OpenStack-based Automated Management of Compute Nodes in Astronomical Observatories: A Postprint

Authors: Shao Cen, Deng Hui, Wang Feng, Wei Shoulin, Mei Ying, Shi Congming, Liang Bo, Dai Wei, Liu Cuiyin

Date: 2017-09-26T00:00:00+00:00

Abstract

Astronomical data processing constitutes a crucial component of astronomical research. With the rapid advancement in capabilities and observational capacities of new-generation telescopes, establishing high-performance real-time computing platforms at observational sites for rapid data analysis and processing has emerged as a prevailing trend. Addressing the construction requirements for the real-time data processing system of the Mingantu Radio Spectral Heliograph and Mingantu Observatory, this study systematically investigates implementation methods for OpenStack-based local cloud and automatic system management modes, proposes a dynamic approach for starting and stopping compute nodes, and conducts practical testing. Experimental results demonstrate that this paradigm fully satisfies the demands of astronomical data processing and is more efficient than conventional data processing methods that statically allocate computing resources. It can effectively reduce energy expenditure and lower observational costs, providing valuable reference for the future construction of high-performance computing platforms at astronomical observatories.

Full Text

Research on Automatic Management of Compute Nodes at Astronomical Observatories Based on OpenStack

Shao Cen[1], Deng Hui[1], Shi Congming[1], Wang Feng[1], Liang Bo[1], Wei Shoulin[12], Mei Ying[2], Dai Wei[2], Liu Cuiyin[2]

[1]Key Laboratory of Applications of Computer Technologies of the Yunnan Province, University of Science and Technology of Kunming, Kunming 650500, China, Email: dh@cnlab.net
[2]Yunnan Observatories, Chinese Academy of Sciences, Kunming 650011, China

Abstract

Astronomical data processing is a critical component of astronomical research. With the rapid advancement in capabilities of new-generation telescopes, constructing high-performance real-time computing platforms at observational sites to enable rapid data analysis and processing has become an emerging trend. Addressing the construction requirements for the real-time data processing system of the Mingantu Ultrawide Spectral Radioheliograph (MUSER) and Mingantu Observatory, this paper systematically investigates implementation methods for local cloud infrastructure based on OpenStack and automated system management patterns. We propose a dynamic approach for starting and stopping compute nodes and conduct practical testing. Experimental results demonstrate that this model fully satisfies the requirements of astronomical data processing, offering greater efficiency compared to traditional static resource allocation methods. It can effectively reduce energy consumption and lower observational costs, providing valuable reference for future construction of high-performance computing platforms at astronomical observatories.

Keywords: Astronomical data processing; OpenStack; Automatic management

Introduction

As astronomical data volumes expand rapidly, traditional computing methods can no longer meet processing demands. Distributed computing and virtualization technologies have become research hotspots in astronomical data processing due to their low cost and high efficiency. Real-time processing of large observational datasets during astronomical observations and subsequent data reduction requires stable and efficient infrastructure support, which directly impacts processing efficiency [1]. This places extremely high demands on underlying computing resources. The Mingantu Ultrawide Spectral Radioheliograph (MUSER), located in Zhengxiangbai Banner, Xilingol League, Inner Mongolia, is a new-generation solar radio interferometer in China [2]. A high-performance computing platform has been deployed at Mingantu Observatory to ensure real-time observations. However, this raises issues where on-site observation assistants and resident scientists may not be familiar with computer system maintenance, making it difficult to quickly recover from operating system or software failures [3]. Keeping all compute nodes powered on continuously incurs substantial costs in annual electricity fees when there is limited computational demand. Introducing cloud computing into astronomical data processing can simplify overall system management requirements, allow some compute nodes to enter low-power offline states, and improve system maintainability. Static virtual machine allocation patterns often result in low resource utilization or waste, while manual intervention allocation modes increase system costs [4]. To address issues of insufficient computing resources, resource waste, and low utilization in traditional astronomical data processing models, this paper proposes an automatic management pattern based on OpenStack. This pattern adopts

OpenStack virtualization management software as the foundational platform, providing virtual computing resources for astronomical data processing with dynamic management. It can increase or decrease compute resources based on current system workload, shutting down nodes with lighter computational loads to improve resource utilization and reduce physical machine overhead, thereby addressing problems in traditional astronomical data processing.

