Evolution of the HEPS Jupyter-based remote data analysis System

Zhibin Liu[1,2*], Qiulan Huang[1*], Haolai Tian[1,3], Yu Hu[1], Jingyan Shi[1], Ran Du[1], Hao Hu[1], Lu Wang[1], Fazhi Qi[1]

[1]Institute of High Energy Physics, CAS, 100049 Beijing, China
[2]University of Chinese Academy of Sciences, 100049 Beijing, China
[3]Spallation Neutron Source Science Center, Dongguan 523803, China

Abstract. High Energy Photon Source(HEPS) Experiment is expected to produce large amount of data and have diverse computing requirements for data analysis. Generally, scientists need to spend several days to setup their experimental environment, which greatly reduce the scientists' work efficiency. In response to the above problems, we introduce a remote data analysis system for HEPS. The system provides users a web-based interactive interface based Jupyter, which makes scientists are able to process data analysis anytime and anywhere. Particularly, we discuss the system architecture as well as the key points of this system. A solution of managing and scheduling heterogeneous computing resources (CPU and GPU) is proposed, which adopts Kubernetes to achieve centralized heterogeneous resources management and resource expansion on demand. An improved Kubernetes resource scheduler is discussed, which dispatches upper applications to nodes combining with the computing cluster status. The system can transparently and quickly deploy the data analysis environment for users in seconds and reach the maximum resource utilization. We also introduce an automated deployment solution to improve the work efficiency of developers and help deploy multidisciplinary applications faster and automatically. A unified certification is illustrated to make sure the security of remote data access and data analysis. Finally, we will show the running status of the system.

1 Introduction

High Energy Photon Source (HEPS)[1] is the fourth-generation synchrotron radiation source with the highest spectral brightness in the world. High-throughput synchrotron radiation experiments with ultra-high spatial resolution, time resolution, and energy resolution will be carried out. It is estimated that the 15 beamlines of the first phase of HEPS will generate 200TB/day experimental data per day, and the peak value can reach 500TB per day[2]. In order to process such massive data in a timely manner, a large amount of heterogeneous computing resources such as CPU, GPU, FPGA are required to accelerate the calculation process. However, users from other fields, such as biology, chemistry, are

* Corresponding author: huangql@ihep.ac.cn, liuzhibin@ihep.ac.cn

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

troubled with the complexity of production environment deployment, short of operation and maintenance experience, difficulty to migrate the environment to other machines and so on. To address those issues, this paper proposed a web-based interactive data processing platform. Users can select application services through WEB browser, and make resource application and environment configuration through one-click action. The system optimizes the scheduling algorithm of computing resources. The entire scheduling process is transparent to users. Scientists only need to focus on scientific data analysis and do not need to care about application deployment and environment configuration, so as to improve the daily work efficiency. This paper is organized as follows: The rest of Section 1 introduces an overview of HEPS computing requirements and dataflow. Section 2 illustrates the system architecture and discusses the realization of key technical points. Section 3 shows the running status of this system. Section 4 summarizes the whole paper.

1.1 HEPS computing system requirements

After the completion of the HEPS, HEPS will have 15 beamlines in the first phase. Each beamline station has different experimental methods, different data analysis techniques, different types of computing resource requirements, and different volumes of data generated. Some experiments will produce extremely large data (such as diffraction tomography experiments, one experiment may produce 50TB of data), which have high demands for computing power. After communicating with the scientific research scientists of those beamline stations, we summarize the computing requirements of HEPS, which are shown in Table 1. It can be seen from Table 1 that HEPS scientific computing has the characteristics of diversity: (1) Various algorithms and software: self-development and commercial purchase; (2) Different Operating systems: Linux and Windows; (3) Different computing timeliness: task needs to be finished in seconds to hours; (4) Different resource types: CPU and GPU. These characteristics show that HEPS scientific computing is very different from High energy Physical(HEP) computing. HEP computing is high throughput batch computing mode. Generally, HEP computing tasks are batch jobs without timeliness requirements and the HEP software is relatively simple, running in Linux.

Table 1. HEPS computing system requirements.

