Power Grid Data Monitoring and Analysis System Based on Edge Computing

Wang Tianyou, Qin Yuanze, Huang Yu, Lou Yiwei, Xu Chongyou and Chen Lei

July 10, 2022

Power Grid Data Monitoring and Analysis System Based on Edge Computing

1st Tianyou Wang
National Engineering Research Center for Software Engineering
Peking University
Beijing, China
240993819@qq.com

2nd Yuanze Qin
National Engineering Research Center for Software Engineering
Peking University
Beijing, China
qyz@stu.pku.edu.cn

3rd Yu Huang
National Engineering Research Center for Software Engineering
Peking University
Beijing, China
hy@pku.edu.cn

4th Yiwei Lou
National Engineering Research Center for Software Engineering
Peking University
Beijing, China
cyfqylyw@gmail.com

5th Chongyou Xu
State Grid Ningbo Electric Power Supply Company
Ningbo, China
hello1018@126.com

6th Lei Chen
State Grid Zhejiang Electric Power Co. LTD.
Hangzhou, China
chenlei3909@163.com

***Abstract*—With the continuous accumulation of large-scale power grid data, the traditional centralized data analysis method is more and more expensive for data transmission. Based on this, we designed a grid big data monitoring and analysis system and transferred the computation process to the edge node close to the data source through an edge computing strategy. On the one hand, data processing and data analysis algorithms are encapsulated by container technology, and the algorithm is mirrored to the edge nodes of the power network through the system to complete the computation. On the other hand, the computing clusters are deployed at the edge nodes of the power network, which is responsible for the scheduling, execution, and status monitoring of computing tasks. Computing tasks can be flexibly managed in a cluster by extending user-defined resources. Through the reserved parameters, users can intervene in task execution policies, and tasks can be configured. The edge node sends the calculation result or early warning information to the central monitoring service through the asynchronous message. Compared with the traditional centralized data analysis system, the proposed method relieves the problem of the overhead of massive data transmission in the network, reduces the application cost, helps to apply the data analysis to more edge side nodes, and fully excavates the potential value of grid data.**

***Index Terms*—smart grid, edge computing, data monitoring, data analysis**

I. Introduction

In 2019, State Grid put forward the construction strategy of "three types and two networks" [1], which is an important measure to improve the reliability and stability of the power supply. With the promotion of electric power construction of Internet of things, large-scale sensing equipment and intelligent terminals brings more abundant data resources [2], [3], which also efficiently collect the information of the key power equipment operating state and environment state in each link of power supply [4]. How to take good use of these power grid business data, mine its potential value, and make it play a role in the actual production of power grid enterprises has become an important topic that researchers pay attention to. With the gradual popularization of big data, cloud computing and machine learning technologies in the traditional industrial field, the application of these technologies in the power grid field is also promoting and achieving remarkable results, which brings development motivation for smart grid. As the number of intelligent terminal devices increases gradually, the power grid service data accumulates in the edge nodes of the power grid network. This presents a great challenge to the traditional "acquisition + concentration" power monitoring analysis. In particular, the problems of high transmission cost of data in the network and heavy computing load of the master station are prominent [5]. Therefore, how to make good use of the advantages of grid edge nodes close to data sources and give full play to the role of edge nodes in grid business data processing and analysis is the key to solve this problem.

II. Related Work

A. Edge computing

The basic idea of edge computing is to run computing tasks on computing resources close to data sources to reduce transmission overhead and improve data security [6].

Edge computing was first proposed in the content distribution network, which relies on the content distribution platform to schedule requests of users' access to the nearest cache server thus speeding up content acquisition, bringing a better experience for users, and improving network quality. Edge computing refers to a new computing model that performs computation at the edge of the network [7]. At present, edge computing has been applied to practical scenarios such as smart homes and smart cities [8], creating value in various fields with its efficient computing model and supporting the emerging application of the Internet of everything.

