EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

Heterogeneous High-Performance System Algorithm Based on Computer Big Data Technology

Dongyang Pan*

School of Mathematics and Information Engineering, Xinyang Vocational and Technical College, Xinyang 464000, Henan, China

Abstract

INTRODUCTION: This paper proposes a scheduling algorithm for heterogeneous systems based on prioritization (PQDSA). This algorithm is a sort method based on a directed acyclic graph (DAG). The critical nodes in the network are grouped according to the communication and computing costs in the network. This increases the parallelism between task schedules and reduces the completion time of work sets. Then, a method of assigning multiple tasks to multiple processors using interpolation is proposed. The PQDSA method can effectively reduce the time of scheduling multiple tasks and improve the scheduling effect. PQDSA is compared with EDL-θ and EDF scheduling methods. The results show that this method has better scheduling efficiency.

Keywords: Priority queue division, Heterogeneous system, High-performance computing, Time of completion, Dispatching Efficiency

Received on 24 August 2023, accepted on 06 October 2023, published on 18 October 2023

doi: 10.4108/eetsis.3789

*Corresponding author. Email: pandongyang@xyvtc.edu.cn

1. Introduction

Heterogeneous computing is a kind of computing mode composed of arithmetic units with different kinds of instructions and structures. Commonly used computing units include CPU, GPU, DSP, ASIC, FPGA, etc. [1]. Its advantage is that in addition to using the traditional CPU calculation, it can accelerate the computing processing speed through other computing units so that the computing device has a more efficient processing capacity. Heterogeneous computing has been increasingly applied in cluster systems such as supercomputers and cloud servers. Graphical Processing units (GPUs) are increasingly used in compute-intensive software fields because of their high computing speed, fast storage bandwidth, and parallelization. Whether on a PC or a supercomputer, GPUs play a dominant role. Among the 500 supercomputers in the world, including the SGI AltixUV and CrayXK6, all use graphics processors on a large scale. GPU parallel computing is moving into the mainstream. In recent years, with the development of CPU and GPU technologies, many high-performance computers have been used to build heterogeneous computing platforms [1].

CPU and GPU hardware characteristics are different. The CPU uses a chaotic mode of operation, which uses the prefetch and Cache hierarchy to reduce the memory access delay. It has a larger cache capacity and more data and logical computing units, which makes the CPU ideal for performing complex and data-dependent computing tasks such as distributed computing, data compression, artificial intelligence, and physical simulation. The GPU uses a continuous running method. It operates in many units. It relies on massive multithreading to hide its memory access latency, while GPU performance depends on the application [1]. Even the same program can impact the performance of the GPU. The main uses of graphics processors are image analysis, data processing, video processing, etc. These differences lead to significant differences in CPU and GPU execution performance. Some applications can run fast on the CPU (GPU) but slow on the GPU (CPU) [2]. It is imperative to effectively utilize the advantages of CPU and GPU in heterogeneous systems, improve the operating efficiency of hardware,

and achieve optimal resource allocation. Therefore, a dynamic task scheduling algorithm is adopted to reasonably allocate computing tasks to corresponding hardware computing units in a particular proportion to achieve the above purpose [3].

The academic community divides workload balancing in computer systems into two types: one is static, and the other is dynamic. Static scheduling refers to setting the assignment rate of work according to the expected work cycle before the work starts. The theoretical calculation of the prediction time is realized by calculating the execution performance of each processor, compiling time parameters and offline learning time. Although this method does not need synchronization between multiple tasks and the communication cost is low, its application scope is narrow, and there are problems of unequal load distribution. Qilin method is a classical static task scheduling method. It applies the running and data transfer speeds tested in the learning process and constructs the evaluation model between CPU and GPU. A hybrid static scheduling algorithm is presented in the literature [4]. An optimization method based on a genetic algorithm is proposed. Firstly, it uses a list heuristic scheduling algorithm, including subsequent nodes, to obtain a scheduling result close to the best. Then the genetic algorithm is used to optimize the scheduling results in the first stage. Communication and calculation methods are proposed in the literature [5]. The algorithm realizes the real-time statistics of system running time and data sending time by offline method. The predicted operation and delivery accuracy depend primarily on the actual operation. Due to different operation requirements and different hardware conditions, the required operation time will be different. This method has some limitations in practice.