	Requirements
Computing timeliness	Seconds level~Hours level
Computing resources	CPU, GPU
Software service	CT Tomopy, Spark, Deep learning, etc
Operating systems	Linux, Windows

1.2 HEPS dataflow

Fig. 1 shows the HEPS data flow. The beamline server gets data from the front-end detector and then records the valid data in the beamline persistent storage device. At the same time, a part of the data stream from the detector will also directly enter the fast-preview system

for quick review. The online system processes the data and saves the rapid data processing results to the central storage system. The offline system will read the data from the central storage system for offline data analysis. Users can only read the data in the central storage system. If the data is not used for a long time like 6 months, it will be transferred to tape archive system. Transfer Server will be responsible for the data movement from the central storage system to the tape library or remote site. Our remote data analysis system provides online data processing service as well as offline data processing service.

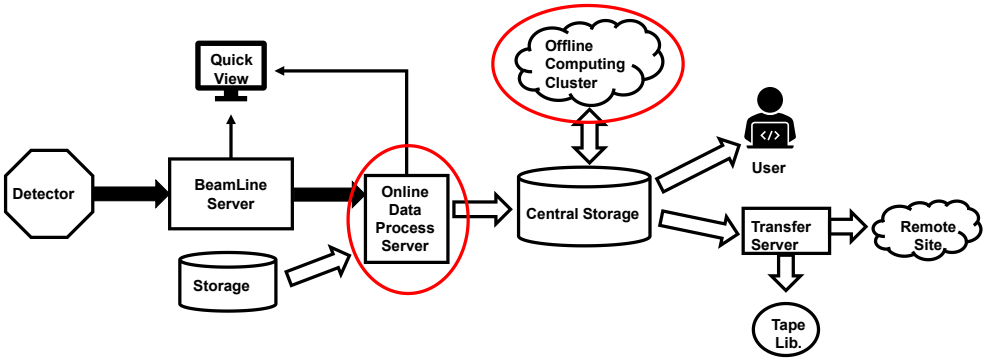

Fig. 1. HEPS dataflow.

2 System Architecture

This system aims to provide a web-based interactive data processing system. Users can log in with IHEPSSO (IHEP single sign on interface) through the Jupyterhub page[3, 4]. After the authentication is completed, they can select the required application environment and computing resources. The platform will select the appropriate node transparently and deploy the computing environment quickly for the user. The user can conduct interactive data analysis in the Jupyter environment. Application software developers only need to push the source code and configuration to the code repositories, which will trigger the automated build test and deploy the application software to the platform automatically.

The system design and implementation are based on containers and container orchestration technology. We use containerization technology to define applications to avoid repeated deployment of complicated environments through container construction. Docker[5] is a management tool for containers and images. Compared with virtual machines, it can use system resources more efficiently, it can start up quickly, less start latency, and it has a consistent operating environment for migration more easily. In order to manage and schedule resource for multi-user and multi-application efficiently, we use Kubernetes[6, 7] to orchestrate containers. Pods are the smallest deployable units of computing that you can create and manage in Kubernetes, we use a pod with specific software installed as an application. Our system supports the heterogeneous resources scheduling, such as CPU, GPU, and FPGA. We have optimized the Kubernetes default scheduler to deploy applications to appropriate nodes by developing a new scheduling framework to maximum resource utilization and to achieve rapid deployment in seconds. Our system also supports automated operation and maintenance deployment. Developers only need to submit a Dockerfile that contains all the commands that would be called by a user to assemble an image. Then the system will build, test and deploy to the production environment automatically. We have integrated IHEPSSO in the system to provide a safe and convenient way to access and isolate the system.

The system architecture is shown in **Fig. 2**. There are some key technologies (1) Provide visual resource application and operation interface; (2) Support the management and scheduling of heterogeneous resources. (3) Application developers can test and deploy applications automatically; (4) Realize unified authentication and login to ensure secure data access and reasonable resource usage.