With the large-scale access of smart terminals such as smart meters, massive power grid service data accumulates at the edge of the power grid network, edge computing is gradually paid attention to in the field of the power grid, and the computing capacity of edge nodes is gradually improved. In this paper, combining with the actual demand to design a system strategy, at the edge of the calculation on the edge of the grid node deployment compute cluster, will be distributed computing tasks from center to edge, transferring processes from the central to the edge, combined the technology of data mining and machine learning, near the location of the data source to complete data analysis, aimed at reducing massive power grid data in the network transmission overhead, Improved edge data utilization is applied to power distribution status analysis and device status analysis to improve power supply service quality.

B. Container technology

Docker, as a kind of container technology, is the most used container engine in the community. Compared with traditional virtual technology, Docker is more lightweight, occupies fewer resources, and can ensure isolation. Developers can easily containerize applications.

Due to the advantages of Docker technology and the requirements of our system, we use Docker technology to build the image of the basic algorithm, encapsulate the data processing, calculation logic of data analysis, and the required dependent environment in the image, and use the version control ability of the image to manage the version of the algorithm. Using the portability of the container, the algorithm is plug and play, and the computing task is sent to the edge nodes of the power grid for quick start and operation through mirroring, and the data processing and data analysis are completed at the edge nodes. In this way, the algorithm development process can be separated from task operation, and algorithm development needs to pay attention to the correctness of the application running in the mirror. Through the container packaging dependent libraries and applications, it can ensure that the application has been run in the same environment. In the task running, the system only needs to provide the necessary computing resources and storage space for the container.

Kubernetes [9], [10] can complete the functions of resource scheduling, deployment, and operation. Among them, API Server is the central nerve of the cluster. All operations on resources are carried out through it, and communication of many components also needs to go through API Server. Etcd stores the status information of the entire cluster, such as the information of each Node in the cluster, and API Server is responsible for communicating with it. The Controller Manager is responsible for maintaining the normal running of the cluster. For example, if a Pod exits abnormally due to an error, the Controller Manager is responsible for recreating the Pod on the Node. The scheduler is responsible for resource scheduling. Based on Pod status information provided by API Server, Scheduler allocates resources to run Pod on Node. Kubelet is a component on Node that is responsible for creating, running, automatically repairing containers on Node and interacting with the API Server to update their status. Kube-proxy, like Kubelet, is a component of Node. It provides the ability of load balancing and Service discovery, and also maintains Service status.

C. Distributed machine learning

With the improvement of the scale of training data and the increasing of training models, the problem of insufficient performance of a single machine has brought many inconveniences, and distributed machine learning technology [11]–[13] has gradually attracted people's attention. At present, parallelism is mainly considered from data and the model. Data parallelism is suitable for the case where the number of model parameters is small but the number of training data is large. All data is divided into multiple data blocks, and each node obtains a subset of all data without overlapping. A complete copy of the parameters is stored at each compute node, and the local copy of the parameters is trained with the data subset of the node. The parameter updates of each node are summarized through the parameter server. The advantage of data parallelism is that multiple computing nodes do not interfere with each other, and the parallel stochastic gradient descent algorithm applies the method of data parallelism [14]. Model parallelism is suitable for large models [15], and a large model can be divided into multiple small modules. The advantage of model parallelism is that large models can be processed and conflicts of model updates can be avoided, and faster convergence can be achieved through scheduling among multiple compute nodes [16].

III. THE PROPOSED SYSTEM

A. System architecture

The system adopts a three-layer architecture design scheme, as shown in Figure 1. Users interact with the system through the user layer, can access and manage node information, algorithm information, application information and task information, and can create computing tasks through the user layer interface and deliver them to the target edge node. The service layer manages node information, algorithm information, application information, and task information and interacts with the storage layer to manage the data information in the system. In addition, the service layer can interact with the edge nodes compute clusters through the Kubernetes cluster management module. On the one hand, the computing tasks can be created to run and condition monitoring. On the other hand, more detailed cluster running status information can be obtained and displayed to the user via the user layer, helping users understand the state of the cluster running.

