40% of the total IT expenses.

"Big data technology" has become increasingly common in everyday applications, generating tremendous value. For instance, in daily life, major e-commerce platforms use big data technology to analyze users' purchase records, browsing history, and personal information such as age and income based on the purchase of a product. This allows them to infer user preferences and recommend related products. When traveling, people use ride-hailing apps, real-time bus arrival systems, or map services to check traffic congestion. All these conveniences are a result of big data technology. Similarly, during the COVID-19 pandemic in 2020, experts and scholars predicted the spread of the virus using big data, tracked and recorded suspected cases and close contacts, demonstrating the immense value big data technology has for human society.

However, the application of big data technology also poses numerous challenges to social life. In today's society, people are almost constantly online, creating new data at every moment. As big data technology becomes widespread, its development and application pose significant challenges to resource utilization and environmental protection. Despite various big data processing technologies and systems in widespread use, issues like cluster heterogeneity in big data processing systems still exist. During the use of big data platforms, unnecessary energy consumption occurs due to hardware, strategies, and algorithms, leading to serious environmental pollution. This indicates that there is still room to improve the resource efficiency of big data platforms.

Since the Industrial Revolution, human science and technology have rapidly developed, but the severe environmental pollution caused by technological progress has undoubtedly had a massive negative impact on human society, with issues like global warming, accelerated extinction of precious animal and plant species, and an increase in natural disasters. The global COVID-19 pandemic in 2020 once again reminded people of the urgent need for self-reformation, to accelerate the development of green modes of development and lifestyle, and to build an ecological civilization and a beautiful Earth. Therefore, reducing unnecessary energy consumption and environmental pollution during the use of big data technology, improving the resource efficiency of heterogeneous big data systems, and reducing the consumption of redundant resources have become urgent and significant.

On September 22, 2020, Chinese President Xi Jinping delivered an important speech at the 75th United Nations General Assembly, solemnly declaring, "China's carbon dioxide emissions will strive to peak before 2030 and achieve carbon neutrality before 2060. Countries need to establish a new development concept that is innovative, coordinated, green, open, and shared, seize the historic opportunities of a new round of scientific and technological revolution and industrial transformation, promote a 'green recovery' of the world economy after the pandemic, and gather strong and sustainable development forces." We believe that big data technology is at the core of this new round of scientific and technological revolution and industrial transformation. Standing at the historical intersection of the COVID-19 pandemic, addressing the issue of energy efficiency in big data technology will greatly advance China's goal of achieving "carbon neutrality" and also contribute to promoting a "green economic recovery" worldwide.

Thus, with the goal of improving the energy efficiency of big data systems, reducing the environmental pollution caused by excessive energy consumption, "green computing" has become one of the hottest research topics in recent years. There is an increasingly strong market demand for efficient, low-cost, low-energy green computing products. This paper will also focus on "green computing," researching energy optimization strategies for heterogeneous nodes in big data systems.

(II) Main Work

The main work of this paper is as follows:

Current research often focuses on the assumption that all nodes within a big data system are homogeneous, an assumption that does not match the reality of the production environment. From our understanding and study of previous literature, we find that in practical applications, the following situations mainly cause the emergence of heterogeneous nodes in cluster systems:

(1) Heterogeneity in node hardware. As the era progresses, the scale of clusters needed for data processing is becoming increasingly large, making it difficult to ensure that all compute nodes have exactly the same hardware configuration. Even if they start out identical, the maintenance of nodes during operation or the addition of new nodes will gradually lead to heterogeneity.

(2) Whenever a new task is scheduled to enter the cluster for execution, there are likely many other tasks already in progress, meaning that the computational resources currently available to each node differ. For new tasks, the cluster is undoubtedly heterogeneous at that moment.

This paper focuses mainly on the recombination and redistribution of tasks, integrating small tasks into larger ones, and assigning tasks based on resource characteristics, aiming to balance the load across nodes more evenly, minimize idle time, and thereby improve resource utilization efficiency and reduce energy consumption.

In summary, this paper aims to address three main issues: 1. How to assess the characteristics of nodes and determine the available resources of heterogeneous nodes; 2.

How to determine the association between tasks and nodes to achieve dynamic task matching and allocation; 3. How to schedule resources and tasks.

The framework of the discussion in this paper is briefly outlined as follows: The second part introduces the related research on optimizing the energy usage efficiency of big data platforms; the third part introduces the principles of the optimization strategy for heterogeneous nodes; the fourth part provides mathematical support for the optimization algorithm proposed in this paper and conducts a mathematical proof; the fifth part conducts experimental validation of the paper's algorithm based on the MapReduce platform; and the sixth part offers an analytical summary and outlook for future work.

II. Related Work

(I) Parallel Computing Models

With the advent of the Web2.0 era, the volume of information on the internet is growing exponentially, with data generated daily in the unit of terabytes. Traditional single-machine computing modes are no longer able to support such vast data volumes, necessitating the division of computational tasks into smaller, single-machine manageable tasks in a distributed manner. Additionally, most of this data is non-relational and unstructured, presenting a significant challenge to the traditional relational data storage and computation models. This has led to the emergence of distributed computing frameworks and cloud computing. By far, prominent distributed computing frameworks include Hadoop, Storm, Spark, Dryad, among others.

Hadoop is one of the earliest distributed computing frameworks, primarily based on Google's MapReduce programming model. Its open-source implementation is extremely powerful, comprising the MapReduce computation framework and the HDFS storage framework for storing computational data. Hadoop is typically suitable for batch processing, latency-insensitive, or offline data processing tasks. Storm, proposed by Twitter and developed in a Lisp-like language, is a distributed computing framework for real-time big data processing based on stream computing. Storm has characteristics of real-time processing and stream computing, which Hadoop cannot support. This to some extent resolves the issues of significant latency and complex maintenance in the later stages of the Hadoop computing framework. It is generally used for stream data processing and scenarios requiring high real-time performance. Spark, written in Scala and based on Resilient Distributed Datasets (RDD), is designed for iterative computations, where intermediate data caching allows for rapid computation, addressing the inefficiency of Hadoop in processing iterative tasks. However, Spark generally requires a larger memory footprint and is suitable for batch processing and iterative tasks.

Next, MapReduce will be discussed in detail:

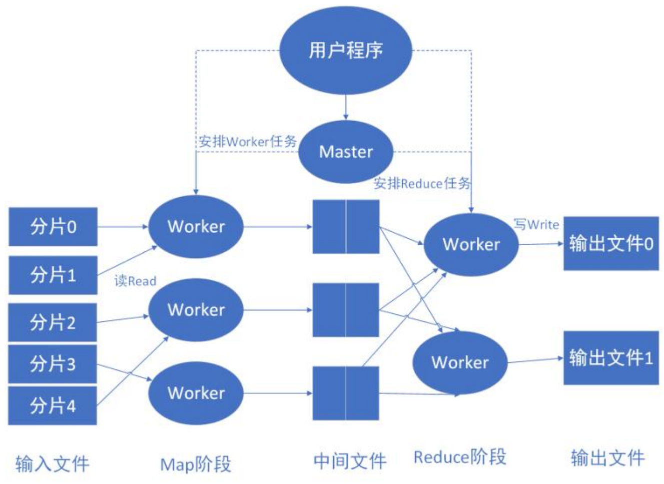


Figure 2-1 MapReduce framework

As illustrated in Figure 2-1, the computational process of the MapReduce model is divided into two stages, Map and Reduce, with user programs implementing Map and Reduce functions through the MapReduce library. The execution steps are summarized as follows: Firstly, the input files from the user program are divided into N data blocks, typically ranging from 16MB to 64MB, as depicted, divided into five shards in the diagram. These shards are then transferred to other machines, where the user program's copies include a Master that schedules Map and Reduce tasks, and multiple Workers that execute tasks, with those executing Map tasks referred to as Mappers and those handling Reduce tasks as Reducers.

