Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 3890255, 11 pages
https://doi.org/10.1155/2022/3890255

Research Article

Construction of Artistic Design Patterns Based on Improved Distributed Data Parallel Computing of Heterogeneous Tasks

Yao Sun

Yantai Vocational College, Yantai 264670, Shandong, China

Correspondence should be addressed to Yao Sun; 20171006@ytvc.edu.cn

Received 17 January 2022; Revised 11 February 2022; Accepted 19 February 2022; Published 31 March 2022

Academic Editor: Gengxin Sun

With the continuous upgrading of hardware in the terminal equipment, how to provide high-performance computing for low-tech threshold users has become a current research hotspot. In the era of green high-performance computing, the heterogeneous computing system can provide good versatility, performance, and efficiency and has broad development prospects. This article provides an in-depth analysis and research on the construction and application of improved models using the artistic design pattern of heterogeneous tasks and parallel computing. Based on the hardware resources in the existing desktop system, this article optimizes the original heterogeneous parallel technology from the aspects of task division and data transmission to reduce the complexity of data allocation and processing for users. Based on the analysis and study of the multicore CPU and GPU architectures in the desktop system, as well as the original CPU-GPU heterogeneous parallel technology, this article optimizes the solution of heterogeneous parallel computing, designs a heterogeneous parallel computing architecture, and deploys a heterogeneous parallel computing architecture. The nodes of the desktop system constitute the parallel computing system. In terms of task allocation, the computing system divides tasks according to the parallelism of tasks. According to the computing resources and bandwidth conditions of each heterogeneous node, starting from the parallel execution time, the task scheduling algorithm is optimized, and the load balancing scheduling scheme is designed to achieve the optimal allocation of resources. In terms of storage resources, the computing system adopts distributed storage as a whole. The CPU-GPU heterogeneous parallel in the desktop system adopts virtual unified storage. Global distributed storage and local shared storage are used to balance overall performance and programming complexity. This article introduces the design and implementation of JTangSync, a distributed heterogeneous data synchronization system. The system adopts a distributed architecture, and each node is organized by a data source module, a data transmission module, a processor module, etc. The data source module is responsible for extracting data, the data transmission module is mainly responsible for efficient data transmission, and the processor module is responsible for data processing. More importantly, each module is designed as a replaceable plug-in, which is convenient for secondary expansion. Each node relies on ZooKeeper to form a cluster, which realizes distributed functions such as centralized management of distributed resources, failover, and resumed transmission. Compared with the mainstream scheduling algorithms HEFT, CPOP, PEFT, and HSIP on heterogeneous systems participating in the experimental evaluation, the scheduling length ratio of DONF series algorithms is reduced by 36.3%–67.5% and the parallelism is increased by 17%–125% in terms of efficiency. Compared with the existing database synchronization system, the JTangSync system has built-in multiple heterogeneous database data sources and supports the synchronization of complex heterogeneous databases. The system supports users to develop and customize their own data sources and data processing programs, to promote secondary development. By adopting the custom compressed data exchange format and network optimization methods such as packet merging, caching, and adaptive compression algorithm, the system has high performance.

1. Introduction

With the rapid development of technologies such as the Internet of Things, mobile Internet, and artificial intelligence, human society has entered the era of big data, and data generation is growing exponentially [1]. A 2017 joint study by Seagate Technology and International Data Corporation (DC) indicates that by 2025, the total amount of global data is expected to reach 163 ZB, 10 times the total amount of global data in 2016. Big data here is large in scale volume and data dimensionality, and its analytical processing is often time-sensitive and accuracy-demanding [2]. The advent of Big Data has also changed the way scientific research is conducted, over calculations that yield unknown and plausible theories. Big data hides great value and has been considered a strategic resource by many countries [3]. The mining of big data value has placed higher demands on computing. However, the gap between available computing power and computing demand is instead widening due to the failure of Moore's law and the limitation of power consumption wall, so there is an urgent need to explore new ways of data-intensive computing. Supercomputers based on heterogeneous multicore architectures are considered to be the "killer CfD" weapon for solving big data problems, but exploiting their full potential faces challenges in-memory access, thread organization, data sharing, programming models, etc.

The rapid growth of Internet users and the increasing research and applications in areas such as big data and artificial intelligence have led to a dramatic expansion in the volume of data requiring computation and a rapid increase in the demand for computational power for related applications [4]. Various computationally intensive applications such as databases, intelligent algorithms, deep learning, online prediction, and unmanned vehicles are demanding computation far beyond the capabilities of traditional CPU processors. The problem of parallelizable computing was initially explored with parallel computing to improve computational performance. Parallel computing is the process of using multiple computing resources simultaneously to solve a computational problem. It decomposes the problem being solved into several parts, each of which is computed in parallel by a separate processor, to achieve the goal that the time taken to solve the problem with multiple computing resources is less than the time taken with a single computing resource [5]. Parallel computing can be divided into temporal parallelism and spatial parallelism. Temporal parallelism refers to pipelining techniques, and spatial parallelism refers to the concurrent execution of computations by multiple processors, that is, connecting more than two processors through a network to achieve simultaneous computation of different parts of the same task or large problems that cannot be solved by a single processor. A parallel computing processor system can be either a specially designed supercomputer containing multiple processors, or a cluster of several independent computers interconnected in some way [6].

Heterogeneous computation refers to how computing is performed by a system composed of computing units using different types of instruction sets and architectures. Heterogeneous computing is a special way of performing the computation in a hybrid system composed of various computing units such as CPUs, digital signal processors (DSPs), graphics processors (GPUs), application-specific integrated circuits (ASICs), and field-programmable logic gate arrays (FPGAs), using different types of instruction sets and computing units of different architectures [7]. Some of the more popular heterogeneous computing platforms include CPU + GPU, CPU + ASIC, and CPU + FPGA. GPUs and FPGAs as co-processors have higher performance than CPUs in terms of parallelism, bandwidth, and latency. GPUs are general-purpose processors like CPUs, but have a different architecture; the hardware resources in the GPU are heavily used as logical algorithm units (ALUs) and a small portion as control circuits. This provides the basis for large-scale parallel processing of data. There is a large amount of open-source software and application software in the GPU space, with abundant and easily accessible resources for programming, low development costs, and short development cycles. The Nvidia vendors provide a rich architecture, and some deep learning tools support GPUs, which have been widely used in areas such as deep learning training [8].