Dynamic scheduling is an algorithm that determines the load allocation ratio based on resource sharing between GPU and CPU. Although dynamic scheduling costs are higher than static scheduling, its time estimation is more accurate. Literature [6] proposes a dynamic scheduling algorithm based on universal memory address space access (AHS). The system's load characteristics and the CPU and GPU computing speed are analysed in real-time without offline data processing. Literature [7] proposes a dynamic adaptive scheduling algorithm (DSS). The work blocks assigned to the processor during the initial work process are relatively large. As the number of tasks remaining decreases, so does the number of tasks for the CPU and GPU. Literature [8] describes a method of job theft. Because the GPU cannot actively communicate with the CPU, the GPU cannot make load requests to the work pool. In literature [9], when there is no dedicated route, the communication is controlled by a dynamic program combined with GPU and CPU, but this method will reduce the operation efficiency. This thesis aims to improve the parallelism of multi-core systems and improve the efficiency of multi-core systems. A priority queue partitioning scheduling algorithm (PQDSA) is designed for parallel work with multiple entries and multiple exits. First, the number of priority queues is determined according to the number of input nodes, and then it is sorted and scheduled. This article can simplify a complex and diverse set of tasks.

2. Task scheduling model

In a heterogeneous system, assigning a limited number of tasks to different processing units to complete the operation is complex. Usually, such problems are described by DAG diagrams. A task can be expressed as a node. Nodes are the minor units in the scheduling process [10]. When a task is to be completed, the host finds the appropriate processor according to a particular search strategy and schedules it. It distributes the work to the appropriate processor to achieve a predetermined optimal solution. Figure 1 shows a task assignment model that assigns five tasks to two processors.

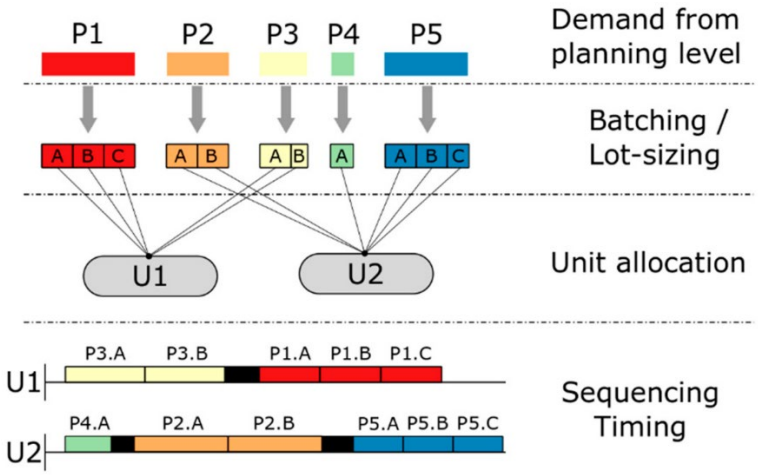

Figure 1. A scheduling model for five jobs on two processors

2.1 Description of the application model

Scheduling of multiple tasks is a significant problem in heterogeneous computing environment. Such $F=\{P,H,D,V\}$ class of tasks with a restrictive relationship can be described by a DAG graph. In this model, $P=\{p_i \mid (i=1,2,\cdots,n)\}$ represents a set of multiple task nodes. n is the number of knots for the whole job. $H=\{h_{ij} \mid h_{ij}$ stands for the edge$\}$ between node p_i and node p_j. There is a priority constraint relationship between node p_i and node p_j, and p_j is the direct successor of p_i. $V=\{v_{ij} \mid v_{ij}$ stands for the cost of communication between p_i and E. p_i is the direct leader of p_j. v_{ij} stands for additional communication load when two work nodes are running on different processors. $D=\{d_{ij} \mid$ represents the computation cost of node p_i on processing $q_j\}$. Here q_j refers to the j handler in a heterogeneous system. P is a set of handlers. $P=\{q_1,q_2,\cdots,q_m\},m$ represents the total number of processors. When one or more entry and exit nodes exist, a virtual node with only entry and exit nodes can be constructed [11]. This virtual node requires neither extra communication cost nor extra operation cost. When scheduling, we need to receive all the information from each leading node, and then we can enter the ready state and schedule. The DAG task diagram is shown in Figure 2. The value at the oriented edge represents the dependence between the two nodes and is called the communication cost.