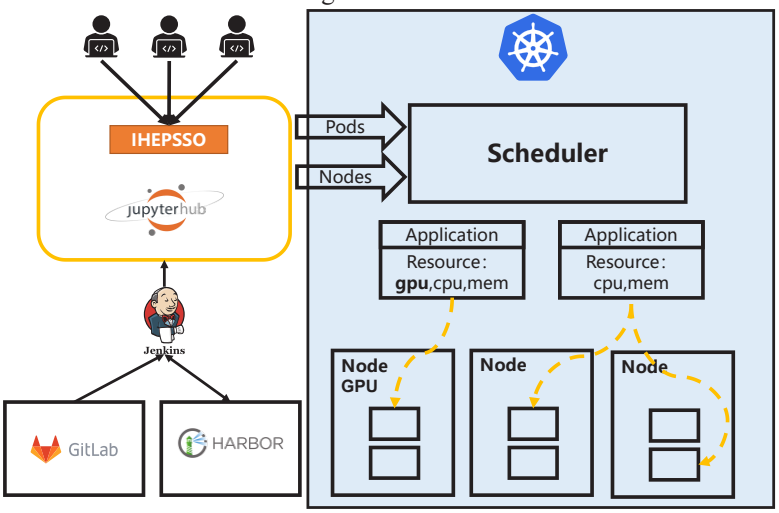

Fig. 2. System Architecture.

2.1 Heterogeneous Resources Management and scheduling

A principal consideration of heterogeneous resource management is to use heterogeneous resources as kubernetes native resources, which provides a logical resource pool to upper application transparently. As a container orchestration and scheduling engine, Kubernetes can use heterogeneous resources through declarative APIs. Kube-scheduler is the default scheduler for Kubernetes and runs as part of the control plane. Kube-scheduler provides some scheduling strategies which can be used to specify the predicates and priorities that the kube-scheduler runs to filter and score nodes. However, it still cannot meet the needs of different scenarios, such as considering network load and non-native heterogeneous resources. We optimized the Kubernetes default scheduler to support CPU, GPU. When the user selects an application, the system will automatically apply for computing resources for the application according to the user group and application type. The computing system sends requests to the scheduling system. The scheduling system selects the appropriate node considering several metrics which collected by shell script including computing resource(such as the number of free CPU, GPU), memory, storage capacity, iowait, network load, service quality, cluster resource utilization and so on. The entire scheduling process is transparent to users. With the use of docker deployment, the application will start in seconds.

The scheduling process is shown in the **Fig. 3**. Pod, the smallest scheduling unit in Kubernetes, contains user's request information, such as image, data, CPU, memory, and GPU. The Pod requested by the user will be placed in a priority queue for scheduling according to the Quality of Service for Pods shown in Table2. In the filtering stage, the filter first traverses the queue of storage nodes according to the needs of the Pod, select the nodes that meet the needs of the pod and put it in the candidate node queue, and ignore the nodes that do not meet the needs. Next, we will enter the scoring stage. We have implemented a custom scoring algorithm that combines with storage, network, and service

quality to score nodes. The node with the highest score will be bound to the Pod, and finally the Pod will be scheduled to this node.
(1) In the filtering stage, we will consider four points: storage, Pod and Node matching correlation, Pod and Pod matching correlation, and Pod fragmentation correlation. We will filter out the nodes in the following cases: a) the node has disk pressure; b) the node label and the Pod label doesn't match; c) there is anti-affinity between the Pod waiting to be scheduled and the Pod already running on node; e) the Pod doesn't spread between nodes.

(2) In the scoring stage, we will consider node resource usage, network load, and quality of service (Qos) for scheduling.

(3) The Node and Pod selected will be bound in the binding stage, and then the Pod will be scheduled to the node.

Table 2. Quality of Service for Pods

Type	Describe	Example
Guaranted	Pod must have a resource limit and request, and they must be the same	Memory request: 200Mi Memory limit: 200Mi
Burstable	Pod has a resource request, but does not meet the criteria for Guaranteed	Memory request: 200Mi
BestEffort	Pod must not have any memory or CPU limits or requests	-