2. Related Works

In the current market situation, the dominant heterogeneous computing systems are mainly CPU + GPU and CPU + FPGA, which have their advantages. The speed of GPU development in recent years has left Moore's law, which has lasted for more than 50 years. The widespread popularity of GPU in various fields has greatly improved the speed of computer graphics processing, and the quality of graphics processing has also made a qualitative leap. And one of the most surprising things is that the GPU, which came about because of graphics processing, is now achieving refreshing success in general-purpose computing. Since 2001, when NVIDIA introduced the GeForce3 series of GPU graphics cards, GPUs have been used for computation beyond graphics rendering, and in 2007, NVIDA's development of CUDA made it possible for developers to use the C language for general-purpose GPU programming, unlocking the computational potential of GPUs [9]. The increase in computing performance brought about by CUDA led to the expansion of GPUs into the fields of medicine, "big data," aerospace, and artificial intelligence. Subsequently, Apple collaborated with several well-known companies in the industry to introduce OpenCL, a more versatile heterogeneous parallel programming language, further enhancing the collaboration between hardware in heterogeneous systems, and further increasing the efficiency of heterogeneous computing.

Aldo Camargo et al. used CPU-GPU heterogeneous computing for surveillance video stitching with unmanned aerial systems (UAS), where the GPU is responsible for completing LR (resolution) mosaic construction and the CPU is responsible for error stitching, and this technique significantly speeds up video stitching while also improving the resolution of the video. Julien Bert et al. and Takashi

Machida et al. proposed a CPU-GPU-based framework for pedestrian and vehicle detection, which enables low-cost real-time image processing and improves the speed of pedestrian and vehicle detection by more than 30 times compared to traditional CPU implementations based on CPU-GPU heterogeneous computing. Julien Bert et al. use CPU-GPU heterogeneous computing to accelerate Monte Carlo simulation (MCS), which enables the application of MCS in medical imaging and particle therapy through the acceleration effect of GPU on computation. Siddharth Bhatia et al. used CPU-GPU heterogeneous computing for error-weighted parallel analysis of K-critical search to enhance analysis efficiency through GPU-accelerated heterogeneous platforms [10]. Han Bo used CPU-GPU collaborative computing for earthquake hazard prediction and proposed a seismic simulation method based on urban buildings, which is more than 30 times more efficient than traditional CPU computing. Li Peng long et al. came up with a CPU-GPU collaborative fast orthorectification method, which is more than 50 times faster than the traditional CPU-based approach after optimizing GPU configuration and storage. Peng Jingwei et al. used GPU heterogeneous parallel platform for a multiscale collaborative head detection system; using GPU to accelerate HOG feature extraction and CPU to control the normal operation of each module, the detection efficiency of this heterogeneous system is improved nearly 10 times compared with the traditional CPU detection. Munshi and Mohamed proposed a hybrid domain full-waveform inversion technique based on CPU-GPU-accelerated platform—the use of heterogeneous platform greatly improved the computational efficiency of the system [11].

High-performance computer technology continues to develop, and many countries around the world seek to break through in this area. At present, the high-performance CPU products and many manufacturers including, Intel and AMD, are largely monopolized by developed countries such as the United States and Japan. The processors of high-performance computers are roughly divided into general-purpose accelerated processors, multicore processors, and embedded processors. Intel produces multicore processors that include Haswell architecture, and the XeonE5 processor based on this architecture integrates 15 processing cores with a frequency of 2.6 GHz and a peak performance of 172Gflops or more In late 2014, the US Department of Energy announced the construction of the supercomputers "Summit" and "Sierra," to be installed at the Oak Ridge and Lawrence Livermore National Laboratories. The two supercomputers were jointly built by IBM and Nvidia and use a hybrid CPU + GPU architecture. The "Peak" system is used primarily for scientific computing; the "Ridge" system is used primarily for US National Nuclear Security Missions [13]. The US Department of Energy designed both systems using the IBM Open Power architecture, the Nvidia Tesla-accelerated computing platform, a heterogeneous computing model, and Nvidia NVLink high-speed GPU interconnects. Summit is equipped with devices capable of 150–300 Pflops of performance, with system-level application performance up to five times that of the Titan system. Each compute node includes multiple IBMPower 9 CPUs and multiple Nvidia Tesla GPUs based on the Nvidia Volta architecture. The supercomputer is powered by the NVIDIA/AMD GPU family of processors, Intel Xeon Phi processors, etc. There are two types of Intel Xeon Phi processors: Knight Ferry and Knight Corner. The Intel Xeon Phi processors come in two varieties: Knight Ferry and Knight Corner [14].

3. Analysis of Artistic Design Models for Parallel Computing of Heterogeneous Tasks

3.1. Heterogeneous Task Parallel Computing System Construction. The k-means algorithm is a well-known clustering analysis method widely used in artificial intelligence and data mining, such as bioinformatics, image segmentation, and information retrieval. In this chapter, based on the study of parallel optimization of data stream (n) and clustering prime number (k) in k-means algorithm, a novel multilevel hierarchical massively parallel optimization scheme is proposed and designed and implemented, in which not only data stream (n) and clustering prime number (k) can be partitioned in parallel, but also data dimension (d) can be partitioned in parallel so that n, k and d can all change independently; an intracore group is divided into two stages [15]. The US Department of Energy designed the two systems using the IBM Open Power architecture, the Nvidia Tesla-accelerated computing platform, a heterogeneous computing model, and Nvidia NV Link high-speed GPU interconnects. Among them, Summit is equipped with enough devices to achieve 150–300 Pflops performance, and the system-level application performance reaches five times that of the "Titan" system. The SW26010 heterogeneous multicore processor potential is fully exploited to solve the performance bottleneck problem of high-dimensional data processing, and the data dimension can reach 196,608 dimensions and the clustering prime can exceed 160,000; the cache-sensitive hierarchical mapping method is designed and implemented, which is able to handle high and low-dimensional data sets adaptively and efficiently; and the massively parallel communication optimization method, which can handle large-scale clustering problems and maintain high performance and scalability, significantly improve the performance of previous methods. Experimental results show that for $n = 1{,}265{,}723$, $k = 2{,}000$, and $d = 196{,}608$ scale datasets with 4,096 compute nodes (1,064,496 compute cores) applied in parallel, the parallel optimization method in this chapter achieves a performance of fewer than 18 seconds per iteration.