1. OpenStack Cloud Platform Introduction

OpenStack is an open-source cloud platform management software jointly developed by NASA and the cloud computing provider Rackspace. Since its first release, thousands of community members and companies have participated in version upgrades and improvements. OpenStack can provide a complete set of infrastructure services (IaaS), platform services (PaaS), and software services (SaaS) to various fields. Its three core components are Nova, Glance, and Swift. Nova handles virtual machine management, including creation and deletion. Nova itself contains many sub-services: Nova-API provides service interfaces and receives user requests; Nova-Scheduler handles virtual machine scheduling, deciding on which physical node to create a virtual machine; and Nova-Compute performs virtual machine creation [6]. Glance is the image management component, responsible for image upload, deletion, and other operations, storing images in backend storage devices or using local file systems. Swift provides object storage services. Users can create containers in Swift to store various types of files. Swift' s object storage service offers strong scalability, redundancy, and persistence, providing extremely high security for users through multiple replica storage [7]. In addition to these three core components, OpenStack also provides authentication services (Keystone), virtual network services (Quantum), virtual volume services (Cinder), and a dashboard (Horizon). The relationships among components are shown in Figure 1 [Figure 1: see original paper]. All OpenStack components can be used together to provide complete cloud services, or each component can be installed separately as an independent sub-service for extensive secondary development. This demonstrates that OpenStack is feature-rich, fully meets the requirements of large-scale astronomical data processing, offers good scalability, and can be customized according to needs.

2. OpenStack Cloud Platform Deployment

The deployment scheme for MUSER consists of one control node and N compute nodes. All nodes run Ubuntu 14.04 Server. The control node manages global services, including virtual network services, client network services, and database services using MySQL. The control node only provides proxy services, while compute nodes perform actual object storage and virtual machine creation. On the control node, we deploy Keystone, Quantum-Server, Horizon, Glance, and Nova-API. Compute nodes deploy Nova-Compute, Quantum-agent, Swift-Storage, and Swift-Proxy, as shown in Figure 2 [Figure 2: see original paper]. Both control and compute nodes are equipped with two network cards.

Quantum provides virtual network services, first adding all virtual machines to an internal LAN to ensure inter-VM communication. For management convenience, Open vSwitch is used to create VLANs for virtual machines. The internal LAN is then connected to the external network via a router, allowing each VM to access external resources. The external network here refers to the network established by OpenStack that shares the same subnet as the network card. Virtual machine clusters responsible for different functional modules can be isolated through subnet partitioning. OpenStack provides each VM with a floating IP for direct external access. The client interface is provided by Horizon on the control node. This deployment scheme offers high scalability, making it very convenient to add compute nodes by simply deploying Nova-Compute, Quantum-agent, and Swift-Storage on the new node. Users can access the control node' s external network directly through a browser.

Automatic Management Mode

To achieve automatic management of computing resources, we monitor CPU usage, memory usage, and network traffic for each VM in the cluster. When a VM' s monitored values remain below set thresholds for a period, indicating light computational load, the VM is shut down to reduce system overhead. When monitored values across the cluster exceed thresholds for a period, indicating heavy computational load, a new VM is started to balance the load. When a particular VM' s load becomes too high, additional resources are allocated promptly to meet computational demands. Implementing this automatic management mode requires solving two key problems: first, how to obtain resource consumption data from each compute node, and second, how to perform start/stop operations on compute nodes.