At the user layer, users can interact with the system through the Web front-end page, and manage nodes, algorithms, applications, and tasks through the user client interface. The service layer is divided into information management and Kubernetes cluster management module. The information management

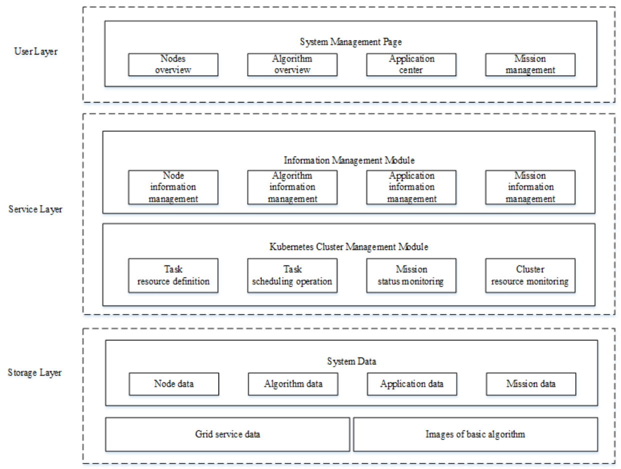
User Layer
System Management Page
Nodes overview
Algorithm overview
Application center
Mission management
Service Layer
Information Management Module
Node information management
Algorithm information management
Application information management
Mission information management
Kubernetes Cluster Management Module
Task resource definition
Task scheduling operation
Mission status monitoring
Cluster resource monitoring
Storage Layer
System Data
Node data
Algorithm data
Application data
Mission data
Grid service data
Images of basic algorithm

Fig. 1. System architecture

module responds to users' queries, additions, and deletions of node information, algorithm information, application information, and task information, providing users with system data management capabilities. Kubernetes cluster management module is the interface between the system and Kubernetes cluster interaction, responsible for the task resource definition registered in the cluster, the task scheduling operation, task status monitoring, and cluster resource usage monitoring. The storage layer is responsible for mirroring storage system data, service raw data, and basic algorithms.

B. The plan of Edge computing

To alleviate the data transmission overhead cost for the grid network edge node for data analysis, this paper designs an edge computing scheme based on the container and Kubernetes technology and applies that to the power grid data monitoring and analysis system. By distributing algorithm images to the edge side of computing clusters, which replaces the upload grid business data to the data center, it takes full advantage of edge nodes' proximity to data sources. When the computing task is delivered to the edge node, a data block has been completed and the data scale has been reduced. Combined with the system's support for distributed computing tasks, multiple working nodes can be started to jointly complete the computing task. The calculation results of edge nodes are sent back to the central monitoring service by asynchronous message. The overall scheme of edge computing is shown in Figure 2.

In the edge computing scheme, algorithm mirroring is the basis of all computing tasks. In order to facilitate users to execute policies through the system configuration algorithm, the algorithm accessing the system needs to configure interfaces according to certain specification reserved parameters and follow the environment variable acquisition specification defined by the system.

Computing tasks exist in Kubernetes cluster of edge nodes in the form of user-defined resources. Users create computing tasks through the system and send them to edge nodes.

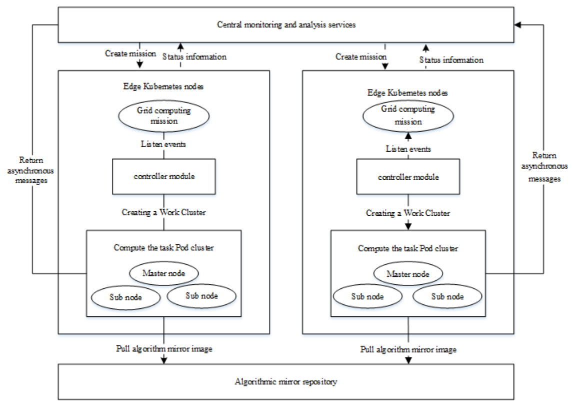

Fig. 2. Edge computing overall scheme

After listening to the creation event, the custom controller will allocate resources and create work clusters to complete computing work. Before the task is run, the image related to the task will be pulled to the edge node, and the execution parameters and commands will be input to make it run in the Pod, and the operation will end after the corresponding task is completed. For computing tasks requiring network topology information such as distributed machine learning, related information can be written into Pod environment variables through user configuration and predefined templates in the system, which can be directly obtained when the task is running. Asynchronous messages can be sent to the service's message management module for tasks that require feedback on run results.