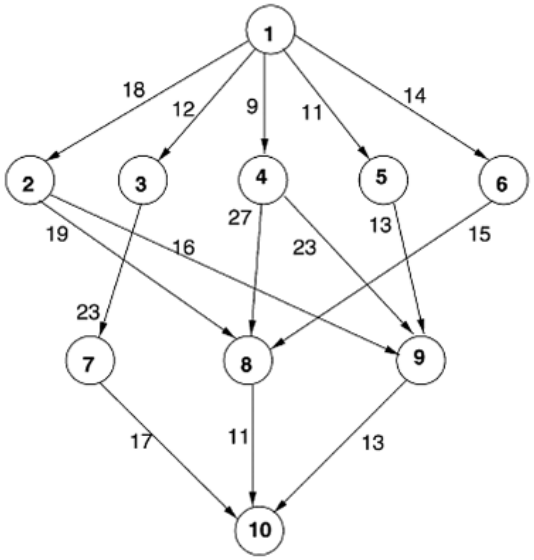

Figure 2. DAG application with ten work nodes

Definition 1: Start and end times. If $T1(p_i,q_j)$ and $T2(p_i,q_j)$ represent the start and end times of work p_i on a processor q_j, then the mathematical relationship between the two problems can be expressed by the formula (1) :

$$T2(p_i,q_j)=T1(p_i,q_j)+v_{ij} \tag{1}$$

Definition 2: Precursor and successor nodes. $pred(p_i)$ represents the precursor node of work p_i. If task p_i has no prefix node, then the job is the input node. $succ(p_i)$ represents the next work node performed under item p_i. If there is no subsequent node, then this node is the output node. The average operational cost of working node p_i is:

$$\overline{D_i}=\sum_{j=1}^{n}\frac{d_{ij}}{n} \tag{2}$$

d_{ij} represents the amount of computation of task p_i in processor q_i. n represents the number of processors. Among them, the communication cost in the network, the computing cost of the network itself on the processor and the resource allocation of the front end of the network are three important factors that affect the network performance.

Definition 3: Task Level $TV(p_i)$ value. Each node is recursively calculated, and the T-level value $TV(p_i)$ of each node is obtained. Its formula is:

$$TV(p_i)=\max_{p_j\in pred(p_i)}\{TV(p_j)+v_{ij}\}+\overline{D_i} \tag{3}$$

The calculation formula for input node $TV(p_i)$ is expressed as formula (4):

$$TV(p_{entry})=\overline{D_{entry}} \tag{4}$$

Definition 4: Plan the length. P_{exit} represents the output node. $T2(P_{exit},q_j)$ represents the completion time of the entire schedule [12]. If there are multiple output nodes, virtual nodes with zero computation cost and zero communication cost are added to the output node so that the output node becomes the final unique output node. The scheduling length is:

$$makespan=\max\{T2(P_{exit},q_j)\} \tag{5}$$

The ultimate goal of the sorting problem is to distribute the work of each node in the network reasonably according to a particular order to achieve the shortest sorting time.

Definition 5: Scheduling effectiveness [13]. The scheduling efficiency of the algorithm refers to the ratio of task scheduling acceleration to the total number of

processors. Its value varies between 0 and 100%. This method not only considers the execution speed of the DAG algorithm, but also considers the number of processing units. Its formula is:

$$Efficiency = \frac{Speedup}{n} \quad (6)$$

n represents the number of processors. $Speedup$ is the acceleration ratio. Acceleration ratio refers to the ratio of the sequential computation time of the task set to the scheduling length. The notations and parameters that can be used in this article are summarized (Table 1).

Table 1. Symbol definitions

Argument	Definition
P	Working set
p_i	Task i
P_j	Process j
d_{ij}	The computational cost of task node p_i on processor q_j
v_{ij}	Communication overhead between task nodes p_i and p_j
$pred(p_i)$	The pilot Node set of Task Node p_i
$succ(p_i)$	A group of subsequent nodes of Task Node p_i
$\overline{D_i}$	The cost of averaging operations
P_{entry}	Inlet node
P_{exit}	Exit node
S_i	Priority queue the blank queue required for sorting
$AVGT(p_i)$	Average completion time of task node p_i
$Mpred(p_i)$	Sum of direct leader nodes of task node p_i
$Mexit(S_i)$	Sum of Output nodes of queue S_i
$TV(p_i)$	t level of Task Node p_i