Through systematic analysis of OpenStack' s client code, we found that while it provides callable APIs, the functionality is relatively dispersed, and some methods require overly complex parameters with verbose formats. Therefore, we re-encapsulated these methods to form a streamlined API set convenient for implementing automatic management. The re-encapsulated APIs include user management, virtual network management, and VM management interfaces. Key methods include: **create_user**, which creates a user in OpenStack requiring parameters such as username, password, and tenant (a project in OpenStack that can contain many users). This method first checks if the tenant exists, creates it if not, then creates the user under the tenant while checking uniqueness and adding relevant permissions. **create_img** creates an image under a specified user, requiring parameters like username, image format, and image location (either a local path or network resource link). **create_net** creates a network under a specified user, requiring parameters including username, network name, network address pool, DNS, and DHCP support. **create_vm** creates a virtual machine under a specified user, requiring parameters such as username, CPU count, memory size, and image name, with optional parameters to specify network and other configurations. All these methods verify user legitimacy and

resource uniqueness before creation. Image preparation is critical when deploying the cloud environment. By packaging the operating system and production environment into an image, we can rapidly deploy an astronomical data processing cloud environment through these APIs. When the production environment changes, snapshots can be used to create new VMs [9].

There are many methods for monitoring VMs in a cluster. Libvirt is a C function library for Linux that supports management of multiple virtualization technologies including KVM, Xen, and QEMU [10]. While Libvirt commands do not directly provide VM CPU usage, memory usage, or network traffic, we can calculate resource utilization over time from the information they provide. The main commands used are: **virsh vcpuinfo**, **virsh dommemstat**, and **virsh dumpxml**. These commands return VM CPU information, memory information, and boot parameters respectively. To calculate CPU usage, we periodically extract CPU time parameters from virsh vcpuinfo output, then divide the difference in CPU time between two timestamps by the interval duration. For memory usage, we divide the time period into N segments and extract instantaneous memory usage mi N times within the period. The instantaneous memory usage rate at each moment is calculated by comparing mi with the VM' s total memory M. The average memory usage rate for the period is then computed. For network traffic, virsh dumpxml provides VM boot information from which we extract the network device name, then use ifconfig to obtain traffic reports for that device. Similarly, we calculate traffic over a time interval. We encapsulate the resource monitoring component as a background service running on each compute node, which periodically sends data to the control node. Upon receiving data, the control node compares each metric against preset safety ranges. If a VM' s average load exceeds the upper safety limit, the system uses the **create_vm** method to start a new VM and add it to the compute cluster. If a VM' s load falls below the lower safety limit, the **delete_vm** method is used to shut it down and remove it from the cluster. The complete processing flow is shown in Figure 4 [Figure 4: see original paper].

Experiments and Analysis

To verify overall system availability, we deployed OpenStack using two blade servers—one as the control node and one as the compute node. Using our encapsulated APIs, we automatically deployed an astronomical observation data processing environment, including tenant and user creation, image upload, network creation, and VM creation. For networking, we created an internal network and an external network. All VMs connect to the internal network for management convenience and to ensure inter-VM communication. The external network provides access to external resources and assigns floating IPs for direct external VM access. A router connects the internal and external networks.

The experimental environment is shown in Table 1 . Based on actual computational tasks, the safety range for the system was set as follows: CPU usage 0.4-0.8, memory usage 0.4-1 (GB). The monitoring service on compute nodes

sends data to the control node every hour. After receiving data, the control node compares it against the safety range to decide whether to start or stop VMs.

At the start of the experiment, the cluster was assigned a small computational task. As we gradually increased the computational load on the cluster, we observed that new VMs were automatically added to the compute cluster (slave4). The blade servers used in the experiment feature voltage monitoring, allowing us to obtain power consumption measurements for two scenarios as shown in Figures 5 [Figure 5: see original paper] and 6 [Figure 6: see original paper].