In order to enhance the universality and expansibility of the system, the common tasks of power grid data statistics, data processing, and data analysis are fully considered in the system design. In that way, the general task creation mode and state management strategy are designed. The operation of computing tasks is managed through a customized controller. At the same time, the differences between different types of algorithm tasks are transferred to the upper layer by means of predefined templates and user-defined environment variables, which ensures the stability and universality of the core module of the system and enables the unified management of various types of computing tasks.

C. Computing task management

The delivery process of a computing task in the system is shown in Figure 3, which includes the following steps: node registration, algorithm, application registration, task construction, delivery, task creation in the cluster, task running, and sending the calculation results to the message management module of the monitoring and analysis system.

When registering a node, configure node connection parameters and save node information after the connection succeeds. When registering an algorithm, configure the image address information and environment variable parameters that the algorithm depends on. Select an algorithm image and run the command when registering the application. The steps in

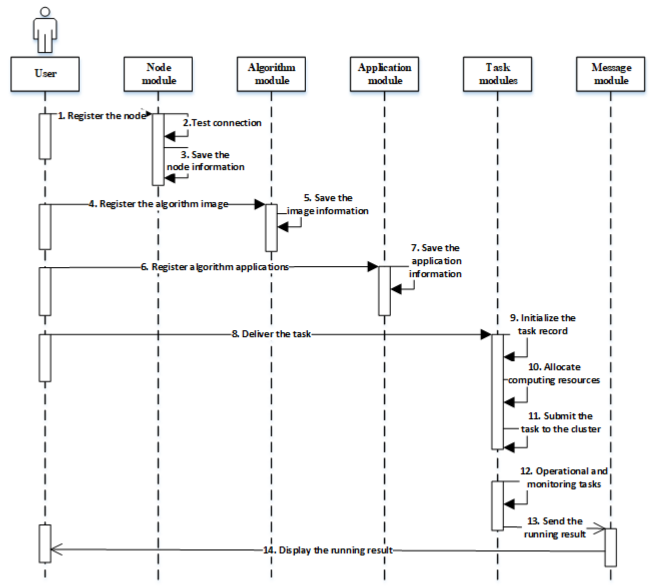
User
Node module
Algorithm module
Application module
Task modules
Message module
1. Register the node
2.Test connection
3. Save the node information
4. Register the algorithm image
5. Save the image information
6. Register algorithm applications
7. Save the application information
8. Deliver the task
9. Initialize the task record
10. Allocate computing resources
11. Submit the task to the cluster
12. Operational and monitoring tasks
13. Send the running result
14. Display the running result

Fig. 3. Work flow of the proposed system

the node registration, algorithm, and application registration processes are not required during most task creation. They are required only when the node, algorithm, or application is registered with the system for the first time. Most tasks are executed directly from the task delivery process.

When a task is delivered, the user arranges the task workflow based on the created application, determines the execution sequence of the application, configures the execution period of the task, and selects the edge node to deliver the task. After the task is created, it is stored as a database record in the relational database of the system. The task creation module will read the task information in the database after listening to the task creation, submit the task to Kubernetes cluster, and give the specific task creation to the task's custom controller for processing.

The process of creating a task in a cluster starts when the custom controller of the task resource listens to the creation event of the task resource in the cluster. The custom controller of the task resource reads the newly created task object in the cluster, constructs the required Pod resource and Service resource according to the task template information submitted to the cluster, and injects related information into the Pod according to the environment variable parameters configured for the algorithm by the user in the system, to be obtained when the container starts. The purpose of the Service is to provide a stable access interface to the Pod and solve PodIP changes in the event of a Pod restart. The results of the task are finally presented to the user through asynchronous messaging.