The goal of the unsupervised machine learning k-means clustering algorithm is to find a set of clustering points such that the average distance between the sample points and their nearest clustered center of mass is shortest, defined formally as given sample points n, $X_i = \{x\}, x \in R_d$, $i \in \{1, \ldots, n\}$, where each sample point is a d-dimensional data vector $X = (x_1, \ldots, x_n)$. Use u to denote the sample data dimension index, where $u \in \{1, \ldots, d\}$. The clustering prime $c = \{c_1, \ldots, c_n\}, c \in R_d, j \in \{1, \ldots, k\}$. The

minimization cost function O(C) of Lloyd's algorithm can be expressed as follows:

$$O(c) \leq \frac{1}{n^2} \sum_{i=1}^{n} f\left(x_d, c_d^i\right). \quad (1)$$

Note that these notation definitions are primarily from the work of Greg Hamersley and James Mewling et al. This chapter will customize the application of notation only when needed. In the above algorithm definition, the first step (Assign) is to assign each sample to the nearest cluster prime based on the Euclidean distance, and the second step (Update) is to compute the mean value in the specified d-dimensional sample vector space and update the cluster prime; the above two steps are repeated until each c is less than a specified threshold or the maximum number of iterations is exceeded. In this chapter, data, streams, and data samples are one concept and are often interchanged in use. The k-means algorithm is a well-known cluster analysis method that is widely used in artificial intelligence and data mining, such as bioinformatics, image segmentation, and information retrieval.

$$\mathrm{dis}\left(x_d, c_d^i\right) = \sum_{j=1} \left(x_u, c_j\right). \quad (2)$$

Since data-intensive computational problems usually require the implementation of many-to-many parameter synchronization statutes at large-scale scaling, the network topology-sensitive Allreduce operation is custom-implemented from the perspective of communication computation time optimization to reduce communication time. The main reasons for the relatively poor communication performance of the MPI_Allreduce routines provided by the compiler by default are the following: first, the network of supercomputers is characterized by high latency, so the MPI_Allreduce routines designed for low-latency network hardware are no longer applicable. The network of supercomputer achieves similar high bandwidth as the InfiniBand network, but has higher latency when the message length is greater than 2 KB, as shown in Figure 1, which compares the bandwidth performance of the supercomputer network with the InfiniBand FDR network; second, the communication pattern in the MPI_Allreduce routine is insensitive to the network topology of the supercomputer, while if each node in one supernode communicates peer-to-peer with a different node in another supernode, it will result in oversubscription across supernodes, and the bandwidth will be clipped to about a quarter of the full bandwidth when there is oversubscription between two supernodes; third, the MPI_ Allreduce routine data collection followed by statute operations (e.g. summation) is performed on the master core (MPE), which is limited by the computational performance of the master core (MPE), especially in the case of large number of parameters, and the communication is very inefficient. In total, 256 compute nodes are connected through a custom interconnect board to form a single supernode. All supernodes are connected to a central routing server. Internal supernode communication is more efficient than communication between supernodes. Therefore, to improve the communication efficiency of the k-means algorithm and synchronize the data of parameters such as clustered prime between compute nodes, this chapter customizes a low-latency Allreduce communication mechanism by using the basic network communication primitives such as MPI_Send and MPI_Recv. The core design idea is to use the slave core cluster array to perform statute operations after completing data collection (e.g., summation) to solve the computational bottleneck problem.

As can be seen from Figure 2, the parameter specification operation distributed over multiple compute nodes is divided into three phases. In the first phase, the binomial tree topology specification operation is executed within the super nodes and the result is a statute to one of the nodes (as the root node). In the second phase, the Allreduce specification operation is executed between the root nodes of different super nodes and each root node will get the result view of the entire Allreduce operation. In the third phase, the root node broadcasts the results to all other compute nodes in its super node [16]. To obtain better performance for large-scale scaling, ring topology and butterfly topology are adopted. Allreduce operation strategies are also investigated in this chapter, but the butterfly topology-based approach does not consider the hierarchical network structure characteristics leading to poor inter-supernode communication, and the ring topology-based approach contains more communication steps that do not apply to the high latency network characteristics of the "Shenwei-Taiko Light."

Figure 1 shows the schematic diagram of the Allreduce operation strategy between two super nodes using 64 KB and 1024 KB when the compute nodes are expanded from 2 to 2048, respectively. A binomial tree topology statute operation is performed first within the super node, then a full statute operation is performed between the super nodes, and finally, a broadcast operation is performed within the super node to complete the Allreduce operation between the compute nodes. As can be seen from the figure, the system's MPI_Allreduce operation has a significant increase in latency for the Allreduce operation when scaled to 64 compute nodes, while using the communication optimization method in this chapter, the latency grows slowly and linearly with lower latency, and has a 5–20 times performance improvement compared to the original Allreduce operation.

3.2. Artistic Design Model Design. With the rapid development of the Internet of Things (IoT), there is a significant heterogeneity in the type of business data, application expectation latency, and device type in the massive IoT services, for example, applications such as augmented reality, virtual reality, and driverless are often computationally complex, intensive, and have high latency requirements, while wearable devices, such as smartwatches and bracelets, have relatively low latency and computation requirements. In addition, due to the complex and time-varying network environment and

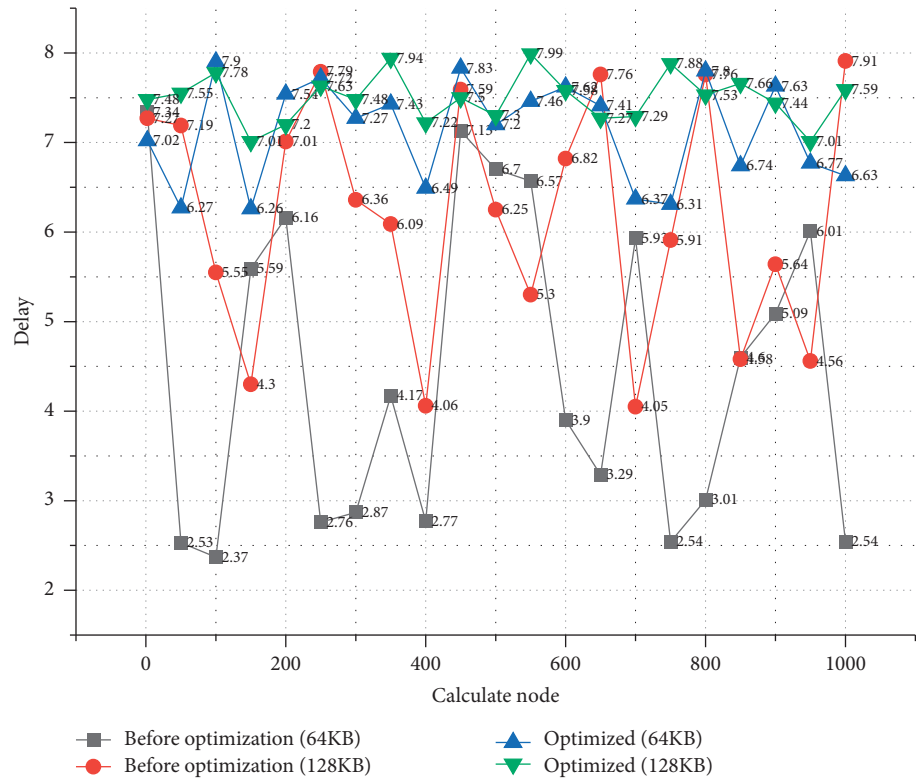

Figure 1: Performance test results before and after operation optimization.