Mappers read the data shards, transform them into <key, value> pairs, and after processing by the Map function, output <key, value> pairs as intermediate results, which are cached in node memory. Subsequently, the locations of these intermediate results are reported to the Master through a "heartbeat mechanism", which then informs the Reducers. The intermediate results are periodically written to local disks and partitioned according to the defined number of Reduce tasks. Reducers, upon receiving the locations of the intermediate results, transfer the data from the Mappers, and after sorting, aggregate the <key, value> pairs corresponding to the same key. The results produced are then added to the output file.

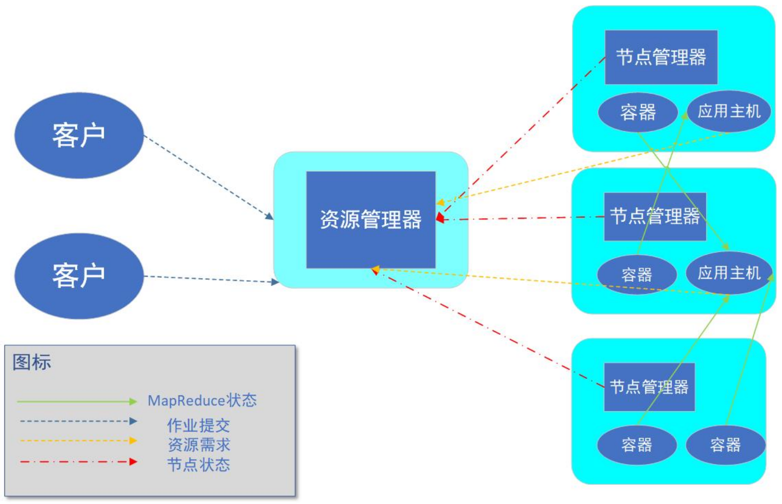


Figure 2-2 Yarn framework

A brief introduction to the Yarn architecture follows as shown in Figure 2-2. Yarn (Yet

Another Resource Negotiator) architecture, having evolved over a long period from the Hadoop 1.0 version, addresses the single-point failure, the overwhelming task load of the JobTracker, memory overflow (resource allocation only considering MapReduce task numbers, not CPU, memory), and unreasonable resource partitioning (compulsory division into slots) existed in the original MapReduce. Its main improvement lies in separating the resource management function from MapReduce to form Yarn, turning it into a pure resource management and scheduling framework. The MapReduce 2.0 version, therefore, does not handle resource scheduling and management, which is provided by Yarn.

The Yarn system primarily consists of one ResourceManager and several NodeManagers. The ResourceManager is responsible for allocating computing resources available in the system to different applications, which are encapsulated and executed within containers. Users allocate computing resources such as CPU, memory, network bandwidth, etc., to these containers according to actual needs. The NodeManager takes care of resource management on individual nodes, monitoring container resource usage and handling commands from the ResourceManger and ApplicationMaster. The ResourceManager contains Scheduler and ApplicationManager components, with the Scheduler allocating resources to running applications based on available resources and queue task conditions, and the ApplicationManager accepting task applications, arranging specific nodes, and starting the ApplicationMaster of the containers executor on the nodes. The ApplicationMaster is responsible for program monitoring and tracking.

(II) Green Computing

As big data technology further develops, data volumes will inevitably become much larger than today. Enormous data volumes can lead to resource waste and environmental pollution, hence the concept of "green computing" was proposed.

Green computing broadly refers to computer and related resource usage in an environmentally responsible manner, including the use of efficient and energy-saving CPUs, servers, and peripherals, reducing resource consumption, and proper handling of electronic waste. The U.S. Environmental Protection Agency pioneered the path of green computing in 1992 with the voluntary "Energy Star" labeling program aimed at improving hardware efficiency. Similar initiatives have also been launched in Asia and Europe.

However, governmental regulations and documents are only part of the green computing philosophy. To further advance and implement green computing, manufacturers and users must actively take measures in real life. With "green computing" as the goal, many manufacturers initially adopted an external power manager chip to monitor the working status of the system, reducing component energy states when the computer is idle. When designing hardware aligned with green computing principles, material selection considers the ease of disposal; tasks related to computing are completed within continuous time blocks, stopping hardware work at other times; LCDs replace CRT monitors; paper usage is reduced with proper recycling of waste paper; electronic waste is handled in accordance with government and regional regulations; alternative energy sources are used within computer workstations, servers, networks, and data centers.

(III) Previous Research on Resource Efficiency Improvement in Big Data Systems

A task's energy efficiency is often reflected by the ratio of the effectiveness of tasks completion to energy consumption, necessitating a balance between the two. Previous research indicates two main schools of thought: [3-5]one advocating that better performance requires more energy consumption, and the other advocating that performance improvements can coexist with high resource use efficiency[6-7], i.e., achieving good performance using as few resources as possible. This paper aims to align with the latter, striving to enhance performance while minimizing resource use.

According to a 2018 paper by Song Jie et al.[8] on optimizing energy usage efficiency in big data platforms, they support the viewpoint that performance can be improved while minimizing resource use. Their approach involves setting a "Best Resource Ratio" (BRR), suggesting that when resources allocated to tasks meet a certain ratio, the efficiency of resource and energy use is maximized. Their method is divided into "task planning" and "resource scheduling"; the "task planning" algorithm is responsible for selecting appropriate tasks from the task list and assigning them to a suitable node, while the "resource scheduling" node allocates resources to tasks based on the BRR. They also mathematically proved their point.

In addition to this, many scholars have analyzed how to improve the resource utilization efficiency of big data systems from the perspectives of node data load balancing, data localization, and cluster heterogeneity, based on the widely applied MapReduce framework. To address the impact of data skewness on the resource utilization efficiency of big data systems, researchers have added two pre-run sampling and counting tasks to propose skew optimization for Theta joins. Similarly, Kwon and others [9] designed the SkewReduce system, which optimizes spatial features through preprocessing to extract data partitioning and sampling programs for applications. It is evident that the two algorithms mentioned above can alleviate the problem of data skew to a certain extent, but the downside is the apparent overhead of setting up pre-run jobs, which are only applicable to certain specific applications. Some scholars have chosen to collect data information during task execution. According to [10], Ibrahim and others have divided the intermediate data into more partitions than there are Reducer nodes. After the Map tasks are completed, the intermediate data is allocated to the Reducer nodes using the bin-packing method. On this basis, they further proposed the TopCluster algorithm to obtain an approximate distribution of input data. Each Map task in this algorithm samples the largest cluster of keys and their frequencies and transmits the collected information to the Master for aggregation. The Master can then calculate the partitioning cost more accurately based on this information. However, the disadvantage of the above algorithms is that all Map jobs must be completed before the Reducer nodes can start to obtain the required intermediate data, which completely sacrifices the parallel computing design of the Map and Reduce phases in the Hadoop framework.