The steps that a task goes through are encapsulated in an algorithm image. Relatively generic steps include command, parameter, and environment variable parsing, data reading, algorithm execution, results, and feedback. The execution steps of the specific task calculation are determined by the algorithm developer in the development of the algorithm, but the parameter parsing, environment variable parsing and result sending parts need to follow the development specifications of the system. The relevant functions make it possible to interact with the task management module and message management module correctly.

IV. Multiple Algorithm Adaptation

In the design of this system, the universality of the system is fully considered, and it can support the operation of various data statistics tasks, data processing tasks and data analysis tasks. Aiming at the machine learning task, the system realizes the adaptation of XGBoost algorithm and LightGBM algorithm which achieve excellent performance in all kinds of data science competitions. In addition, in order to support the calculation and analysis of massive data and to support the operation of distributed algorithms, the system introduces the start of multi-copy distributed computing. The algorithm and its dependencies are encapsulated in the container by Docker container technology, and the command interface and parameter configuration interface are reserved. The algorithm is stored in the image repository. After the image address and variable parameter are registered in the algorithm management module, users can adapt the algorithm to build applications in the system, and then create computing tasks and send them to edge nodes for operation.

The general data statistics and data processing tasks only interact with the task management module of the system. However, the system needs to solve the problem of calculation and analysis of massive data. It needs to start several working nodes for distributed computing. Therefore, two distributed machine learning algorithms are taken as examples to introduce in detail the design and implementation of adaptation of algorithms with obvious characteristics in the proposed system.

A. The design and implementation of XGBoost algorithm access adaptation

XGBoost (eXtreme Gradient Boosting) [17] is the most famous Boosting algorithm, which has been used in some work of major Internet companies and gradually plays a role in the industry.

When the XGBoost algorithm adapted in the system, it is necessary to pay attention to the task management module and ensure that can support stand-alone operation and distributed operation. In addition to the conventional data processing and model training code logic, because the algorithm image will eventually be delivered to the Edge node Kubernetes cluster to run, the algorithm execution command and data parameter configuration need to be specified by the user at the time of creation. Moreover, the algorithm needs to reserve parameter interfaces, which are injected into the runtime container by the task management module of the system. In addition, the network needs to be set up in the distributed scenario. When the node runs, it needs to obtain the request address, port of the central node, and the serial number of the current node in the whole work cluster to complete the training task of specific data blocks. Information such as the IP address of the working node can be obtained only when the task is created, in which the task cannot be configured in advance. In that way, the system records information such as IP address and node number when creating Pod in the task creation module of the system. The related configuration of network topology information is passed to Pod by writing environment variables, so that the related configuration information can be read directly from environment variables in the algorithm mirroring. The pseudocode of XGBoost on computation task running is shown in Algorithm 1.

Algorithm 1 XGBoost on computation task running

```
Input: $args$ - The list of parameters input by the user
Output: $model$ - The generated model after training
1: $user\_args$ ← get_user_args($args$); // Parses parameters from user input commands
2: $master\_addr$, $port$, $rank$,
   $world\_size$ = extract_xgb_env(); // Obtain cluster network topology information from environment variables
   // $master\_addr$: Master node address
   // $port$: port number
   // $rank$: Current node number
   // $world\_size$: Number of compute nodes
3: if $world\_size < 1$ then
4:     if $rank = 0$ then
5:         start_rabit_tracker($master\_addr$, $port$,
           $rank$, $world\_size$);
           // If the distributed algorithm is to run, the node with rank 0 starts as the master node and is responsible for overall coordination
           // Obtain assigned data according to the sequence number and the node number
           // Read data based on the user's input address
6:     end if
7:     $train\_data$ ← read_$train\_data$($rank$, $world\_size$, $user\_args$.path);
8: end if
9: $model$ ← xgb.train($train\_data$, $user\_args$); // Pass in parameters and data that the user input to complete the training process
10: if $rank = 0$ then
11:     return $model$; // The Master node returns the trained model and stores it in the configured path
12: else
13:     return NULL;
14: end if
```

The system needs to support multiple types of computing tasks. Considering flexibility and compatibility, the system supports two methods to obtain network topology information through environment variables.

