Postprint: Automatic Deployment Method for Astronomical Application Software Based on Container Technology

Authors: Yao Kun, Dai Wei, Yang Qiuping, Mei Ying, Shi Congming, Wang Feng

Date: 2018-12-03T00:00:00+00:00

Abstract

The Square Kilometre Array (SKA) telescope is about to commence construction, with various sub-work packages entering the critical design review phase. Cloud and container technologies represent potential future platform technologies for the SKA Science Data Processor (SDP). This paper addresses the requirements for rapid deployment, execution, and testing of astronomical application software in the context of SDP' s ultra-large-scale massive data processing, fully considering challenges such as the complex runtime environment of astronomical application software and the difficulties of deploying ultra-large-scale computing clusters in cloud computing environments. The paper systematically investigates and proposes a general automatic deployment method for astronomical application software utilizing Docker technology. Taking the widely-used visibility function calibration software SAGECaL as a case study, the paper first analyzes its relevant characteristics and the difficulties associated with its distributed deployment, subsequently presenting an automatic deployment method for SAGECaL distributed clusters based on Docker container technology. Experimental results demonstrate that the proposed automatic deployment method substantially enhances the deployment efficiency of SAGECaL distributed clusters, satisfying the requirements for basic platform deployment and switching necessary for the project team' s SKA-SDP related testing work. Furthermore, this work provides valuable insights for the rapid deployment and execution of other astronomical software in cloud environments.

Full Text

Preamble

Automatic Deployment Method for Astronomical Application Software Based on Container Technology

Yao Kun[1], Dai Wei[1]*, Yang Qiuping[1], Mei Ying[23], Shi Congming[1], Wang Feng[123]
[1]Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China
[2]Yunnan Observatories, Chinese Academy of Sciences, Kunming 650011, China
[3]Center for Astrophysics, Guangzhou University, Guangzhou 510006, China

Abstract

The Square Kilometer Array (SKA) telescope is entering the construction phase, with each sub-work package moving into critical design evaluation stages. Cloud and container technologies represent promising platform technologies for the SKA Scientific Data Processor (SDP). This paper addresses the challenges of rapid deployment, execution, and testing of astronomical application software required for SDP' s ultra-large-scale data processing. Considering the complex runtime environments of astronomical software and the difficulties of deploying ultra-large-scale computing clusters in cloud environments, we systematically investigate and present a general automatic deployment method for astronomical applications using Docker technology. Taking the widely-used visibility function calibration software SAGECaL as an example, we first analyze its characteristics and distributed deployment challenges, then propose an automatic deployment method for SAGECaL distributed clusters based on Docker container technology. Experimental results demonstrate that our proposed method significantly improves deployment efficiency for SAGECaL distributed clusters, meeting the requirements for basic platform deployment and switching in SKA-SDP related testing work. Additionally, this work provides valuable insights for the rapid cloud deployment and execution of other astronomical software.

Keywords: Automatic deployment; Container technology; SAGECaL; SKA; SDP

0 Introduction

The Square Kilometer Array (SKA) is the world' s largest radio telescope currently under construction, with the Scientific Data Processor (SDP) representing one of its critical work packages. To address challenges in ultra-large-scale data processing for SDP, researchers have conducted extensive studies across various domains in recent years, including hardware accelerators (GPUs), computing frameworks [1], and system architectures [2], achieving significant results [3].

Among various platform technology options, cloud computing has emerged as

a viable choice for SKA astronomical data processing due to its cost savings, dynamic resource integration, scalability, simplified management and maintenance, and disaster recovery capabilities. Unlike typical internet-based application deployment, astronomical software requires numerous third-party libraries during deployment and involves numerous runtime parameters to meet different requirements. Current usage of container technology in astronomy remains at the level of application packaging. Particularly when astronomical software is managed by dedicated personnel, users face delivery delays while waiting for system administrators to deploy specific software [4]. Further experimentation and research are urgently needed to achieve rapid deployment in cloud environments, significantly reduce usage costs, and improve overall system availability. Meanwhile, using Docker [5] with container orchestration tools such as OpenStack [6], Mesos, and Kubernetes to package complex astronomical applications for faster deployment is becoming increasingly relevant [7].

SAGECaL (Space Alternating Generalized Expectation Maximization Calibration) [8] is a fast, memory-efficient radio interferometric calibration program supporting point source, Gaussian, and Shapelet models. SAGECaL supports CASA' s MS (Measurement Set) data and implements adaptive updates of ADMM (Alternating Direction Method of Multipliers) parameters in its distributed applications [9]. SAGECaL employs the Expectation-Maximization (EM) algorithm to obtain maximum likelihood estimates for observation instrumentation and sky model parameters. While standard EM algorithms estimate parameters for nonlinear superimposed signals, SAGECaL uses the SAGE algorithm—an improved EM variant—to achieve faster convergence, reduced computational complexity compared to least-squares estimation, and improved calibration quality over peeling methods. To meet the massive data processing requirements of next-generation radio interferometers like SKA, SAGECaL supports distributed parallel calibration across frequencies and can be deployed in single-CPU, GPU-accelerated, or MPI parallel computing environments.