2.2 Preference Quality Distribution System (PQDSA) Algorithm

Priority queue division Phases

When the task is scheduled, the number of entering nodes determines the priority. More than one direct leader node is called a critical node. The main idea of task partitioning is to divide key nodes into suitable queues to generate the best task-scheduling queue. This calculates the average AVGT completion time for each node [14]. The node is divided into several small blocks based on the value of AVGT. And assign it to the appropriate processor. A blank queue $S_i(i = 1,2,3\mathrm{L})$ must be used when prioritizing. The size of i is determined by the number of input nodes. The parameter $Mexit(S_i)$ represents the number of nodes expelled from queue S_i. $Mpred(p_i)$ represents the total number of direct leading nodes in node p_i. For complex task applications with multiple input and output nodes, nodes with no operational cost and no transmission delay can act as pseudo-input and pseudo-output nodes. Virtual nodes do not affect the overall task assignment. AVGT is obtained from the input node by recursion. Its expression is expressed in formula (7):

$$AVGT(p_i) = \max_{p_j \in pred(p_i)} \{AVGT(p_j) + v_{ij} + \overline{D_i}\} \quad (7)$$

v_{ij} represents the communication cost required to transmit data from task node p_i to task node p_j.

When nodes p_i and p_j are assigned to the same processor, the value of v_{ij} is 0. $\overline{D_i}$ represents the computation of node p_i per processor. The average AVGT of input node P_{entry} can be expressed as:

$$AVGT(P_{entry}) = \overline{D_{entry}} \quad (8)$$

$\overline{D_{entry}}$ is the average value of the input node p_{entry}. The sorting order is determined according to the number of input nodes in the DAG task graph. With the help of blank queue $\overline{D_{entry}}$, select the item node and place it in the blank queue. The AVGT of each node is recursively calculated from the input node. When a node is selected in the queue, it is processed as follows:

If node p_i has only one direct precursor node, then this node is placed directly in the queue of its predecessor nodes. 2) If node p_i has multiple direct leader nodes, that is, node $Mpred(p_i) > 1$, the node is a key node. AVGT was compared with their leading nodes. Place a node in the queue of nodes with the largest AVGT value. Figure 3 shows the result of the task splitting queue (image cited in Wireless Communications and Mobile Computing, 2018, 2018.).

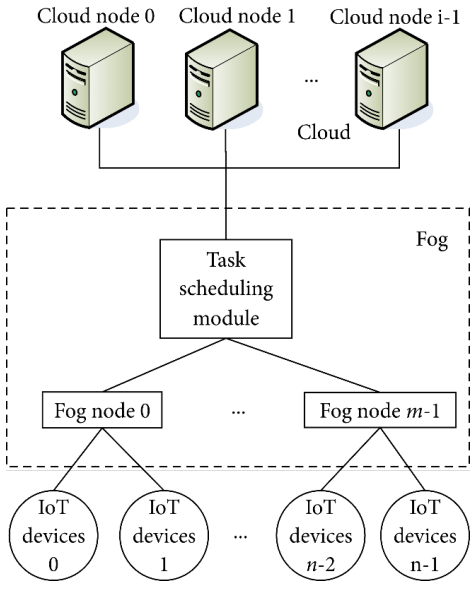

Figure 3. Results of task priority queue partitioning

Stages of prioritization

The number of input nodes determines the queue order. Adding nodes in the queuing process increases the communication cost and parallelism between nodes in the network. Because of the presence of essential nodes, the data correlation between queues will not be eliminated. When deciding the order of A group of multiple tasks, we must first estimate $TV(p_i)$ of each work node and then sort each working node. In the process of executing tasks, a significant problem is how to determine the location of essential node tasks [15]. If the previous node of a critical node has not been arranged, the previous node must be arranged first before the next node can be arranged. If this node is crucial, then the number of direct leading nodes $Mpred(p_i)$ of this node is more than one. If all direct leaders of a node come out of the same queue, then the key node is assigned directly. If one of its immediate front ends has other queued nodes, consider whether this node has been scheduled and whether the data has arrived. When it is judged that the leading node has been arranged and the data has arrived, the critical node can be arranged. Nodes of $TV(p_i)$ with the same value can be assigned arbitrarily. In the priority selection phase of the task, the $TV(p_i)$ values of the nodes in priority queue S_1 and S_2 are first calculated. Sort by its size in units of $TV(p_i)$. You get an example to represent priorities S_1 and S_2. The corresponding set of nodes is $\{P_1, P_3, P_6, P_8, P_9\}$ and $\{P_2, P_4, P_5, P_7, P_{10}\}$.