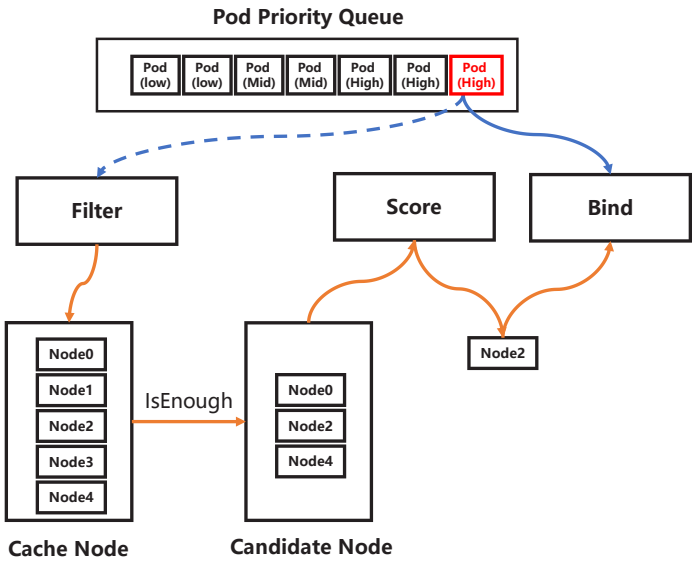

Fig. 3. Pod scheduling process

2.2 User interface

We use Jupyterhub to create a multi-user Hub which spawns, manages, and proxies multiple instances of one user's Jupyter notebook server. In order to achieve unified management of services, we deploy Jupyterhub in kubernetes as a Deployment that can maintain a specific number of replicas. Kubernetes Ingress is an API object that manages external access to the services in a cluster, typically HTTP, we use Ingress to expose Jupyterhub's services that can be accessed by external traffic from the cluster, at the same time, it also achieves load balancing of multiple replicas of traffic. When the user sends a request to create an application, Jupyterhub will spawn a pod, and the Ingress will add an HTTP route mapped to the pod based on the user name. The user's subsequent access can be carried out through this HTTP route. Jupyterlab allows users to write code in a notebook document, explore and analyze data, plot data, and share results interactively. We installed Jupyterlab in the basic image, and used the Jupyterlab commands as the startup entry, so that we can ensure that each Pod can use Jupyter as the access entry when it starts. We also add annotations to Ingress to support websocket traffic to support interaction with the terminal in the pod through the web.

2.3 Automated deployment

Developers build a new application through Dockerfile, and then push the code to repository. After the code is reviewed and approved by the administrator, it will be merged into the main code repository. After the merge operation is completed, an automatic test will be performed, and then a new image will be generated, and the image will be pushed to the image repository. The agent software Jenkins[8] will check for updates to the image repository, pull new images from the repository, and update the configuration file. The Kubernetes cluster will detect that the current state is inconsistent with the specification of the configuration file, and then deploy a new image.

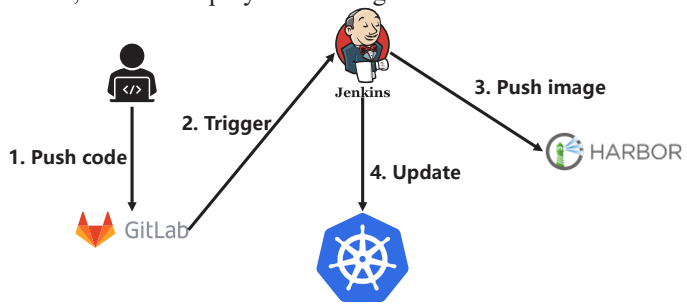

Fig. 4. Developers automate deployment through Jenkins.

2.4 Unified certification

IHEP Single Sign On service (IHEPSSO) is an authentication system based on Oauth2.0[9], and users can access WEB-based information service as well as computing system after authentication. We developed a plugin based IHEPSSO for Jupyterhub to support multi-users. In order to access IHEP resources such as access software data stored in CernVM-FS[10], experimental data and personal data stored in the Lustre[11] parallel file system, we need to load the corresponding directories in the container depending on the user's attributions like user ID or group ID. Fig 5 shows the workflow of user authentication. For security, we do not store user information in the image, so when the container is initialized, a new user will be created in the container based on the user information returned by IHEPSSO, so that the logged-in user has permission to access IHEP Resource. The system

also defines and grants different access permissions for different user/groups to achieve secure data access and reasonable resource usage.