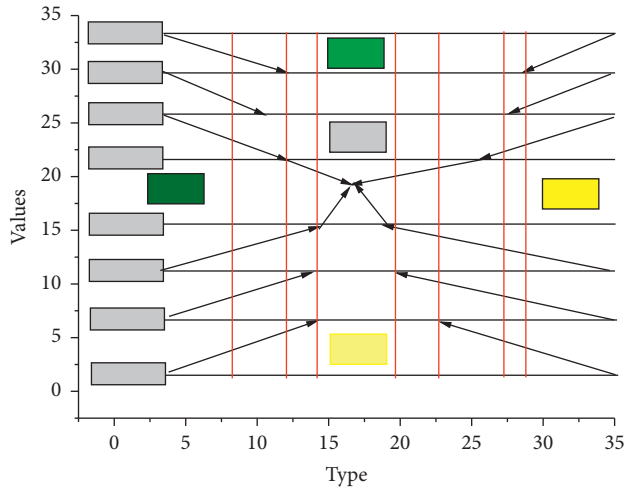

Figure 2: Schematic of Allreduce operation policy between the two super nodes.

traffic characteristics in the actual MEC-IoT, it is difficult to achieve efficient real-time resource allocation and excellent user experience, and there is an urgent need to dynamically optimize the energy consumption and latency of different applications and services. In addition, the collection of information from massive IoT devices and traditional centralized management and control have become extremely unrealistic [17]. Therefore, in future MEC-IoT scenarios, there is an urgent need to design a more efficient and flexible distributed optimization-based task offloading and resource on-demand allocation mechanism. In this chapter, a typical MEC-IoT heterogeneous task offloading queuing model is considered. Without the loss of generality, assume that the model has a total of M heterogeneous IoT devices or application services. Define m, to denote the ith IDAs, $V_i \in \{1, 2, \ldots, m\}$, n, to denote the jth MSs, $V_j \in \{1, 2, \ldots, n\}$. The entire offload system operates in discrete time slots in such a way that t is the length of a time slot t. A cache-sensitive hierarchical mapping method is designed and implemented, which can adaptively and efficiently process high-dimensional and low-dimensional data sets; at the same time, a large-scale parallel communication optimization method is also designed, which can handle large-scale clustering problems and maintain high performance and high scalability significantly improving the performance of previous methods.

If the process of task arrivals in IDAs obeys an independent Poisson distribution, define a to denote the number of tasks arriving in time slot t with an arrival rate of 1. Tasks arriving in IDAs will first be cached in the local task queue to be queued for processing, and define b to denote the number of tasks processed in time slot t by m. In addition, with the help of techniques such as large-scale multi-input multi-output (MIMO), IDAs can offload tasks to multiple MSs for processing at the same time. Let us denote the number of tasks offloaded from m to n, in time slot t, where f denotes the CPU computation frequency in HZ allocated to m, by

MEC server n, in time slot t, and L denotes the number of CPU cycles in cycles/bit required for a unit-bit task of m. For the sake of analysis, this chapter makes $j = 0$ when denoting local processing and $b = b + > b$, where bo denotes the number of tasks processed locally by m. We define the task queue backlog of m, at the start moment of time slot t. Further, the updated equation for the m task queue backlog can be obtained as follows:

$$Q_{t-1} = \min\left[Q_{t-1} + b_{t,i}\right] - a_{t,i}. \tag{3}$$

Heterogeneous computing systems can be divided into two categories according to the different types of implementations of computing units: first, on a single computer, multiple numbers and classes of computing units coordinate and cooperate to complete the computation—single-computer heterogeneous computing; second, multiple classes and numbers of computers connected into clusters through a network to complete the computation cluster heterogeneous computing. Single-computer heterogeneous computing can be further divided into two categories: multimodal heterogeneous computing and hybrid heterogeneous computing. Its powerful feature set provides developers with unparalleled creativity, with advanced sampling tools and effect modules, modular architecture, and advanced scripting, delivering ultra-accurate reproductions of real instruments, original standalone instruments, built-in sequencing capabilities, etc. Multimodal heterogeneous computing mainly reflects the spatial heterogeneity of the system; multiple subtasks are executed in parallel at the same time in different units with different instruction streams and data streams. Hybrid heterogeneous computing embodies mainly the temporal heterogeneity of the system, which supports tasks to complete computation at different moments in the form of different instruction streams and data streams. There are likewise two categories under cluster heterogeneous computing: homogeneous cluster heterogeneous computing and hybrid cluster heterogeneous computing. Homogeneous cluster heterogeneous computing refers to the collaboration of multiple computers of the same architecture and different performance to accomplish the same type of computational tasks, and hybrid cluster heterogeneous systems refer to the coordination of multiple computers of different architectures and different performance to cooperate to accomplish different types of computational tasks. The framework of the parallel computing model for heterogeneous tasks is shown in Figure 3.

There are two major sources of high-performance and low-cost computing power for heterogeneous systems: first, the rich variety of hardware resources with a powerful performance on board, and second, the task schedule that matches the rich hardware computing resources. In the case that the hardware configuration of a heterogeneous computing system has been determined, the effectiveness of task scheduling directly determines the computational performance of the system. Heterogeneous task scheduling is an NP-hard problem, and the optimal solution cannot be derived in polynomial time. Therefore, how to find the suboptimal solution that infinitely approximates the optimal solution is compelling in the field. The heuristic algorithm is an excellent solution strategy that has been arrived at after relentless exploration and experimentation, which can find a scheduling solution that is infinitely close to the optimal solution within a reasonable algorithm execution time.