Processor selection phase

The algorithm sorts multiple tasks in order, dramatically reduces the correlation between tasks and increases the parallelism, thus shortening the average execution time of multiple nodes. Then, according to the tasks in the queuing process, determine the order in which they are produced. Determine that the critical node meets the priority constraint condition [16]. This section focuses on scheduling tasks on the appropriate processor to execute to make the node progress faster. In applying the DAG work chart, work is assigned to specific processing units to be completed. When the node is finished, it needs to wait until all of its work is completed, send the corresponding data to its processor, and start its work. There is no communication overhead when two work nodes run simultaneously on the same processor. This critical node has the highest task priority in the same hierarchy. The formula for calculating the real start time (ST) and the actual end time (FT) of work is as follows:

$$ST(p_{entry}) = 0 \quad (9)$$

$$ST(p_i) = FT(p_i) - d_{ij} \quad (10)$$

$$FT(p_i) = \max_{p_j \in pred(p_i)} \{ST(p_i) + d_{ij}\} + k \square v_{ij} \quad (11)$$

$k = 0$ when p_i and its start node p_j are running on the same processor. $k = 1$ when p_i and its leader p_j are running on different processors.

If each job in the queue has only one direct leading node, then all the jobs are assigned to the same processor.

There is no additional communication burden in this case, and no need to wait for transmission from other nodes. When the task has more than one direct leader node, this node is the critical node. The last precursor node where data arrives is the crucial parent node.

3. Experiment and analysis

3.1 Experimental parameter setting and deployment

The hardware requirements for this test are shown in Table 2. Using Rodinia standard system, 11 open CL systems are selected as research objects [17]. They are backprop, bfs, gaussian, Hotspot, hotspot3D, lud, nn, nw, pathfinder, srad and streamcluster. Because the voltage/frequency range that each CPU/GPU can adjust is different, the voltage/frequency setting is standardized first. The minimum CPU frequency is 1200 MHZ. Its maximum operating frequency is 3300 MHZ. These frequencies are divided into ten frequency levels at a step size of 233 MHZ. $f_C \in$ [0.59, 1.64] can be obtained by standardizing it with the default CPU frequency of 2100 MHz. A GPU core with a minimum frequency of 567 MHz and a maximum frequency of 1595MHz was used. The article divides it into ten levels. The default frequency of the GPU core is 1357 MHz, which is standardized to obtain $f_{Gc} \in [0.44,1.22]$.

Table 2. Experimental environment

Operating system	**Ubuntu 16.04**
CPU	Intel Xeon E5-2695∗2
GPU	Nvidia Tesla P40∗4
Internal memory	32GB DDR 4∗8
Hard disk	480GB SSD 4T BHDD

The precise figures of DVFS power consumption parameters P_{C0} and λ of CPU are given. Perform 11 tasks on the CPU with ten different voltage and frequency Settings. The average result is obtained by fitting the obtained results. The result is P_{C0} =57.57W and λ =31.75. Use ten different voltages/frequencies on the GPU to complete 11 tasks. The result of fitting was P_{C0} =65.11W, σ =66.44.

Eleven tasks are completed at ten different frequencies on the CPU. The measured execution time and energy consumption value establish the system performance model. The state of operation of a particular type of task at a particular type of voltage/frequency on the processor can be derived from the system performance model.

When the task of 11 test sets is executed on the GPU when the number of cores on the CPU to assist the GPU calculation is less than 3, the execution efficiency of the GPU will be significantly affected. Therefore, in this experiment, three cores from the 18 cores of the XeonE5-2695 CPU were selected as auxiliary cores to run at the highest frequency of 3300MHz. The remaining 15 cores are used as computing cores to maximize the utilization of CPU computing resources.

This paper adopts the CPU Freq tool under Linux to realize the dynamic adjustment of CPU voltage and frequency level. It can view and individually adjust the frequency of each core in the CPU. Use the Nvidia-semi software on the GPU to adjust the voltage/frequency. This tool, developed by Nvidia Inc., monitors GPU usage and changes its state. Use Intel's Power Governor tool to measure the power consumption of the CPU. Use Nvidia-smi to measure the power consumption of your GPU. The energy consumption of the whole server and the energy consumption of the switch is measured by the energy meter.

The energy consumption $E_{turnon/off}$ of the server switch is 262.50MJ. When the CPU and GPU are idle, power consumption is set to the lowest voltage/frequency setting. The CPU power consumption at 1200 MHZ is 28.05W, the GPU power consumption is 9.49W, and the GPU power consumption is 9.89W at 544MH core frequency. The measured other power consumption P0 value is 44.46 W. It is also assumed that each server can be equipped with 1/2/4/8/16/32 CPU/GPU pairs. It is expressed as P1/P2/P4/P8/P16/P32.