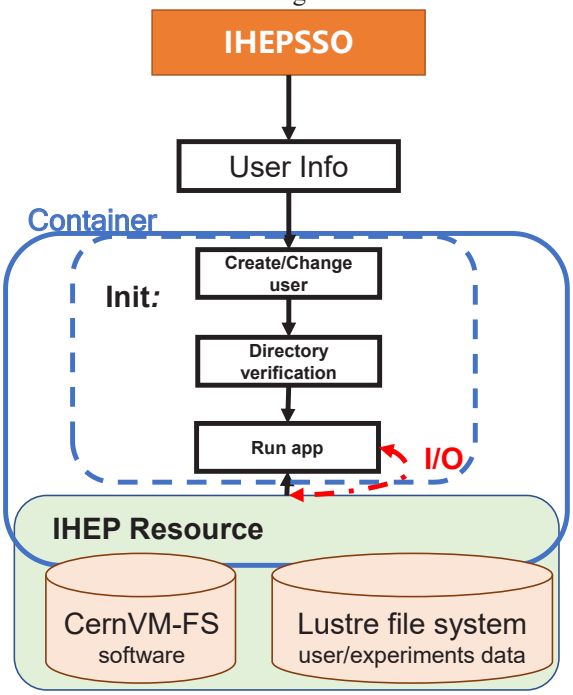

Fig. 5. User authentication based on IHEPSSO.

3 Running Status

The computing system for HEPS aims to provide an effective environment for a wide range of scientific computing applications. As shown in Fig.6, we currently provide four application services such as, CT 3D reconstruction service, which is used for tomographic data processing and image reconstruction; Deep learning service, which integrates multiple machine learning frameworks and dependent libraries; Spark driver service, which can accept spark jobs submitted by users, and will spawn multiple executors to run the user jobs, and finally return the results to the user. In order to meet diverse requirements, we also provide customization functions to users, and new images will be updated and released through our automated deployment workflow. The user chooses the service in a friendly portal. Once the service is chosen, the system will generate the service environment automatically. This process is forwarded remotely and is not restricted by the local environment. As shown in Fig.7, user chooses CT 3D reconstruction service which is based on Tomopy ang Jupyter Widget to visualize the reconstruction results.