The automatic management model proposed in this paper automatically adds VMs when the compute cluster has many tasks and shuts down idle VMs when tasks are few, effectively improving resource utilization and reducing unnecessary overhead. Comparing power consumption between the two scenarios: after one hour of operation with 2 VMs, power consumption was 2400-2500W; with 7 VMs, it was 2700-2800W. The difference is approximately 0.3 kWh per hour. Shutting down some VMs during low computational load can effectively reduce power consumption and system costs, which has significant reference value for MUSER and any astronomical data processing project. The power consumption comparison is shown in Figure 7 [Figure 7: see original paper].

Conclusion

This paper studied the application of cloud computing technology in astronomical data processing, examining the basic architecture and deployment of the OpenStack open-source cloud platform. Using MUSER as an example, we proposed an automatic management model based on OpenStack that dynamically allocates computing resources according to current system workload. Compared with traditional astronomical data processing models, our approach effectively improves resource utilization and reduces system overhead. We tested the feasibility of the OpenStack-based automatic management model in astronomical data processing, which has practical significance. Future work will focus on developing more highly automated tools, considering automatic addition and deletion of physical nodes to provide faster and more efficient data processing services for astronomical research.

References

[1] 颜毅华, 张坚, 陈志军, 等. 关于太阳厘米-分米波段频谱日像仪研究进展 [J]. 天文研究与技术——国家天文台台刊, 2016, 13(1): 91-98.
Yan Yihua, Zhang Jian, Chen Zhijun, et al. Progress on Chinese solar radioheliograph in cm-dm wavebands[J]. Astronomical Research & Technology, 2016, 13(1): 91-98.

[2] 邓辉, 王锋, 梅盈, 等. 基于 QT 的 MUSER 观测数据多屏图形化实时显示系统的设计与实现 [J]. 天文研究与技术, 2016, 13(2): 243-250.

Deng Hui, Wang Feng, Mei Ying, et al. Design and implementation of a multi-monitor display system based on the QT for MUSER observations[J]. Astronomical Research & Technology, 2016, 13(2): 243-250.

[3] 周鑫磊, 王威, 王锋, 等. 气象站实时数据采集在 MUSER 中的应用研究 [J]. 天文研究与技术, 2016, 13(3): 503-509.
Zhou Xinlei, Wang Wei, Wang Feng, et al. A study on the real-time collection of the Vantage Pro weather station and the application in the MUSER[J]. Astronomical Research & Technology, 2016, 13(3): 503-509.

[4] Wang F, Mei Y, Deng H, et al. Distributed data-processing pipeline for Mingantu Ultrawide Spectral Radioheliograph[J]. Publications of the Astronomical Society of the Pacific, 2016, 128(969): 115001.

[5] Rosado T, Bernardino J. An overview of OpenStack architecture[C]//International Database Engineering & Applications Symposium. ACM, 2014: 366-367.

[6] Beernaert L, Matos M, Vilaca R, et al. Automatic elasticity in OpenStack[C]//Proceedings of the 18th Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management. ACM, 2014: 1-6.

[7] Hu B, Yu H. Research of scheduling strategy on OpenStack[C]//Proceedings of the 2013 International Conference on Cloud Computing and Big Data. IEEE, 2013: 81-90.

[8] Zhang Y, Krishnan R, Sandhu R. Secure information and resource sharing in cloud infrastructure as a service[C]//Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security. ACM, 2014: 191-196.

[9] 梁宇, 杨海波, 李鸿彬, 等. 基于 OpenStack 的资源监控系统 [J]. 计算机系统应用, 2015, 24(4): 44-47.
Liang Yu, Yang Haibo, Li Hongbin, et al. Resource monitoring system based on OpenStack[J]. Computer Systems & Applications, 2015, 24(4): 44-47.

[10] Libvirt: The virtualization API[EB/OL]. https://libvirt.org/, 2016.

Note: Figure translations are in progress. See original paper for figures.