Some experts and scholars have also considered node data skewness and jobs' data localization simultaneously to optimize big data systems. Hsu and others [11] decided to dynamically allocate data before the Map jobs were executed to increase the degree of jobs' data localization. At the same time, virtual machines are used during the Reduce phase to balance the load across all Reducer nodes. In [12], Ibrahim and others proposed the LEEN algorithm, which first considers data localization when allocating intermediate data after the map phase, followed by load balancing between computing nodes. The CLP algorithm proposed in [13] is the exact opposite of the LEEN algorithm; after obtaining the distribution of intermediate data, the stage of allocating intermediate data prioritizes the load balancing between computing nodes, followed by jobs' data localization. From the existing research, we can see that prioritizing node data skewness or data localization can have different impacts on different job types, thereby significantly affecting job runtime. Compared to this, the algorithm proposed in this paper is more straightforward and does not have issues with varying execution results due to different internal algorithmic structure priorities.

Compared to the directions of two research mentioned above, studies on system heterogeneity have become a popular research direction in improving big data system performance in recent years.

In [14], the LATE algorithm selects the task with the longest remaining time based on the completion rate of tasks and arranges its replicas on other faster-running nodes, thus solving the issue of inconsistent task completion times caused by system node heterogeneity. Our ODTC algorithm also takes into account the different computing capabilities of heterogeneous nodes, assigning different tasks to different nodes to improve resource utilization efficiency. In [15], Aysan Rasooli and others proposed a dynamic scheduling algorithm that determines the optimal scheduling strategy based on the average execution time and arrival rate of tasks, distributing tasks to different queues based on their resource requests, and designed a model to predict the type of tasks to place them into different queues. The algorithm proposed in [16] classifies all tasks according to their computational intensity and I/O intensity, and the scheduling node allocates resources according to the characteristics of tasks in different types of queues. Following this, there are algorithms such as HAPS [17] that allocate resources based on historical execution time of tasks, and algorithms that have obtained the relationship between the energy consumption and load of computing nodes through monitoring methods, applying the power limit of control feedback theory to propose resource allocation schemes for MapReduce tasks.

Addressing the shortcomings of various algorithms proposed in numerous studies, the research strategy of Chen Lei and others in [18] made significant improvements. In response to data skewness, they proposed the load-balancing strategy MRSIM, adding a load monitoring module to each DataNode to achieve load balancing by transferring computational tasks and data. For the issue of balancing data localization with node data skewness, they proposed a MapReduce computation model optimization strategy based on the Bayes classifier, named BAPM (naive Bayes classifier-based Partitioner for MapReduce). This strategy includes two parts: LPS (data Locality Prior to data Skew) and SPL (data Skew Prior to data Locality), considering data localization and data skewness in opposite orders. After addressing data skewness and data localization, and further considering cluster heterogeneity, they proposed the PIY strategy (Partitioner In YARN) for the MapReduce model. This strategy, based on parallel reservoir sampling, quickly and accurately captures the distribution of job input data. Then, utilizing the BASH algorithm, it takes into account the factors of heterogeneity, data skewness, and data localization before assigning jobs to the Reducer nodes. Following the introduction of all these previous studies, Table 2-1 was created to display the advantages and disadvantages of each algorithm in previous research.

Table 2-1 Advantages and disadvantages of various algorithms in previous studies

Method	Advantage	Disadvantage
SkewReduce	Can alleviate data skewness	Setting up pre-run jobs has noticeable overhead and are only applicable to certain specific applications
TopCluster	Can alleviate data skewness	Has to wait for all Map jobs to be completed before the Reducer nodes can start processing the intermediate data. This approach entirely sacrifices the parallel computing design of the Map and Reduce phases in the Hadoop framework, which is a significant limitation.
CLP、LEEN	Consider both data skewness and data localization to enhance resource utilization efficiency	Prioritizing either data skewness or data localization will affect the final outcome, creating ambiguity
LATE、HAPS	Improve resource utilization efficiency from the angle of Node Heterogeneity	
BRR	Simple, understandable, and allocate tasks based on the best resource distribution ratio	Not taking the heterogeneity of nodes into consideration

The research by Chen Lei and others on the performance and energy optimization methods for big data processing systems is quite comprehensive. However, it is limited to the MapReduce parallel computing model, which is not sufficient. In contrast, the ODTC algorithm proposed in this paper is not only applicable to the MapReduce model but also to a variety of other parallel computing models, making it more universally applicable.

Moreover, apart from Chen Lei's research, the vast majority of the analysis of the above literature is based on the node homogeneity in big data system, not taking into account the impact of node heterogeneity. This is where our research differs. Our focus is on optimizing the resource usage efficiency of "heterogeneous nodes." We need to further consider the characteristics of each node, and after assessing the different computational capabilities of each node, we take into account the heterogeneity of the nodes. Using the ODTC algorithm, we assign the most suitable tasks based on the characteristics of resources available on each node, thereby reducing energy consumption. I believe that incorporating "node characteristics" is one of the innovative aspects that makes our approach superior to the ones mentioned above.

III. Principles of Heterogeneous Node Optimization Strategies

(I) Scenario Constraints and Assumptions

Following the discussion on "Related Work" in the second section, it becomes evident that most studies on the performance and energy usage optimization of big data systems are based on the assumption that all nodes within a big data system are homogeneous. However, assuming homogeneity of all nodes in a big data system clearly does not align with real-world production environments. Taking the widely used MapReduce parallel computing model as an example, further learning from the literature reveals that issues such as cluster heterogeneity, data skewness, and low degrees of data localization can significantly impact the performance of MapReduce jobs. As previously mentioned, the causes of system node heterogeneity can be categorized into two types: hardware heterogeneity of the nodes themselves, and the different available computing resources for each node when new tasks are scheduled to execute in the cluster, making the cluster heterogeneous for the newly added tasks.

This paper focuses on heterogeneous system nodes, specifically when new tasks enter the system, different nodes have varying available computing resources. Different nodes have varied hardware configurations, such as CPU, and differing data processing capabilities, meaning different computational capabilities for different nodes. The various differences in heterogeneous nodes lead to differences in execution efficiency and time when different tasks enter different nodes.

Firstly, we set some limitations and assumptions for the scenario in which our ODTC algorithm is applicable.

1. We assume an ideal state for our algorithm analysis, considering only the system overhead generated during the data processing and ignoring all other factors such as data transmission.

2. We assume a "delayed feedback" effect in the implementation of our algorithm, meaning that each node uses all resources for computation when running tasks, resulting in very high resource utilization efficiency. The algorithm starts scheduling and data migration between idle nodes and nodes with heavier loads only when there are nodes in the system that have completed tasks and entered an idle waiting state.

(II) Conceptual Clarifications

Next, we clarify some concepts involved in this paper.

1. A node's computing resources can include the number and frequency of central processing unit (CPU) cores, as well as the bandwidth of hard drives and network

transmission on the system nodes. A system may have many different nodes, usually with multiple tasks running simultaneously on these nodes. The execution of node tasks requires computing resources, and the use of these resources ultimately results in energy consumption.