For the algorithm developed for the specific business scenarios of the power grid, the complete network topology information can be read directly from the environment variable $network\_info$, and then the specific fields can be used as required. Popular algorithm frameworks in the industry often have unique variable acquisition specifications. Reading environment variables from $network\_info$ needs to supplement environment variable conversion logic during algorithm development, which is not friendly to algorithm developers and is not convenient to use. Although the templates configured by system developers for different algorithm frameworks can be well compatible with the implementation of open-source frameworks, they are not flexible enough, requiring frequent modification of the core code to support the access of algorithms, which increases the complexity of system maintenance. Therefore, the ability to construct environment variables is open to the user in this system. For environment variables that can be set directly by the user, the user can assign values directly. For variables such as network topology information, whose values can be determined only during running, the user only need to configure template placeholders. When environment variables are injected, the system detects the environment variables configured based on the template, obtains information such as the network address from the $network\_info$ structure, and replaces placeholders in the template with valid information. It should be noted that the meaning and presentation of placeholders in the template follow the system specification. In this way, users can configure algorithms based on templates to create environment variable formats that conform to open-source framework specifications, which the system can inject into the Pod at running time. In most cases, a new algorithm can be supported for use in the system without the system modifying the code. In this paper, the construction of environment variables injected into the Pod needs to support two types so that different algorithms can be adapted in the system for use. The pseudocode is shown in Algorithm 2.

When the container encapsulating XGBoost algorithm is started in the cluster, the network topology information of the entire cluster of working nodes has been written into the environment variable. These include the $master\_addr$, $master\_port$, $rank$, and $world\_size$ parameters required for the execution of the distributed XGBoost algorithm, which can be used by reading directly from the environment variable.

B. Design and implementation of LightGBM algorithm access adaptation

The gradient lifting tree is a very popular model in machine learning for a long time. LightGBM [18] is an efficient framework for implementing the gradient lifting tree, which has an excellent performance in training speed, memory consumption and accuracy. LightGBM's efficient support for parallelism makes it easy to use in distributed computing scenarios.

This system is based on the open-source LightGBM framework of Microsoft Asia Research Institute, which supports the delivery of the computation tasks of the LightGBM algorithm in the system, and realizes the operation of distributed tasks based on socket communication. The specific adaptation methods are as follows. First, under the guidance of the official documents of the open-source project, the operating

Algorithm 2 Construct the environment variable input into the Pod

```
Input: $network\_info$ - The structure maintained during construction, containing detailed network topology information in the working network
       $user\_env\_str$ – Character string in json format, containing user-configured environment variables
Output: $pod\_env\_list$ – The list of environment variables for Pod, containing the pairs of key-value.
1:  Initialize $pod\_env\_list$; // Input complete network topology information environment variables
2:  $pod\_env\_list \leftarrow$ add_network_info( $pod\_env\_list$, $network\_info$); // A list of user-defined parameters is parsed from the commands input by the user. Each variable is a key-value pair
3:  $env\_var\_list \leftarrow$ get_user_env_from_str($user\_env\_str$); // Iterate over the argument list
4:  while $x$ = 0 to $len(env\_var\_list) - 1$ do
5:      $env\_var \leftarrow$ en_var_list[$x$]; // Each variable is a key-value pair, where the key is the name of the environment variable and the value is the value or template
6:      if is_template($env\_var$.value) then
7:          $env\_var$.value $\leftarrow$ replace_template($env\_var$.value, $network\_info$); // If it is a template, replace the value with $network\_info$
8:      end if
9:      $pod\_env\_list$.add(env.var); // After replacing the information, add it to the Pod environment variables list as key-value pairs of environment variables
10: end while
11: return $pod\_env\_list$; // Returns a list of constructed Pod environment variables to be input to the Pod template
```

dependencies and operating environment of the LightGBM framework are encapsulated into the algorithm image. At runtime, a training parameter configuration file and node information configuration file are constructed according to user configuration, and a training task can be started according to the two configuration files.