The results obtained from the learning evaluation can provide various information of the learning process. The analysis and timely feedback of this information can effectively adjust and control all aspects of the learning process (including learning objectives) and allow students to understand their own learning in time, develop the existing advantages and disadvantages, constantly improve learning efficiency, and make learning activities enter a virtuous circle. In TripletRun, the processors within a computer node are treated as fully connected, and the links between different processors do not interfere with each other. Similarly, the links between different computer nodes do not interfere with each other. For a given link, whether within a computer node or between computer nodes, the link is exclusive when there is communication going on: subsequent data transfers begin only after all preceding transfers have been completed. TripletRun maintains a data structure designed to keep track of the time available for all network links. This design is slightly simpler, but reflects well the overhead from communication conflicts and is a good compromise between simulation accuracy and simulation speed. The class graph maintains all information about the set of task nodes and the set of edges in the graph, including the communication overhead of the edges, and provides a convenient interface for accessing them from the outside. TripletRun reads information from a JSON format file and uses it to initialize the directed acyclic graph. The main JSON fields of the directed acyclic graph are shown in Table 1 and contain three main fields, namely Nodes, Edges, and Speed Ratio, which represent the task nodes, edges, and speed factors in the program's directed acyclic graph, respectively. There are several secondary fields under the main fields. Although the specific proportion may not be very accurate, this is a metaphor, but at least we should take this aspect into account—occupy a certain proportion, after players play the game for a period of time, each of these rules will occupy less.

The algorithms related to large-scale data computation were initially implemented on general-purpose processors and therefore the related algorithm design, implementation, and optimization methods are also specific to general-purpose processors. Commonly used general-purpose processors today follow the von Neumann architecture [18]. Under the von Neumann architecture, the operation of programs is instruction based and instruction flow driven. In terms of performance optimization of algorithms, optimization efforts focus on reducing the number of instructions needed to implement an algorithm by optimizing its time complexity, thus reducing the program's running time. In addition, the data read and write of the program is implemented by the processor through external bus access to memory (memory + IO memory device), and the data read and write of the program increases the latency of the program due to the small bandwidth and slow speed of the external bus and memory. Therefore, the algorithm

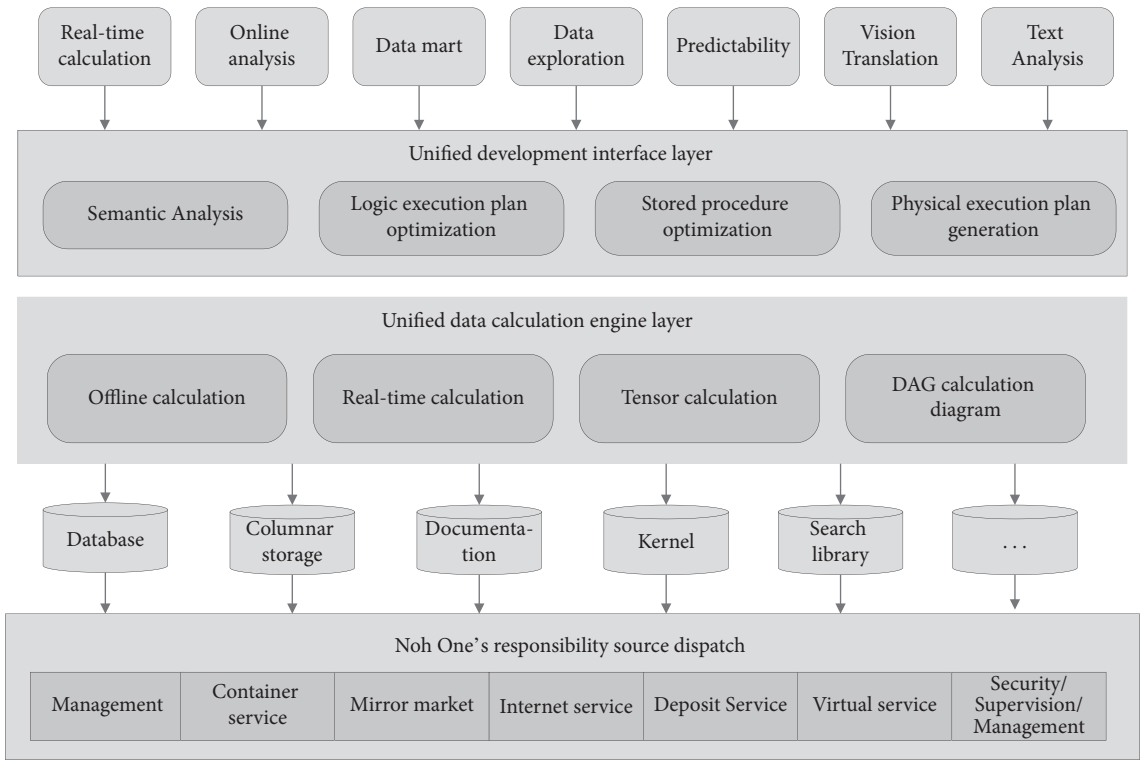

Figure 3: Framework diagram of the heterogeneous task concurrent computing model.

Table 1: Main fields in the definition of a directed acyclic graph.

Main field	Secondary field	Instruction	Unit	Type	Required
Edges	Src	Node identifier	Int	—	Yes
Nodes	Dst	Number of data generation	Float	Ops	Yes
Nodes	G	Destination node	Int	—	No
Edges	Weight	Processor type	Float	Mb	No
Speed	Ratio	Traffic	Int	—	Yes
Ratio	Type	Speed factor	Float	Kb	No

performance can also be improved by reducing the number of read and write memories, increasing data sharing, and other ways to improve the utilization of the on-chip cache (cache) and the efficiency of external memory access during the program design. As shown in Figure 4, FPGAs have a completely different hardware architecture compared to general-purpose processors. Most of the current mainstream FPGA chips have an island architecture. The logic block (LB), which implements the logic circuit, is distributed in matrix islands, and the islands are connected through wiring resources (wiring channels, switch block (SB) and connection block (CB)), and the chip communicates with the outside world through input/output Block (IOB). LB can be used as both computing and storage resources. This hardware architecture of the FPGA makes it break through the limitations of the von Neumann architecture. In FPGAs, the operation of the program is data flow driven. A rubric is an authenticity assessment tool, which is a set of criteria for evaluating or rating a student's work, achievement, growth record, or performance. It is also an effective teaching tool and an important bridge between teaching and evaluation.