2. The energy efficiency of a big data processing platform typically refers to the ratio of the platform's data processing performance to the energy consumed during resource usage. Thus, when resources are fully utilized, i.e., when there are fewer redundant resources, less energy is wasted, and the efficiency of energy utilization is higher.

3. During task execution, computing nodes will have idle resources. For example, CPUs are suspended while waiting for input/output devices or network transmission, and the energy consumed by these suspended CPUs is considered a waste of idle resources.

4. The goal of this paper is to improve the resource utilization efficiency of big data processing systems, thereby reducing unnecessary energy consumption. In other words, we aim to minimize idle resources. If all energy consumption is used for data processing in big data systems, we consider this to be the highest energy utilization efficiency.

From our earlier definition of energy efficiency as the ratio of the big data platform's data processing performance to the energy consumed in the data processing, it is evident that merely allocating more computing resources to tasks cannot improve energy efficiency. Instead, it would lead to the emergence of idle resources and consequently waste energy.

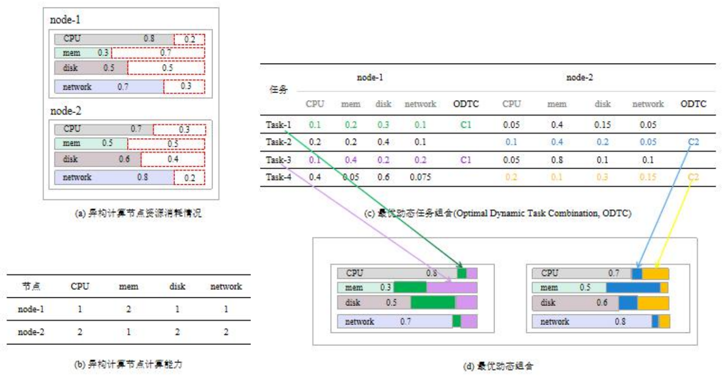
(III) ODTC Algorithm

Based on this, our paper takes into account factors such as the available resources of each computing node in heterogeneous systems, different computational capabilities of nodes, and varying demands of different tasks for computing resources. We adopted a method of recombining and redistributing tasks, named the Optimal Dynamic Task Combination (ODTC) algorithm, to minimize the occurrence of idle resources and maximize resource utilization efficiency.

As shown in Figure 3-1, we present a simple example of the Optimal Dynamic Task Combination (ODTC) algorithm. Figure 3-1-(a) shows the consumed resource situation on nodes 1 and 2; Figure 3-1-(b) shows the computational capabilities of nodes 1 and 2; Figure 3-1-(c) shows four tasks waiting to be executed in the queue and their respective demands for node resources; Figure 3-1-(d) illustrates the task allocation after applying the Optimal Dynamic Task Combination algorithm.

The ODTC algorithm operates as follows:

As shown in Figures 3-1-(c) and 3-1-(a), based on the different demands of tasks 1-4 on nodes 1 and 2 for node resources, and the remaining available resources on nodes 1 and 2, recombining tasks 1 and 3 and allocating them to node 1, and tasks 2 and 4 to node 2, results in the least idle resources on each node. This maximizes energy utilization efficiency while also meeting the resource needs of each task. In simple terms, different nodes have varying computational capabilities, leading to different resource requirements, efficiency, and execution times for the same task on different nodes. On this basis, considering the available remaining resources on different nodes, tasks are recombined and dynamically allocated to nodes in the system.



节点	CPU	mem	disk	network
node-1	1	2	1	1
node-2	2	1	2	2

任务	node-1					node-2				
	CPU	mem	disk	network	ODTC	CPU	mem	disk	network	ODTC
Task-1	0.1	0.2	0.3	0.1	C1	0.05	0.4	0.15	0.05	
Task-2	0.2	0.2	0.4	0.1		0.1	0.4	0.2	0.05	C2
Task-3	0.1	0.4	0.2	0.2	C1	0.05	0.8	0.1	0.1	
Task-4	0.4	0.05	0.6	0.075		0.2	0.1	0.3	0.15	C2

Figure 3-1 A concise example of using ODTC algorithm to improve resource utilization

IV. Optimal Dynamic Task Combination

(I) Energy Consumption Assessment Model

To enhance the efficiency of resource usage in big data systems, we first need to establish a model for assessing energy consumption. The variables in the model are defined as follows:

Definition 1 Resource Quantity: Referring to the study by Song Jie[7] et al. in 2018, we found that the definition of resource quantity is set as a variable with "time series" characteristics, able to reflect the instant resource quantity of a node at any given moment, which is a significant advantage. However, the study considered only CPU, hard disk, and network resources, excluding memory. In our model, memory will also be taken into account. We assume at time t, n types of resources are allocated to a task:

(1) $r_i(t)_c$: The resource quantity of the i-th resource consumed by the task.

(2) $r_i(t)_a$: The resource quantity of the i-th resource allocated to the task.

(3) $\Delta r_i(t) = r_i(t)_a - r_i(t)_c$: The remaining resource quantity of the i-th resource allocated to the task.

(4) $/r_i/$: The total resource quantity of the i-th resource on the node.

We consider four types of resources: CPU, memory (mem), hard disk (disk), and network (network). When evaluating the resource quantity of different resources, we have to standardize the units for these four resources. The unit of resource consumption is set as MB/S. The hard disk resource consumption during data processing is defined as $r_d(t)$, and the network resource consumption is defined as $r_n(t)$。$r_d(t)$、$r_n(t)$, representing the bandwidth of hard disk and network resource transmission, respectively. The memory resource consumption is defined as $r_m(t)$, representing the amount of tasks being executed in memory at that time. The CPU resource consumption is defined as $r_c(t)$, referring to the number of tasks that the benchmark data processing tasks are processing. We use a linear search algorithm as the benchmark data processing task, the idea of which is to iterate through every element in the list to find the target element until either the target element is found or the list element is exhausted.

Definition 2 CPU Resource Quantity: We define f(t) as the frequency of the CPU at time t (GHz) and w(t) as the utilization rate of the CPU at time t. The resource quantity used by the CPU at time t is α*f(t)*w(t) (MB/s). The coefficient α represents the amount of data (in MB) that a 1GHz CPU can search per second using the linear search algorithm. For example, if we say 200 MB/s of CPU resources are allocated to a task, it essentially means that the task has been allocated the computational capability to linearly search 200 MB of data per second.

Definition 3 Throughput: We use the variable $v_i(t)$ to represent the throughput of the i-th resource, which refers to the speed of processing data using the i-th resource when a resource of size $r_i(t)_a$ is allocated, and the actual consumption of the i-th resource is $r_i(t)_c$. For instance, the throughput of the CPU $v_c(t)$ refers to the amount of data being processed by the CPU at time t, given the allocated $r_c(t)_a$ and actually used $r_c(t)_c$ resource quantities, with memory following the same principle. The throughput of the hard disk $v_d(t)$ refers to the amount of data being read or written by the hard disk at time t, under the allocated $r_d(t)_a$ and actual resource consumption $r_d(t)_c$, and the same applies to network resources.

Definition 4 Energy Consumption: For the definition of energy consumption estimation in computer systems, we refer to the research published in 2015 by Yu Junyang[19] et al., which focused on establishing accurate computer system energy consumption estimation models to estimate the enormous energy consumption of service nodes in large data systems. Aiming for "sufficient representativeness, easy monitoring, and minimal parameters," they standardized data units and established a correlation matrix between variables, choosing Processor Time (the percentage of time the processor spends executing non-idle threads), Memory Used (actual memory utilization rate), and Page Faults (average number of error pages) as the basic parameters for modeling computer system energy consumption.