This paper models the case of sending a task to a task cluster in one hour. In the base unit of 1 second, Time $\in$ [1,3600]. The amount of work completed in each period corresponds to the normal distribution. Each completed assignment was randomly selected from 11 assignments.

3.2 Experimental comparison algorithm

This paper will give a comprehensive evaluation of these new scheduling methods. PQDSA, EDL-θ scheduling algorithm and EDF algorithm were configured on different server CPU/GPU pairs. The experimental results are given under different running times and load conditions [18]. The three methods' energy consumption and running speed differences were compared. EDL-θ algorithm is a new method combining DVFS technology with DRS technology.

3.3 Algorithm comparison and result analysis

Algorithm comparison results

PQDSA method is a priority queuing method based on job category. The benefits of this approach are shown in Figure 4. Compared with the standard EDF method, the PQDSA method can save 20.52%,26.15%,30.42%,31.56%,36.25%, and 39.17% energy consumption at P1~P32. Compared with the EDL

- theta method can save 7.08%, 5.1%, 5.73%, 6.67%, 10.73%, and 2.92% of the energy consumption. The server's idle energy consumption and switching energy consumption can be effectively reduced by using the heterogeneous algorithm of the platform. The PQDSA method is first to determine the optimal execution times of each job and then schedule it. And default tasks on that processor at an optimal frequency. In addition, the PQDSA algorithm also uses a way to group work. The characteristic of the algorithm is that there is low energy consumption and a short execution time between the tasks in the execution process. Ensure that most of the work is done at the optimal frequency on the preferred processor. Most work is done at the lowest theoretical state [19]. It also prevents DVFS tasks from executing at high frequencies on undesirable processors. Before the DVFS alternative method is applied, the alternative method that does not increase the system's energy consumption should be adopted as far as possible. The PQDSA algorithm generates a smaller energy consumption in the system, significantly reducing idle and server switching consumption. It has the best energy consumption efficiency compared with the other two methods.

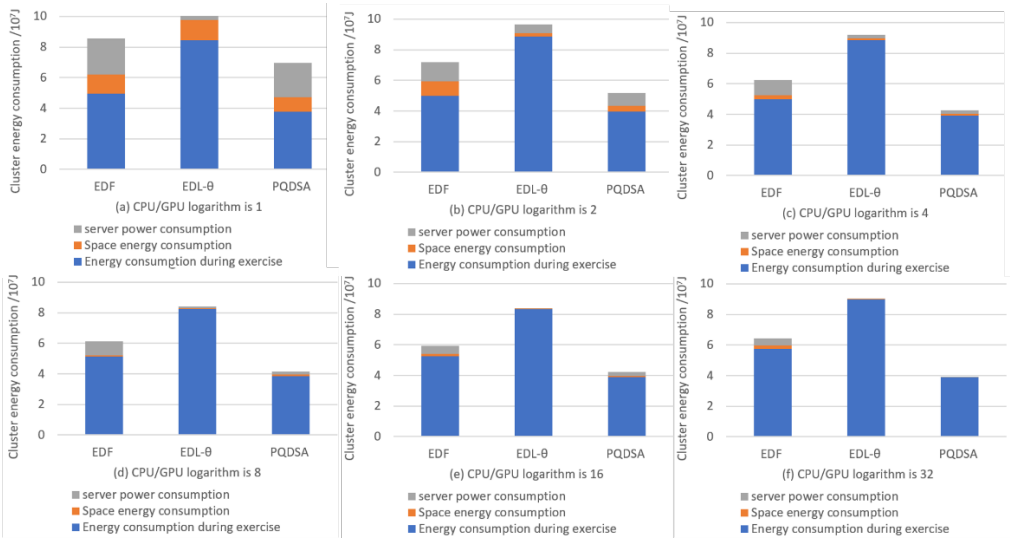

Figure 4. Cluster energy consumption for different CPUs/GPUs

EDL-θ algorithm is based on DVFS technology. The performance of this method is not ideal under the experimental environment in this paper. In P1-P32 cases, the performance is worse than the EDF algorithm. Total energy consumption is 148.33%, 155.31%, 161.35%, 158.54%, 152.50%, and 152.40% of EDF, respectively. Although DVFS technology can significantly reduce the number of open servers and the system's idle energy consumption and switching energy consumption, the current DVFS technology does not divide the system into multiple types, resulting in solid randomness in the system's work. In addition, because DVFS technology is used, tasks are executed at a higher frequency when not preferentially selected [20]. This will cause the energy consumption in the execution process to increase sharply and affect the execution efficiency of the algorithm. The PQDSA method has the best energy consumption efficiency in the case of high load. Compared with EDF and EDL-theta algorithms, the PQDSA algorithm can achieve 31.56% and 56.46% energy-saving effects when 8 CPU/GPU pairs are loaded on each processor.