HLS greatly improves the efficiency of algorithm design on FPGAs by translating algorithms described in high-level languages into HDL descriptions and provides a more efficient way to explore optimizing the performance of program implementations on FPGAs. In theory, the designer only needs to describe the behavior of the algorithm using the high-level language, while the timing, state management, and resource allocation are taken over by the HLS tool. HLS does not use the processor core as the basic unit of operation but is circuit oriented. The designer can direct the tool to perform parallel expansion and pipelining of structures such as for loops in the high-level language, thus enabling instruction-level parallelism and pipelining.

4. Results and Analysis

4.1. Performance of Parallel Computing Systems for Heterogeneous Tasks. A mature and stable heterogeneous computing system has the advantages of abundant computing resources, integrated task scheduling, and efficient task computation. The overall scheduling mechanism of a

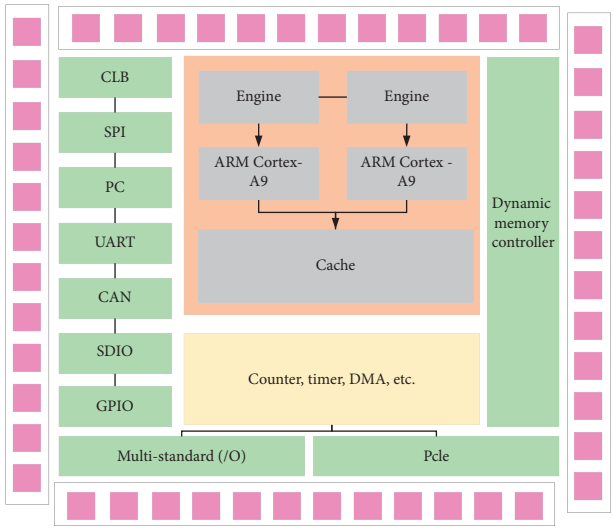

Figure 4: FPGA hardware architecture.

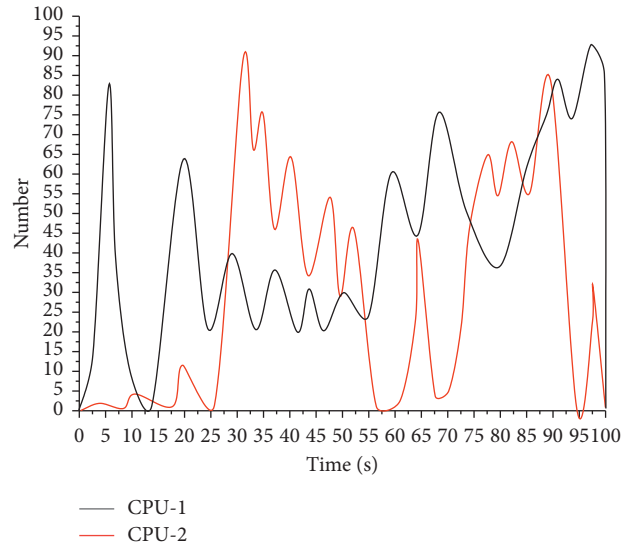

Figure 5: CPU and I/O utilization evaluation.

heterogeneous computing system, which is tailored to the "meta" of the system, allows each heterogeneous computing unit to complement each other's strengths and can significantly reduce the time overhead of computing [19]. The following is a concrete example to explain the basic principle of heterogeneous computing in detail. The practice of using evaluation scales to comprehensively evaluate students' learning, although formative evaluation has been widely used in English teaching evaluation, but due to the lack of objective and effective scales, students' performance is ultimately determined by one paper. Assume that the time required to execute a computational task on a standard serial computer is T. The time consumed by vector computation, multi-instruction multi-data stream computation, single-instruction multi-data stream computation, and single-instruction single-data stream computation is 30%, 36%, 24%, and 10% of the total computation time, respectively. Assuming that a vector machine is used to solve the problem, the speedups corresponding to each of the four types of subtasks performed by the vector machine are as follows: 30, 2, 8, and 1.25. Based on this speedup ratio, the total time T_i required by the vector machine to solve the problem together is as follows:

$$T = 0.8 \times \frac{T}{80} - 0.32 \times \frac{3}{T} + 0.14 \times \frac{T}{1.28} = 0.25T. \quad (4)$$

The speedup ratio S_i for solving this problem using vector machines compared to a standard serial computer is as follows:

$$S_i \approx \frac{T_i}{T} = \frac{0.25T}{T} = 2.21. \quad (5)$$

Different types of resource usage (including CPU, memory, and I/O interrupts) are examined, which are important performance indicators. Using SunwayMR to run the word frequency statistical algorithm, the resource usage during the processing of text file data (WordCount, 250 MB) is recorded, as shown in Figure 5. In the figure, the curves in the second subplot and the third subplot are somewhat consistent in their variation. As these two subplots show, the resource utilization increases very quickly during job execution but always keeps the growth smooth. The first peak occurs mainly due to reading data for the first data processing task, and the second peak occurs mainly in preparation for the reading of other data processing tasks. This indirectly reflects that SunwayMR has good parallel computing and communication capabilities. Immediately, the actual memory usage of the SunwayMR framework when running data processing programs is examined. The Spark framework uses many memory resources when running data-intensive programs. When running the relatively comprehensive PageRank algorithm, if the Spark memory size configuration is set too small, PageRank will struggle to complete the task successfully; the algorithm needs to be able to complete successfully by setting a larger memory size, for example, 10G or larger. Examining the performance comparison and memory usage between the SunwayMR framework and the Spark framework, running the same algorithm (with the same parameters configured), the memory usage per compute node is much less than that in the Spark framework. With the same data size input and the same configuration, the SunwayMR framework running the program will reduce the memory overhead by about 30–40%.

Lightweight detection/recovery model fault tolerance mechanisms will inevitably cause workload during job execution. It becomes important to quickly reduce the job execution recovery time after an abnormal error occurs during job execution. The effectiveness of the designed fault tolerance mechanism was verified by exhaustively examining job recovery and injecting errors into the distributed data parallel computing framework, SunwayMR, which quickly restarts applications by quickly reading execution commands. When testing benchmark performance, processing large-scale data quickly, the detection/recovery-based model fault tolerance mechanism appears to be more lightweight

and less expensive in terms of time and I/O load than the traditional checkpoint/recovery model fault tolerance mechanism (which requires saving a certain number of intermediate results), which verifies the effectiveness of the framework's internal design. After a job has failed to be restarted quickly more than three times, the program user is notified to intervene to recover the job. The reduction of parallel efficiency is mainly affected by the communication overhead, which indicates that in large-scale deep learning, how to alleviate the communication explosion caused by scale growth is one of the key issues to achieve efficient computing.