In the modeling process, computer systems are divided into computation-intensive states and non-computation-intensive states. The former's energy consumption changes are dominated by CPU , with memory and hard disk under low load and minimal impact on energy consumption. The latter's energy consumption is determined by CPU, memory, and hard disk together. A threshold of $Memory_{utilization} \leq 40\%$ AND Pf ≤ 1094 is used to define computation-intensive and non-computation-intensive states, where $Memory_{utilization}$ represents memory utilization rate, and Pf represents the average value of Page Faults/s. The model is established as follows:

$$\begin{cases} y = e^{5.6851}x^{0.1257}(Memory_{utilization} \le 40\% \text{AND Pf} \le 1094) \\ y = 198.20737 - 6.23549x_1 + 200.28839x_2 + 0.0003348x_3(Memory_{utilization} > 40\% \text{OR Pf} > 1094) \end{cases}$$

(4-1)

In the model, x represents the percentage of time the CPU spends on non-idle threads, while x_1, x_2, and x_3 respectively represent the percentage of CPU time spent on non-idle threads, the memory utilization rate, and Page Faults. y refers to the energy consumption of the computer system.

Definition 5 Energy Efficiency of a Task: For a task at time t, its energy efficiency EE(t) is defined as the ratio of the throughput V(t) to the energy consumption E(t) of the allocated resources.

$$EE(t) = \sum_{i=1}^{n} EE_i(t) \sum_{i=1}^{n} \frac{v_i(t)}{p_i(t) * {r_i(t)_a}/{|r_i|}}$$

(4-2)

In this context, $p_i(t)$ represents the energy consumption of the i-th component at time t, as the system energy consumption estimation model shown in Definition 4. The unit for energy efficiency is MB/Joule. The ratio ${r_i(t)_c}/{r_i(t)_a}$ represents the variable that needs optimization in our study — **Resource Utilization Efficiency**. When the value of ${r_i(t)_c}/{r_i(t)_a}$ equals 1, i.e., $r_i(t)_c = r_i(t)_a$, it indicates the highest resource utilization efficiency.

(II) Mathematical Justification of the ODTC Algorithm

After establishing a mathematical model for energy consumption assessment and clarifying our optimization goal of resource utilization efficiency ${r_i(t)_c}/{r_i(t)_a}$, we will construct a mathematical model for the distribution of energy consumption. In this paper, we define the horizontal axis as N_i (1≤i≤n), representing the i-th node in the system, and the vertical axis as T_j(1≤j≤m), representing the j-th task that needs to be executed in the system. We assume P_{nm} represents the energy consumption of the m-th task on the n-th node in the system, with the specific method of energy consumption assessment as described in Definition 4. We assume R_{nm} represents the resource consumption of the m-th task on the n-th node in the system. A matrix model is established, as illustrated in Figures 4-1 and 4-2:

$$\begin{array}{c} \\ \mathrm{T}_1 \\ \mathrm{T}_2 \\ \mathrm{T}_3 \\ \mathrm{T}_4 \\ \\ \end{array}\begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_4 & \end{array} \\ \begin{bmatrix} 2 & 3 & 5 & 1 & \dots \\ 1 & 2 & 7 & 8 & \dots \\ 3 & 3 & 1 & 2 & \dots \\ 5 & 1 & 7 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \end{array}$$

Figure 4-1 Matrix of node task energy consumption

$$\begin{array}{c} \\ \mathrm{T}_1 \\ \mathrm{T}_2 \\ \mathrm{T}_3 \\ \mathrm{T}_4 \\ \\ \end{array}\begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_4 & \end{array} \\ \begin{bmatrix} 0.2 & 0.3 & 0.5 & 0.1 & \dots \\ 0.1 & 0.2 & 0.7 & 0.8 & \dots \\ 0.3 & 0.3 & 0.1 & 0.2 & \dots \\ 0.5 & 0.1 & 0.7 & 0.1 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \end{array}$$

Figure 4-2 Matrix of node task resource consumption

In the matrix of node task energy consumption shown in Figure 4-1, we aim to identify a series of minimum values, ensuring that at least one value is chosen from each column of the matrix. This means that for each task, we need to find a corresponding node that minimizes its energy consumption. However, the elements in the matrix of node task energy consumption must correspond one-to-one with the elements in the matrix of node task resource consumption. While aiming to minimize the selected elements in the matrix of node task energy consumption, we must also satisfy a constraint condition: for each column in the matrix of node task resource consumption, the sum of the elements corresponding to the chosen elements must be less than or equal to the remaining available resources of that node.

We construct a directed weighted graph based on the "matrix of node task energy consumption", as shown in Figure 4-3.

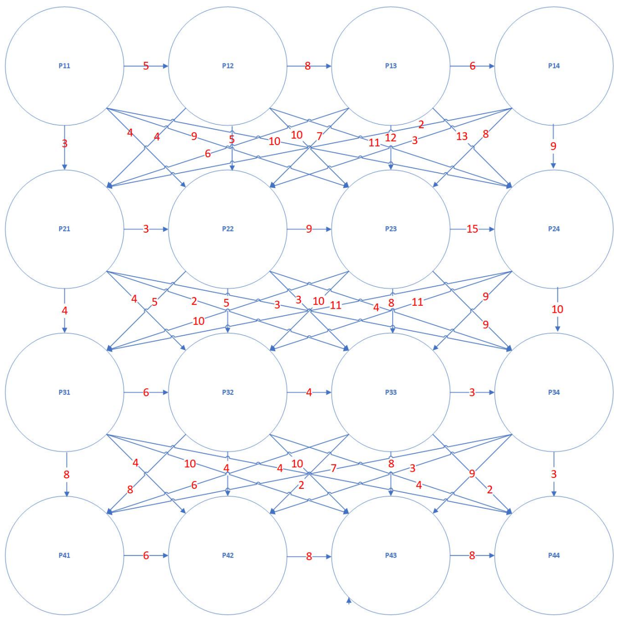
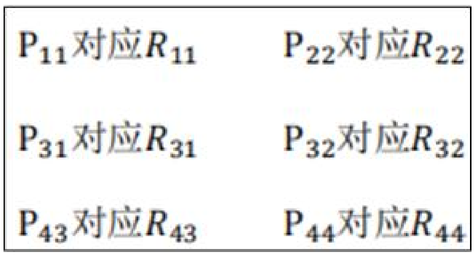


Figure 4-3 Directed weighted graph for matrix of node task energy consumption

In this directed weighted graph, there are 16 vertices P_{11}- P_{44} interconnected as illustrated. Let P_{ij} (1≤i≤4, 1≤j≤4) be a vertex in the graph, representing the energy consumption of the i-th task executed on the j-th node. Every time we choose the node with the minimum energy consumption for the $i-1$ -th task, we must be able to move from the chosen vertex to the next row to select the node with the least energy consumption for the next task i-th. Therefore, all vertices between two adjacent rows are directly connected. The weight of each edge in the graph is calculated by adding the task energy consumptions of the two connected vertices. For example, the weight of the edge connecting vertices P_{11} and P_{12} is the sum of their respective energy consumptions $2+3=5$.