Processor The effect of CPU/GPU logarithm on algorithm performance

When the number of CPU/GPU pairs in the server is small, the running power consumption accounts for most of the total power consumption. At the same time, in the operation of the system, the running consumption of the server side and the server switch consumption also occupy a large proportion. The reason is that the number of CPU/GPU dual cores is too small, making the computing power of a single server too low. More servers must be opened to process the submitted tasks, resulting in a significant server switch energy drain (Figure 5).

The total energy consumption of the three methods decreases as the logarithm of CPU/GPU in each server increases. The overall energy consumption of 32 CPUs/GPUs on EDF was reduced by 14.48% compared to 1 CPU/GPU pair. Compared with the traditional PQDSA method, the energy consumption of the whole system is reduced by 36.68%. The overall energy consumption of the EDL-θ algorithm is reduced by 11.00%. When the number of CPUs and GPUs in a single server increase, the computing capacity of a single server increases, and the number of servers that need to be opened significantly

decreases. This dramatically reduces the energy consumption of the server switch, which in turn reduces the overall energy consumption. Under the proposed test conditions, when the number of CPU/GPU pairs is more, the total energy consumption of the cluster to complete the task is less. The energy consumption ratio during working hours to total energy consumption is taken as the effective energy consumption ratio, as shown in Table 3. The PQDSA method has good performance. Especially when the logarithmic ratio of CPU and GPU is large, the effective energy utilization of the system reaches nearly 100%. 10[7]

Table 3. Effective energy consumption ratio

Indicates the number of CPU/GPU	Effective energy consumption ratio (%)		
	EDF algorithm	EDL-θ algorithm	PQDSA algorithm
1	71.30%	66.74%	83.98%
2	81.24%	80.06%	91.99%
4	86.70%	90.05%	96.65%
8	91.42%	94.82%	98.58%
16	91.49%	97.37%	99.80%
32	93.24%	99.16%	99.98%

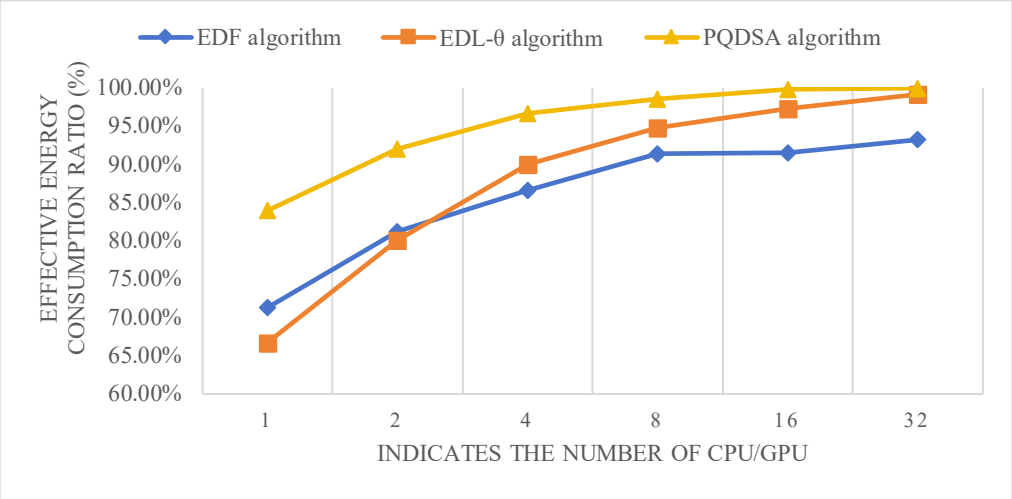

Figure 5. Clustering energy consumption under three different time Settings

4. Conclusion

This paper presents a scheduling algorithm based on priority queue partitioning (PQDSA). The PQDSA algorithm sorts the nodes and divides the node tasks with high complexity and low parallelism into several sequences. There is a dependency between the key node and the leader node. In the grouping process, the critical nodes are placed in the corresponding team according to the average arrival time of nodes. This can reduce the task time consumption. Compared with the traditional EDF and EDL-θ methods, it is found that the PQDSA method has more significant advantages in scheduling length and scheduling efficiency.

Acknowledgments.

This work was supported by Xinyang Vocational and Technical College.