$$P_c^i \simeq \kappa_i f_{\max}^i. \tag{6}$$

The operation of a distributed data parallel computing framework requires the inclusion of master nodes and slave compute nodes, following a master-slave model pattern to build a heterogeneous cluster environment for processing large-scale data. The slave compute nodes are connected by networks such as GigaNet, InfiniBand, etc. Each compute node contains management cores and compute cores, and contains the core group (CG) identifier. Each slave compute node has a CG identifier ID number and the chips are connected with PCI Express. Within each CG, the cores are connected via PCIe. The programmer enters a command at the master node (or slave compute node) to start all compute nodes. This framework enables the lightweight deployment of data-intensive applications to an infrastructure-as-a-service physical facility. Message communication is based on the TCP/IP, socket programming protocol, providing high-performance, real-time distributed message interaction. Each task is distributed to each computer node or multiple computer nodes, thus relieving the pressure of data processing. Based on a master-slave dual-node message communication mechanism, the system evaluates each compute node's memory size, number of threads, etc. The master node and each slave compute node have message collectors that listen for messages (e.g., heartbeat messages) through listening ports. The compute nodes fetch tasks and data, wait for the execution of the job to complete, and then return the results to the master node, which merges the intermediate results and outputs the final job execution results.

4.2. Art Design Model Simulation Experiments. This chapter uses the message passing interface (MPI) communication mechanism to implement information interaction between compute nodes. The AES algorithm is implemented on the SW26010 heterogeneous multicore processor using a two-level multi-threaded programming model to parallelize within compute nodes by first starting four processes to four core groups using pthread threads and then initiating 64 lightweights at reading threads to 64 slave cores within a core group. The AES algorithm is programmed in C, except for the SubBytes operation, which is implemented in assembly code. The experiments were conducted on the Shenwei-TaikoLight supercomputer with a maximum of 1,024 compute nodes [13]. On a single SW26010 heterogeneous multicore processor with different core group operating modes, the input data block size increases almost linearly on the performance of the AES body. AES algorithm and the throughput increases from 1 KB to oEM1, 5 GR/s when the data are in block size. When the data block is larger than 16 MB, the block is larger than 16 megabytes. On the contrary, the AES algorithm performance does not grow linearly as the data block increases when the data block is larger than 256 MB. The performance remains the same when the data block is larger than 256 MB, which is caused by the lack of CPE number of computing power. To illustrate the impact of different optimization techniques more clearly on the performance of the AES algorithm, the impact of data block size on computation time, DMA data transfer time, thread start time, and double buffer overlap time, the running time of the algorithm in the single-core group operating mode is counted. It can be seen from Figure 6 that the main reason for the poor performance of the AES algorithm when the data block is less than 1 MB is caused by the thread start time.

When the amount of data allocated to each core group is less than 64 MB, the thread startup overhead is non-negligible, and to maintain system performance, it is necessary to ensure that the amount of data allocated to each core group is not less than 64 MB. To avoid the situation that the network bandwidth will be equally divided among the four core groups when simultaneous computation and communication occurs between different core groups within the same node, that is, there is communication competition for 1/4 network bandwidth clipping situation, this chapter uses the full slice sharing model for online real-time processing. The root process divides the task into N parts according to the task load balancing policy, scatter the task data to other processes, the other processes process the task data according to the received task data, and then send it to the root process after the processing is completed, the root process is responsible for gathering all the data, and so on repeatedly until the task is completed. To avoid network idleness during computation, a double buffer mechanism is used to achieve computation and communication overlap. Since the peak network bandwidth in both directions is 16 GB/s, the throughput of a single computing node of the AES algorithm reaches 13.49GB/s. The communication bandwidth has become a bottleneck for the scalability of the full-slice sharing model. In addition, to guarantee the processing performance of the AES algorithm, a certain data input block size needs to be guaranteed, but the available memory of the system is limited to a maximum of 32 GB and other factors also limit the scalability of the full-shared mode, as can be seen from Figure 7.

Most of the swarm intelligence algorithms have the same algorithmic process framework, with the main difference being in the way information is exchanged. On a single-core set, the VGG-16 network accelerated by the slave core array computes 10.42 times faster than the master core version. In the distributed training with data parallelism, the parallel efficiency reached 81.01% for 512 processes and 53.21% for 1024 processes. However, in 2048 processes, the parallel efficiency at $r = 8$ is only 25.34%, and the absolute speed is even lower than that of 1024 processes. The decrease in

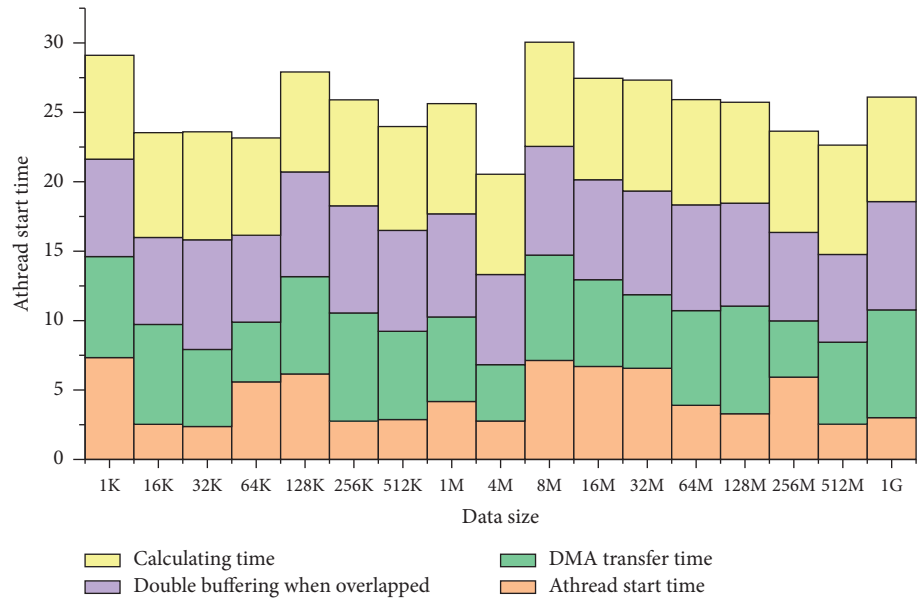

Figure 6: Effect of data block size as a percentage of algorithm run time.