Using the obtained directed weighted graph, we set P_{11} as the starting point and P_{44} as the endpoint, and attempt to find the shortest path in the graph using an appropriate algorithm. The vertices included in this shortest path are considered optimal in terms of minimizing energy consumption while satisfying the resource demands for task execution. It's important to note that both the node task energy consumption matrix and the node task resource consumption matrix describe the same set of nodes: the former shows the energy consumption of these nodes when executing tasks, while the latter shows the resource consumption during task execution. Therefore, the vertices selected in the shortest path of the directed weighted graph Figure 4-3 should correspond one-to-one with the nodes in the node task resource consumption matrix Figure 4-2, as illustrated in Figure 4-4. The sum of the elements in each column of the matrix corresponding to the nodes in the shortest path should be less than the remaining available resources of each node.



P_{11}对应R_{11}	P_{22}对应R_{22}
P_{31}对应R_{31}	P_{32}对应R_{32}
P_{43}对应R_{43}	P_{44}对应R_{44}

Figure 4-4 The vertices correspond to the nodes of the resource consumption matrix

In solving the shortest path problem of the directed weighted graph, this paper considers both the Dijkstra and Bellman-Ford algorithms. The Dijkstra algorithm, proposed by the Dutch computer scientist Edsger Dijkstra in 1959, and the Bellman-Ford algorithm, established by Richard Bellman and Lester Ford, are both designed to solve the single-source shortest path problem. While the Bellman-Ford algorithm has the advantage of handling graphs with negative edge weights, our directed weighted graph, as shown, does not contain edges with negative weights. Therefore, we choose the Dijkstra algorithm for finding the shortest path in our directed weighted graph. We then introduce our innovative algorithm, ODTC (Optimal Dynamic Task Combination).

Algorithm 4-1 ODTC Algorithm

```
Input: G(V,E): The directed weighted graph to be solved;
Output: The shortest distances from the start point s to each vertex; the parent vertices in the shortest paths;
1. Dijkstra(G,w,s)
2. Initialize-single-source(G,s)  // Initialize the single-source directed weighted graph
3.     For each vertex v ∈ G.V
4.         v.d = ∞        // Initially assume the shortest distance from s to all vertices is ∞
5.         v.π = NIL      // Initialize the parent node of all vertices as NIL
6.     s.d = 0            // The shortest distance from the start point s to vertex s is set to 0
7. S = ∅       // Initialize an empty set S to store vertices whose shortest paths are determined
8. Q = G.V     // Initialize set Q containing all vertices G.V of the graph
9. While Q ≠ ∅
10.        u = Extract − min(Q)    // Select the vertex with the minimum value in set Q
```

```
11.         $S = S \cup \{u\}$              // Include vertex u in set S
12.         For each vertex v $\in$ G.Adj[u]
13.             Relax(u,v,w)       // Perform relaxation operation on each vertex
14.             If  $v.d > u.d + w(u,v)$     // A shorter shortest distance is found
15.                 $v.\pi = u$        // Replace the parent vertex
16.                 End If
17.         End For
18.Print-path(G,s,v)   // Print and display the shortest path
19. if  v == s
20.     print s
21. elseif  v.π == NIL   // The parent vertex of vertex is empty
22.             Print "no path from" s "to' v "exists"
23.else Print-path(G,s,v.π)
24.         print  v
25.   End While
```

Algorithm 4-1 ODTC algorithm pseudo-code

The implementation process of the ODTC algorithm is shown in Table 4-1. Lines 2-8 describe the initialization of the directed weighted graph, assuming that the shortest distance from the start point s to all vertices is ∞ and the parent nodes of all vertices is NIL. An empty set S is initialized to store vertices whose shortest paths have been determined, and the set Q contains all vertices G.V of the graph. Lines 9-17 detail the process of finding the shortest path, where the algorithm selects from the set Q, which contains all vertices, the vertex with the shortest distance to s to include in set S. This step is followed by the "relaxation operation" on adjacent vertices of the chosen vertex, i.e., trying to find a better path than the existing shortest path, replacing the original path if a better one is found. The algorithm finally returns two arrays, one representing the shortest distances from the start point s to each vertex, and the other indicating the parent nodes of each vertex in the shortest path.

After applying the Optimal Dynamic Task Combination (ODTC) algorithm to the directed weighted graph shown in Figure 4-3, we obtain Figure 4-4:

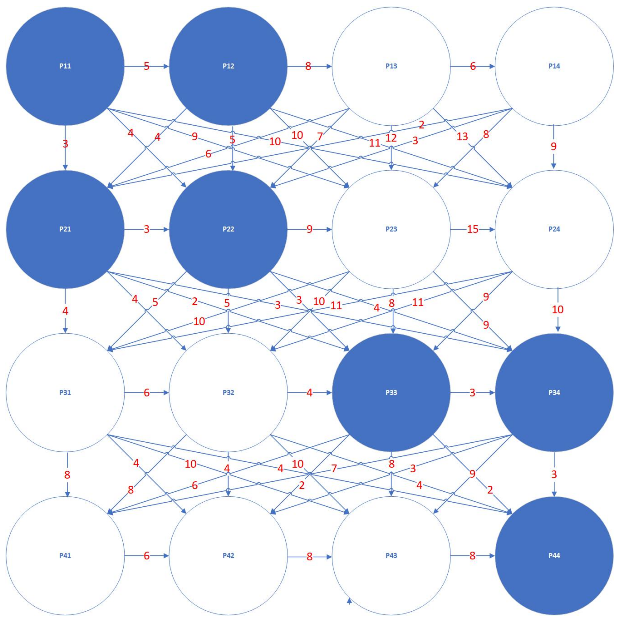


Figure 4-4 The shortest path obtained by ODTC algorithm

As shown in Figure 4-4, after applying the Optimal Dynamic Task Combination (ODTC) algorithm, the shortest path chosen includes the vertices P_{11}、P_{12}、P_{21}、P_{22}、P_{33}、P_{34}、P_{44} from the directed weighted graph. As previously mentioned, these 7 selected vertices correspond to the nodes in the node task resource consumption matrix as R_{11}、R_{12}、R_{21}、R_{22}、R_{33}、R_{34}、R_{44}, and they must satisfy the following condition:

$$R_{11} + R_{21} < N_1 \text{ remaining available resources;}$$

$$R_{12} + R_{22} < N_2 \text{ remaining available resources;} \quad (4\text{-}3)$$

$$R_{33} < N_3 \text{ remaining available resources;}$$

$$R_{44} < N_4 \text{ remaining available resources;}$$

When the vertices included in the chosen shortest path also meet the above conditions, we achieve the optimal dynamic task combination. In this situation, each task can acquire the necessary resources for execution while minimizing the system's energy consumption. If the chosen vertices do not meet the above condition, a sub-optimal allocation scheme is selected as the optimal dynamic task combination.

V. Experiment and Analysis

(I) Experimental Setup

The ODTC (Optimal Dynamic Task Combination) algorithm was validated using the MapReduce parallel computing model, and its energy efficiency was compared with that of the FIFO algorithm to highlight the advantages of the ODTC algorithm.