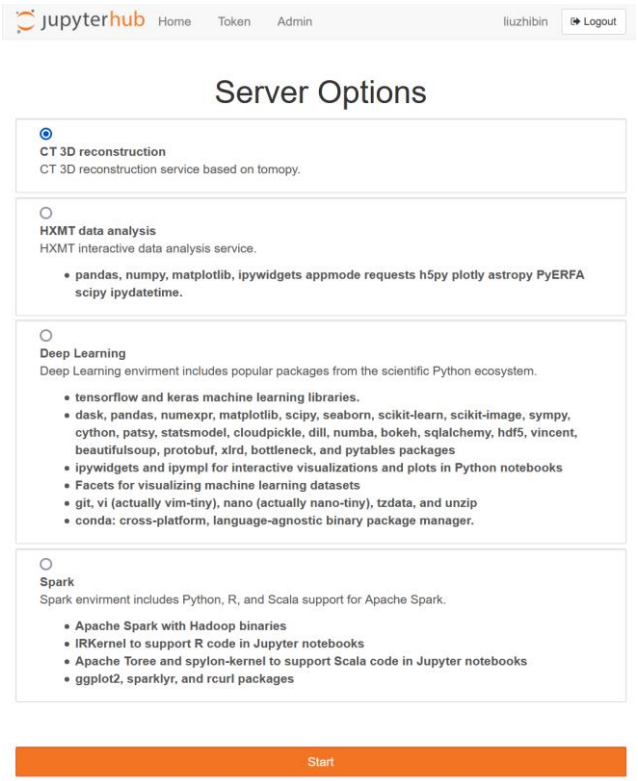

Fig. 6. User service selection interface

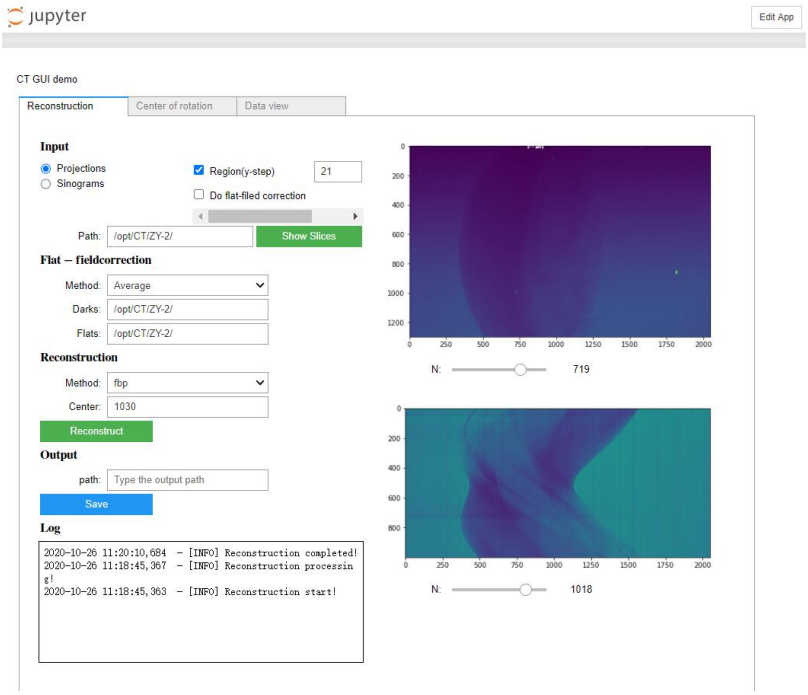

Fig. 7. Example of 3D reconstruction application.

4 Conclusion

This paper we presented the design and implement of a remote re system. We have optimized the default scheduler of Kubernetes, which can quickly deploy applications and improve cluster resource utilization. We also have provided WEB-based interactive data analysis service and developed IHEPSSO authentication plugin to expand Jupyterhub authentication module to ensure security certification and reasonable resource usage. An automated deployment solution is proposed, which can speed up application deployment and reduce the tasks of developers, so as to improve the work efficiency significantly. We have now integrated 4 applications, but they are still in the testbed. From the actual test results of CT 3D reconstruction service, we can see that in the case of a single node, the results can be calculated within a few minutes by processing about 30GB of data. In the next step, in order to better achieve better fast vision results, we need to start with software and computing resources. We will also provide a distributed computing cluster based on spark. The system can also easily support other applications.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (NSFC) under Contracts No. 11875283.

References

[1] Jiao Y, Xu G, Cui X H, et al. The HEPS project[J]. Journal of synchrotron radiation, 2018, 25(6): 1611-1618.

[2] Qi Fazhi, Huang Qiulan, Hu Hao, Tian Haolai, Wang Lu, Wang Yanming, Zhao Haifeng, Zhang Hongmei, Zeng Shan. The Design of Science Data Platform for High Energy Photon Source[J]. Frontiers of Data and Computing, 2020, 2(2): 40-58.

[3] Jupyterhub: https://jupyter.org/hub

[4] Perez F, Granger B E. Project Jupyter: Computational narratives as the engine of collaborative data science[J]. Retrieved September, 2015, 11(207): 108.

[5] C. Anderson, "Docker [Software engineering]," in *IEEE Software*, vol. 32, no. 3, pp. 102-c3, May-June 2015, doi: 10.1109/MS.2015.62.

[6] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. Commun. ACM 59, 5 (May 2016), 50–57. DOI:https://doi.org/10.1145/2890784

[7] Kubernetes: https://kubernetes.io/

[8] Jekins: https://www.jenkins.io/

[9] B. Leiba, "OAuth Web Authorization Protocol," in *IEEE Internet Computing*, vol. 16, no. 1, pp. 74-77, Jan.-Feb. 2012, doi: 10.1109/MIC.2012.11.

[10] Blomer, Jakob, Predrag Buncic, and Thomas Fuhrmann. "CernVM-FS: delivering scientific software to globally distributed computing resources." Proceedings of the first international workshop on Network-aware data management. 2011.

[11] Braam P. The Lustre storage architecture[J]. arXiv preprint arXiv:1903.01955, 2019.