Since the algorithm runs in the Pod created by Kubernetes cluster during execution, relevant parameters constructing the configuration file need to be obtained from environment variables. The code logic for injecting environment variables during Pod creation is common for different types of algorithms, similar to IV-A. LightGBM algorithm needs to obtain the Master node address, Master node port, all Worker node address, Worker node port, total number of nodes in the work cluster, and the serial number of the current node from the environment variables. After obtaining the above information, according to the obtained network topology information and user-configured execution parameters, create a training parameter configuration file, write the training parameters required by the LightGBM framework, create a node information configuration file, write the IP address or domain name and port information of each node in the work cluster. Finally, run the command to train the model. There is no data file processing involved in the code of the LightGBM framework for running computing tasks. Data files are directly transmitted to the framework level for reading and processing through user configuration parameters. The pseudocode is shown in Algorithm 3.

Algorithm 3 Computation tasks based on LightGBM framework

```
Input: $args$ – The list of parameters entered by the user
Output: $model$ –The generated $model$ after training
1:  $master\_addr \leftarrow$ get_env_var("MASTER_ADD"); // Obtain the address of the master node
2:  $master\_port \leftarrow$ get_env_var("$master\_port$"); // Obtain the port number of the master node
3:  $worker\_addrs \leftarrow$ get_env_var("$worker\_addrs$"); // Get the worker node address list
4:  $worker\_port \leftarrow$ get_env_var("$worker\_port$"); // Get the port number of the worker node
5:  $world\_size \leftarrow$ int(get_env_var("$world\_size$"); // Obtain the total number of nodes
6:  $rank \leftarrow$ int(os.environ["$rank$"]); // Obtain the current node serial number
7:  $machine\_conf\_file\_path \leftarrow$ generate_machine_conf_file($master\_addr$, $master\_port$, $worker\_addrs$, $worker\_port$); // Based on the obtained configuration information, create the storage compute node address and port configuration file
8:  if $rank$ = 0 then
9:      $local\_port \leftarrow master\_port$;
10: else
11:     $local\_port \leftarrow worker\_port$;
12: end if
13: $config\_file\_path \leftarrow$ generate_train_conf_file($machine\_list\_file\_path$, $world\_size$, $args$.output_$model$, $local\_port$, $args$); // Create a training profile based on the environment variable configuration information and user input parameters
14: train_LightGBM($config\_file\_path$); // Pass the training profile path to start the training
```

According to the adaptation process of the above two algorithms, different types of machine learning algorithms are submitted to the computing cluster through the same process. Before container startup and algorithm code execution, the system allocates resources, starts resources and manages resources for computing tasks according to the general process. When the container is started and the algorithm code is executed, the difference between algorithms will be perceived through the environment variables injected into Pod. The working node of distributed machine learning algorithm needs to read the node role, node serial number and network topology information from the environment variables to interact with other working nodes. In common distributed computing tasks, nodes do not need to interact with each other, but only focus on node roles and node serial numbers.

V. Conclusion

This paper designed and implemented power grid data monitoring and analysis system based on edge computing. First, we use container technology to encapsulate the computing logic in the image, and then deliver the algorithm image from the central service to the edge node instead of uploading data from the edge node to the central service. Finally, the data analysis process is completed on the edge side near the data source. In this way, the cost of data transmission is reduced, data analysis technologies such as machine learning are applied to power grid business scenarios, and the value of the grid data is fully mined. This paper completed the following work:

(1) Investigate the research status of machine learning technology application in smart power grid at home and abroad. Aiming at the problem that a large amount of data is summarized to the center service leading to the huge transmission cost, a kind of edge computing scheme is designed. Based on the container technology, the algorithm image is delivered to the edge Kubernetes cluster to complete computing tasks, and the universality of the scheme is fully considered to facilitate the access of a variety of algorithms.