The experimental environment is detailed in Table 5-1:

Table 5-1 Experimental environment

	Description
Hardware Resources	1、 DL388Gen10 Xeon3204 6-core 1.9G, 16GB RAM, 600GB 10K SAS disk 2、 DL388Gen10 Xeon3204 12-core 1.9G, 16GB RAM, 600GB 10K SAS disk 3、 DL388Gen10 Xeon3204 10-core 2.2G, 16GB RAM,480GB SSD
Operating System	CentOS 7, Linux 5.3.0-28-generic
MapReduce	1 master node, 10 compute nodes, Hadoop 2.0

(II) Experimental Process

1. Execution Time Comparison

A total of 130 tasks were executed, encompassing CPU-intensive (WordCount) and I/O-intensive (Map-Side-Join) tasks. The time taken by the ODTC and FIFO algorithms to execute tasks at different CPU utilization levels is shown in Table 5-2. Graphical representation of this data is provided in Figure 5-1.

Table 5-2 Time Required for ODTC and FIFO to Execute Tasks at Different CPU Utilization Rates (unit: seconds)

CPU Utilization	**ODTC**	**FIFO**
0	645	660
30%	582	652
50%	562	655
70%	576	658

100%	641	661

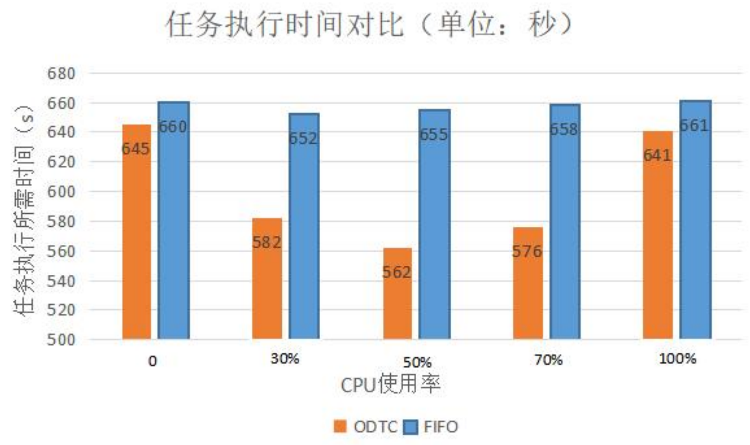


Figure 5-1: Execution Time of Tasks for ODTC and FIFO at Different CPU Utilization Rates (unit: seconds)

As depicted in Figure 5-1, the time required to execute tasks using the FIFO algorithm is longer at all levels of CPU utilization (0%, 30%, 50%, 70%, 100%) compared to the ODTC algorithm. Particularly at CPU utilization levels of 30%, 50%, and 70%, the ODTC algorithm significantly outperforms the FIFO algorithm in terms of execution time. The shorter execution time of tasks validates the significant efficiency improvement of the ODTC algorithm.

When the CPU utilization is neither 0% nor 100%, the big data system has a more diverse set of resources. The ODTC algorithm's advantage in dynamically matching tasks to nodes based on the nodes' available resources and varying resource demands of different tasks becomes more pronounced. Consequently, compared to the FIFO algorithm, the ODTC algorithm not only shortens task execution time but also enhances execution efficiency.

2. Energy Consumption Comparison

The energy consumption of the ODTC and FIFO algorithms in executing tasks at different levels of CPU utilization is detailed in Table 5-3, and a graphical representation is provided in Figure 5-2.

Table 5-3: Energy Consumption of ODTC and FIFO in Executing Tasks at Different CPU Utilization

Rates (unit: kilojoules)

CPU Utilization	ODTC	FIFO
0	566	550
30%	577	623
50%	591	638
70%	623	651
100%	807	812

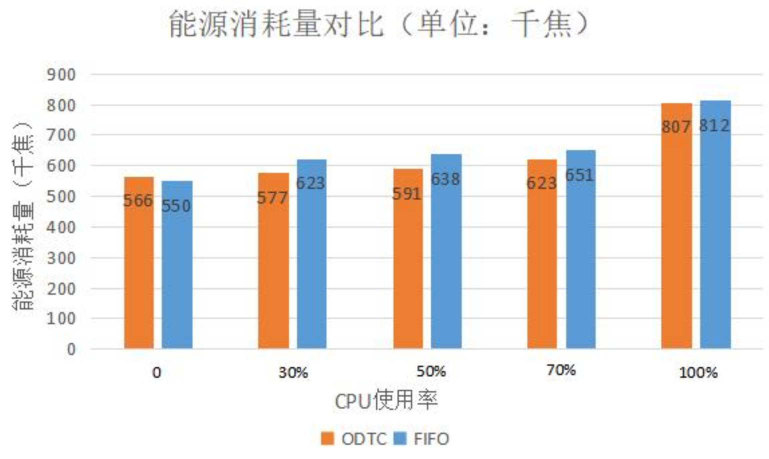


Figure 5-2: Energy Consumption of ODTC and FIFO at Different CPU Utilization Rates (unit: kilojoules)

Figure 5-2 illustrates the energy consumption for executing tasks using ODTC and FIFO algorithms at varying levels of CPU utilization. Notably, when CPU utilization is equal to 0%, the FIFO algorithm consumes less energy than the ODTC algorithm. This is attributed to the limited variety of resources available in the big data system at this utilization level, where the energy spent by ODTC in matching tasks to nodes outweighs the energy savings from the algorithm's optimization. Therefore, when CPU utilization is equal to 0%, ODTC consumes more energy compared to FIFO. However, at CPU utilization levels of 30%, 50%, and 70%, the energy consumption of the ODTC algorithm is consistently lower than that of the FIFO algorithm. Thus, it is evident that at various levels of CPU utilization, the energy consumption of the ODTC algorithm is less than that of the FIFO algorithm.

3. Energy Consumption Comparison for Different Numbers of Tasks

The energy consumption of the ODTC and FIFO algorithms when executing different numbers of tasks is shown in Table 5-4. The data from Table 5-4 is graphically represented in Figure 5-3.

Table 5-4: Energy Consumption of ODTC and FIFO for Different Numbers of Tasks (unit: kilojoules)

Number of tasks	**ODTC**	**FIFO**
10	230	253
50	583	609
100	801	818
150	956	971
200	973	995

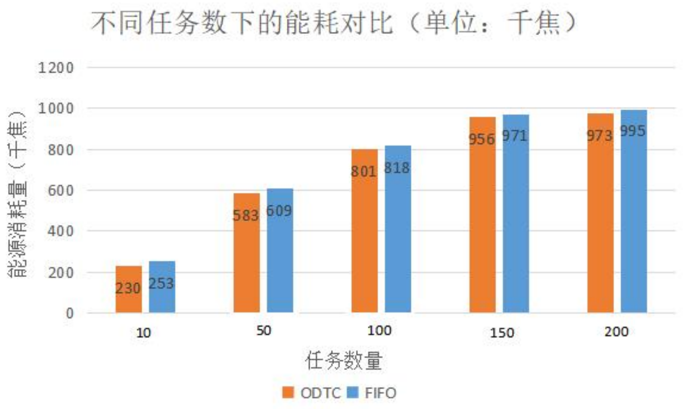


Figure 5-3: Energy Consumption of ODTC and FIFO for Different Numbers of Tasks (unit: kilojoules)

Figure 5-3 presents the energy consumption for executing tasks using the ODTC and FIFO algorithms with varying numbers of tasks. The energy consumption of the ODTC algorithm is superior to that of the FIFO algorithm when the number of tasks is 10, 50, 100, 150, and 200. This demonstrates that regardless of the number of tasks, the ODTC algorithm consistently consumes less energy than the FIFO algorithm.