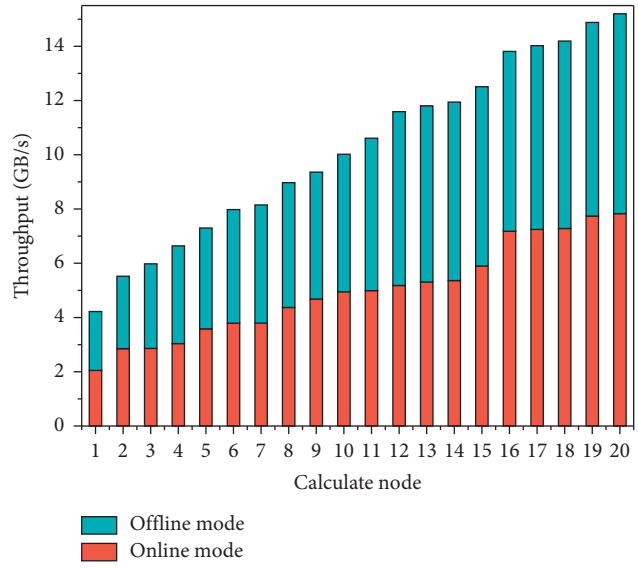

Figure 7: Scalability performance test results of AES algorithm with a different core set operating modes.

parallel efficiency is mainly affected by the communication overhead, which indicates that in large-scale deep learning, how to mitigate the problem of communication explosion due to scale growth is one of the key issues to achieve efficient computation.

5. Conclusion

In this article, we propose a design pattern for the implementation of high-performance algorithms based on FPGAs, and based on the proposed design pattern, we explore the improvement in the computational performance of algorithms implemented on FPGAs from two perspectives: redesigning algorithms according to the characteristics of the FPGA hardware architecture to treat the solution problem and reconfiguring existing algorithms to make the algorithm architecture fit the FPGA hardware architecture. A stochastic game-based resource allocation algorithm is proposed for the uncertainty of the queuing waiting time of MEC servers. First, a task queuing system model is established, and task local processing and MEC server processing are modeled by analyzing task offloading utility, offloading cost, upload time, and MEC server queuing time, respectively; then, for the uncertainty of MEC server queuing waiting time, multistage stochastic planning based on distributed buy-sell game theory and multistage stochastic planning theory is established for mobile. The existence of the SE equilibrium is proved by using distributed buy-sell game theory and multistage stochastic programming theory to model the buy-sell game based on multistage stochastic

programming for mobile terminal devices and MEC servers, respectively. Finally, it is verified through simulation that the proposed algorithm can effectively improve the success rate of task offloading while maximizing the long-term gains of system offloading. In the future, subsequent work in this article will target the application of the improved model of art design patterns based on heterogeneous task parallel computing for more in-depth exploration. The various challenges faced continue to enrich the content of the proposed design pattern and use more problem-solving examples to validate and optimize the design pattern.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this article.

References

[1] B. P. Bhattarai, S. Paudyal, Y. Luo et al., "Big data analytics in smart grids: state-of-the-art, challenges, opportunities, and future directions[J]," *IET Smart Grid*, vol. 2, no. 2, pp. 141–154, 2019.

[2] B. Jan, H. Farman, M. Khan, M. Imran, I. U. Islam, and A. Ahmad, "Deep learning in big data analytics: a comparative study[J]," *Computers & Electrical Engineering*, vol. 75, pp. 275–287, 2019.

[3] A. Zappone, M. Di Renzo, and M. Debbah, "Wireless networks design in the era of deep learning: model-based, AI-based, or both?[J]," *IEEE Transactions on Communications*, vol. 67, no. 10, pp. 7331–7376, 2019.

[4] S. Memeti, S. Pllana, A. Binotto, J. Kołodziej, and I. Brandic, "Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review[J]," *Computing*, vol. 101, no. 8, pp. 893–936, 2019.

[5] S. A. Enger, J. Vijande, and M. J. Rivard, "Model-based dose calculation algorithms for brachytherapy dosimetry[C]// Seminars in radiation oncology," *WB Saunders*, vol. 30, no. 1, pp. 77–86, 2020.

[6] J. Nunez-Yanez, S. Amiri, M. Hosseinabady et al., "Simultaneous multiprocessing in a software-defined heterogeneous FPGA[J]," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4078–4095, 2019.

[7] Y. Huang, C. Shao, B. Wu, J. L. Beck, and H. Li, "State-of-the-art review on Bayesian inference in structural system identification and damage assessment[J]," *Advances in Structural Engineering*, vol. 22, no. 6, pp. 1329–1351, 2019.

[8] C. Savaglio, M. Ganzha, and M. Paprzycki, "Agent-based Internet of things: state-of-the-art and research challenges[J]," *Future Generation Computer Systems*, vol. 102, pp. 1038–1053, 2020.

[9] H. Akhavan-Hejazi and H. Mohsenian-Rad, "Power systems big data analytics: an assessment of paradigm shift barriers and prospects[J]," *Energy Reports*, vol. 4, pp. 91–100, 2018.

[10] S. K. Panda and P. K. Jana, "An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems," *Cluster Computing*, vol. 22, no. 2, pp. 509–527, 2019.

[11] A. A. Munshi and Y. A. R. I. Mohamed, "Data lake lambda architecture for smart grids big data analytics[J]," *IEEE Access*, vol. 6, pp. 40463–40471, 2018.

[12] K. Cao, J. Zhou, P. Cong et al., "Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems[J]," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1189–1202, 2018.

[13] A. Abeshu and N. Chilamkurti, "Deep learning: the Frontier for distributed attack detection in fog-to-things computing [J]," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 169–175, 2018.

[14] A. Ahmad and S. S. Khan, "Survey of state-of-the-art mixed data clustering algorithms[J]," *Ieee Access*, vol. 7, pp. 31883–31902, 2019.

[15] T. Wei, J. Zhou, K. Cao et al., "Cost-constrained QoS optimization for approximate computation real-time tasks in heterogeneous MPSoCs[J]," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1733–1746, 2017.

[16] C. Chen, K. Li, A. Ouyang, Z. Tang, and K. Li, "GFlink: an in-memory computing architecture on heterogeneous CPU-GPU clusters for big data[J]," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1275–1288, 2018.

[17] W. Zhang, R. Zhang, C. Wu et al., "State-of-the-art review of soft computing applications in underground excavations[J]," *Geoscience Frontiers*, vol. 11, no. 4, pp. 1095–1106, 2020.

[18] W. T. Su, W. F. Pan, and C. C. Chen, "Context-aware task assignment for MapReduce in heterogeneous clouds[J]," *Sensors and Materials*, vol. 29, no. 11, pp. 1497–1512, 2017.

[19] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms[J]," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2176–2190, 2018.