References

[1] Saritha, S., Mamatha, E., Reddy, C. S., & Rajadurai, P. A model for overflow queuing network with two-station heterogeneous system. International Journal of Process Management and Benchmarking, 2022; 12(2):147-158.

[2] Tian, S., Ren, W., Deng, Q., Zou, S., & Li, Y. A predictive energy consumption scheduling algorithm for multiprocessor heterogeneous system. IEEE Transactions on Green Communications and Networking, 2021;6(2):979-991.

[3] Moori, A., Barekatain, B., & Akbari, M. LATOC: an enhanced load balancing algorithm based on hybrid

AHP-TOPSIS and OPSO algorithms in cloud computing. The Journal of Supercomputing, 2022;78(4):4882-4910.
[4] Ben Alla, H., Ben Alla, S., Ezzati, A., & Touhafi, A. A novel multiclass priority algorithm for task scheduling in cloud computing. The Journal of Supercomputing,2021; 77(10):11514-11555.
[5] Wagner, C., Dhanaraj, N., Rizzo, T., Contreras, J., Liang, H., Lewin, G., & Pinciroli, C. SMAC: Symbiotic multi-agent construction. IEEE Robotics and Automation Letters, 2021;6(2):3200-3207.
[6] Liu, L., Tang, J., Liu, S., Yu, B., Xie, Y., & Gaudiot, J. L. π-rt: A runtime framework to enable energy-efficient real-time robotic vision applications on heterogeneous architectures. Computer, 2021;54(4):14-25.
[7] Hu, B., Cao, Z., & Zhou, M. Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems. IEEE Transactions on Services Computing, 2021;15(5):2766-2779.
[8] Mack, J., Arda, S. E., Ogras, U. Y., & Akoglu, A. Performant, multi-objective scheduling of highly interleaved task graphs on heterogeneous system on chip devices. IEEE Transactions on Parallel and Distributed Systems, 2021;33(9):2148-2162.
[9] Ulugbek, A., & Azamat, Q. Model of optimal distribution of network resources with constraints on quality of service indicators. Bulletin of Electrical Engineering and Informatics, 2023;12(2):835-842.
[10] Jenila, L., & Canessane, R. A. Cross Layer Based Dynamic Traffic Scheduling Algorithm for Wireless Multimedia Sensor Network. IJEER, 2022;10(2):399-404.
[11] Khenwar, M., Sisodia, A., Vishnoi, S., & Kumar, R. Exploration: Cloud Computing Scheduling Techniques. Scandinavian Journal of Information Systems, 2023;35(1):673-679.
[12] Madhura, R., Uthariaraj, V. R., & Elizabeth, B. L. An efficient list - based task scheduling algorithm for heterogeneous distributed computing environment. Software: Practice and Experience, 2023;53(2):390-412.
[13] Huang, J., Li, R., An, J., Zeng, H., & Chang, W. A DVFS-weakly dependent energy-efficient scheduling approach for deadline-constrained parallel applications on heterogeneous systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021;40(12):2481-2494.
[14] Kaur, N., Kumar, A., & Kumar, R. 2022; TRAP: task-resource adaptive pairing for efficient scheduling in fog computing. Cluster Computing, 25(6):4257-4273.
[15] Hamid, L., Jadoon, A., & Asghar, H. Comparative analysis of task level heuristic scheduling algorithms in cloud computing. The Journal of Supercomputing, 2022;78(11):12931-12949.
[16] Tran-Dang, H., & Kim, D. S. FRATO: Fog resource based adaptive task offloading for delay-minimizing IoT service provisioning. IEEE Transactions on Parallel and Distributed Systems, 2021; 32(10):2491-2508.
[17] Azizi, S., Othman, M., & Khamfroush, H. DECO: A Deadline-Aware and Energy-Efficient Algorithm for Task Offloading in Mobile Edge Computing. IEEE Systems Journal, 2022;17(1):952-963.
[18] Yesil, S., & Ozturk, O. Scheduling for heterogeneous systems in accelerator-rich environments. The Journal of Supercomputing, 2022;78(1):200-221.
[19] Serdaroglu, K. C., & Baydere, S. An efficient multipriority data packet traffic scheduling approach for fog of things. IEEE Internet of Things Journal, 2021;9(1):525-534.
[20] Liu, J., Huang, J., Li, Z., Li, Y., Wang, J., & He, T. Achieving per-flow fairness and high utilization with limited priority queues in data center. IEEE/ACM Transactions on Networking, 2022;30(5):2374-2387.