(2) Design and implement the power grid big data monitoring and analysis system based on edge computing. Utilize Kubernetes to manage the creation, running, and status monitoring of computing tasks by extending user-defined resources. Through container technology, images of XGBoost algorithm and LightGBM algorithm are constructed according to the algorithm adaptation of the system, and registered in the system to realize the construction and delivery of relevant computing tasks. In addition, the system fully considers the convenience of accessing various types of algorithms, and provides flexible configuration interfaces at the user layer. It can be extended without changing the core modules of the system.

VI. Acknowledgment

This study was funded by Science and Technology Program of State Grid Corporation of China (No.5400-202019120A-0-0-00).

References

[1] Y. Lee, “Focus on ”three types and two networks, world-class” to promote the high-quality development of provincial power grid enterprises (in chinese),” *State Grid*, no. 4, p. 2, 2019.

[2] Z. Li, F. Chen, B. Li, C. Deng, and Z. Tian, “Research on application of hybrid distributed and point optical fiber sensing mechanism in substation (in chinese),” *Electric Power Information and Communication Technology*, vol. 18, no. 11, p. 6, 2020.

[3] J. Guo, Y. Liang, C. Chen, S. Chen, Y. Lu, and H. Huang, “Challenge and application prospect of power intelligent sensing technology (in chinese),” *Electric Power Information and Communication Technology*, 2020.

[4] J. Liu, Z. Zhao, and X. Ji, “Research and application of internet of things in power transmission and distribution system (in chinese),” *Chinese Journal on Internet of Things*, no. 1, p. 15, 2018.

[5] S. Zhang, J. Tong, Y. Zhang, M. Zhang, Y. Lei, and Y. Zhu, “Research on edge computing technology for intelligent sensing layer of energy interconnection,” *Electric Power Information and Communication Technology*, vol. 18, no. 4, pp. 42–50, 4 2020.

[6] W. Shi, X. Zhang, Y. Wang, and Q. Zhang, “Edge computing:state-of-the-art and future directions,” *Journal of Computer Research and Development*, vol. 56, no. 1, p. 21, 2019.

[7] W. Shi, C. Jie, Z. Quan, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *Internet of Things Journal, IEEE*, vol. 3, no. 5, pp. 637–646, 2016.

[8] W. Shi, H. Sun, J. Cao, Q. Zhang, and W. Liu, “Edge computing: a new computing model in the internet of everything era (in chinese),” *Journal of Computer Research and Development*, vol. 54, no. 5, p. 18, 2017.

[9] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, “A kubernetes controller for managing the availability of elastic microservice based stateful applications,” *Journal of Systems and Software*, no. 11, p. 110924, 2021.

[10] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *Cloud Computing, IEEE*, vol. 1, no. 3, pp. 81–84, 2014.

[11] S. Gupta, W. Zhang, and F. Wang, “Model accuracy and runtime tradeoff in distributed deep learning:a systematic study,” *Computer Science*, pp. 171–180, 2015.

[12] Y. Li, D. Han, and Z. Yan, “Long-term system load forecasting based on data-driven linear clustering method,” *Journal of Modern Power Systems and Clean Energy*, 2017.

[13] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, and B. Y. Su, “Scaling distributed machine learning with the parameter server,” *ACM*, 2014.

[14] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, “Parallelized stochastic gradient descent,” in *Advances in Neural Information Processing Systems 23: Conference on Neural Information Processing Systems A Meeting Held December*, 2011.

[15] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing, “Primitives for dynamic big model parallelism,” *Computer ence*, 2014.

[16] S. Lee, J. K. Kim, X. Zheng, Q. Ho, and E. P. Xing, “On model parallelization and scheduling strategies for distributed machine learning,” in *International Conference on Neural Information Processing Systems*, 2014, pp. 2834–2842.

[17] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” *ACM*, 2016.

[18] M. Qi, “Lightgbm: A highly efficient gradient boosting decision tree,” in *Neural Information Processing Systems*, 2017.