In summary, the ODTC algorithm outperforms the FIFO algorithm in terms of shorter task execution time and lower energy consumption at varying levels of CPU utilization. Moreover, at different task quantities, the energy consumption for task execution using the ODTC algorithm is less than that of the FIFO algorithm. These findings validate that the ODTC algorithm not only conserves energy but also enhances task execution efficiency, thereby improving the overall energy efficiency of the system.

VI. Conclusion and Future Outlook

(I) Conclusion

With the rapid development of computer technology, the types and quantities of data have exploded. The increasing size of data centers, along with their escalating energy consumption, exacerbates environmental pollution due to increased electricity use and carbon emissions. At the same time, big data technology has found increasingly widespread practical applications in everyday life, generating significant value. However, the development and application of big data technology also pose substantial challenges to resource utilization and environmental protection.

Therefore, in the context of China's efforts to achieve carbon neutrality by 2060 and promote a "green recovery" of the world economy post-pandemic, improving the resource utilization efficiency of big data systems to reduce unnecessary energy consumption is urgent and significant. At the intersection of a new round of technological and industrial revolutions, reducing unnecessary energy use in big data technology applications could be a major transformation in global environmental protection.

In big data systems, inefficient use of resources leads to idle resources that not only fail to contribute to task processing but also consume unnecessary energy, thereby reducing the system's energy efficiency. For a task in the system, allocating more resources does not necessarily enhance the system's energy efficiency. However, improving resource utilization efficiency can enhance the system's energy efficiency. Therefore, this paper, considering the heterogeneous characteristics of nodes in big data systems, where different nodes have varying amounts of available resources and computing capabilities, and tasks have varying resource requirements, applies the ODTC (Optimal Dynamic Task Combination) algorithm to optimize resource use in big data systems, aiming to improve resource utilization efficiency during task execution. The main contributions of this paper are as follows:

1. Considering CPU, memory, disk, and network resources, we proposed energy consumption evaluation models for both compute-intensive and non-compute-intensive states in big data systems.

2. By employing matrix operations and the concept of the shortest path in a directed graph, we developed the ODTC algorithm. This algorithm finds the optimal dynamic task combination, ensuring each task in the system receives the necessary resources while minimizing energy consumption. Additionally, due to node heterogeneity in big data systems, each node has different remaining available resources. Thus, the optimal dynamic task combination also has to meet the limitations of the remaining available resources on each node to enhance resource utilization efficiency in big data systems, thereby increasing energy efficiency and reducing unnecessary energy loss.

3. We applied the ODTC algorithm's concept to the MapReduce parallel computing model within the Hadoop framework, demonstrating that the ODTC algorithm can significantly improve the energy efficiency of big data systems compared to traditional algorithms like FIFO.

(II) Future Work

In future research, we will investigate the effects of varying intervals for task selection. Different intervals will lead to variations in the number of rows and columns in the node task energy consumption matrix and the node task resource consumption matrix, as constructed in the earlier section on optimal dynamic task combinations. Different numbers of rows imply varying numbers of tasks awaiting execution, while varying numbers of columns indicate different quantities of nodes available for executing tasks. Under these conditions, the time complexity for finding the shortest path using the ODTC algorithm may increase, and the energy consumption might also rise, potentially losing its advantage over the energy consumption of the FIFO algorithm. This paper focused on enhancing resource utilization efficiency during the task allocation phase and did not analyze the optimization of the resource scheduling phase. These aspects are areas for improvement in our future work.

References

[1] J. Wang, L. Yang. "A Brief Discussion on the Impact of Big Data on Modern Society's Daily Life." Science and Technology Wind, 2019(32): 96-97.

[2] S. Gao. "Is IT Technology a Promoter or Solver of Environmental Pollution?" Guangming Daily, 2009-11-22(006).

[3] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Kemper, and T. Neumann. "Heterogeneity-Conscious Parallel Query Execution: Getting a better mileage while driving faster!." Proceedings of the Tenth International Workshop on Data Management on New Hardware. ACM, 2014.

[4] A. Roukh, L. Bellatreche, A. Boukorca, and S. Bouarar. "Eco-dmw: Eco-design methodology for data warehouses." Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP. ACM, 2015.

[5] B. Guo, J. Yu, B. Liao, D. Yang, and L. Lu. "A green framework for DBMS based on energy-aware query optimization and energy-efficient query processing." Journal of Network and Computer Applications 84 (2017): 118-130.

[6]Y. Zhou, S. Taneja, X. Qin, W. S. Ku, and J. Zhang. "EDOM: Improving energy efficiency of database operations on multicore servers." Future Generation Computer Systems (2017).

[7] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li. "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster." Information Sciences 319 (2015): 113-131.

[8] Jie Song et al. Energy efficiency optimization in big data processing platform by improving resources utilization[J]. Sustainable Computing: Informatics and Systems, 2019, 21 : 80-89.

[9] Kwon Y C,Ren K, Balazinska M.Managing Skew in Hadoop[J]. IEEE Data Eng. Bull., 2015,36(1):24-33.

[10] Ibrahim S, Jin H, Lu L1. Handling partitioning skew in mapreduce using leen[J]. Peer-to-Peer Networking and Application, 2005, 6(4): 409-424.

[11] Hsu C H, Slagter K D, Chung Y C. Locality and loading awarevirtual machine mapping techniques for optimizing communications in MapReduce applications[J]. Future Generation Computer Systems,2015, 53(1):43-54.

[12] Ibrahim S, Jin H, Lu L. Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud[C]. //IEEE Second Conference on Cloud Computing Technology and Science(CloudCom)，IEEE,2010.17-24.

[13] Chen Y, Liu Z, Wang T. Load Balancing in MapReduce Based on Data Locality[J].Algorithms and Architectures for Parallel Processing. 2014,(12):229-241.

[14] Kwon Y C, Balazinska M, Howe B. Skewtune: mitigating skew in mapreduce applications[C]//Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2012. 25-36.

[15] Xiao Z, Song W, Chen Q. Dynamic resource allocation using virtual machines for cloud computing environment[J].IEEE Trans. Parallel Distrib. Syst., 2013,24(6): 1107-1117.

[16] Chen C T,Hung L J,Hsieh S Y.Heterogeneous Job Allocation Scheduler for Hadoop MapReduce Using Dynamic Grouping Integrated Neighboring Search[J]. IEEE Transactions on Cloud Computing,2017. 14-25

[17] Caruana G,Li M,Qi M. gSched: a resource aware Hadoop scheduler for heterogeneous cloud computing environment[J]. Concurrency and Computation: Practice and Experience, 2017,29(20):e3841.

[18] L. Chen. "Research on Performance and Energy Consumption Optimization Methods for Big Data Processing Systems." [D]. Beijing Jiaotong University, 2019.

[19] J. Yu, Z. Hu, Z. Zhou, L. Yang. "Research on Computer System Energy Consumption Estimation Models." Journal of University of Electronic Science and Technology of China, 2015, 44(03): 422-427.