During the SKA-SDP design phase, to provide credible computational resource requirement analysis and energy consumption design, our project team and international collaborators plan to deploy and run SAGECaL in different environments to analyze system overhead under various configurations, node counts, parameters, and output precision levels, ultimately optimizing SDP data processing design based on energy and computational costs. Clearly, traditional manual installation and environment configuration methods cannot meet these requirements. This paper presents an in-depth study on rapid SAGECaL deployment in cloud environments, yielding effective solutions that advance SDP-related work.

1.1 Basic Requirements Analysis

To efficiently provide test data, we focused on rapidly deploying a SAGECaL/MPI-based distributed cluster (hereinafter "the cluster"). Key deployment requirements include: editing cluster node lists according to node

configurations, ensuring cluster nodes reside on the same network segment, configuring NFS, setting up SSH passwordless login across nodes, configuring the MPI cluster environment, and ensuring consistent compilation versions of CASACORE, SAGECaL, and OpenMPI across all nodes.

Practical deployment faces several challenges: (1) Diverse Linux distribution preferences among users require specialized SAGECaL compilation and deployment settings; (2) SAGECaL compilation requires manual resolution of component dependencies, particularly for SAGECaL-MPI and SAGECaL-GPU, which need manual configuration file editing based on specific circumstances. This significantly increases deployment difficulty for non-experts.

To address these requirements, we propose using configuration scripts to automate each deployment step. The challenges can be resolved by containerizing SAGECaL with Docker. Combining this with a user-friendly interface further reduces usage complexity.

1.2 Overall Deployment Approach

Cluster automatic deployment involves constructing a SAGECaL Docker image (hereinafter "the image") and launching multiple SAGECaL containers (hereinafter "containers") through deployment services, with cross-container communication achieved via Overlay networks. The image encapsulates not only SAGECaL software but also automatic deployment scripts that generate different configuration behaviors based on node roles. Docker Swarm (hereinafter "Swarm") is Docker's native cluster management tool that establishes container clouds across multiple hosts and manages various resources through a unified interface [10].

[Figure 1: see original paper] Diagram of automatic deployment

Users first initialize available machines by creating a Swarm-managed container cloud, configuring NFS, and establishing a private image registry. Build and destroy services handle image construction and cluster teardown. The deployment service orchestrates these functions, implementing service orchestration, elastic scaling, cluster node discovery and maintenance, and user interfaces. After cluster computation completes, users can retrieve results from designated NFS directories.

1.3 Image Construction and Composition

The image is built upon a mainstream Linux distribution base image. This paper uses Debian 8.6 as the base image, constructed via Dockerfile. Primary software components include SSH, CASACORE, Glib, OpenBLAS, GCC, Make, OpenMPI, and SAGECaL.

[Figure 2: see original paper] Diagram of automatic deployment script encapsulation

As a universal image, containers must automatically perform different operations based on their cluster roles upon startup. During image creation, scripts defining various operations and automatic cluster list maintenance are packaged into the image [11]. To enable passwordless SSH login between cluster nodes and between bare-metal operating systems and the cluster master node, the bare-metal OS public key is also written into the image with SSH login prompt configurations.

1.4.1 Initialization

First, available physical machines are initialized. One physical machine is designated as the master node of the physical machine cluster, with others serving as worker nodes. Manual configuration establishes SSH passwordless login and startup automation scripts on each physical machine' s OS, causing them to actively initiate and maintain SSH connections to the master node upon boot. This enables the master node to detect and monitor all physical machines while maintaining the physical machine cluster list (the principle is identical to Section 1.4.3).

Second, NFS is configured for data sharing among containers. In this work, all machines mount their NFS shared directories into containers, enabling direct data exchange between bare-metal operating systems and containers.

Third, a private image registry is created with load balancing for image distribution. As shown in Figure 1, the private registry employs horizontal scaling across multiple physical machines for fault tolerance. After the build service constructs images, they are simultaneously pushed to all private registries. Nginx load balancing provides a unified image pull address, allowing the automatic deployment service to download required images and launch containers.

[Figure 3: see original paper] Block diagram of container cloud creation

Finally, the container cloud is automatically deployed. The container cloud deployment module encapsulates scripts for executing commands directly on the physical cluster master node, batch command execution based on IP lists, and physical cluster list comparison monitoring. As shown in Figures 1 and 3, users invoke the container cloud deployment module through the user interface. The deployment process proceeds as follows:

1. The container cloud deployment module initializes the Swarm master node via encapsulated commands, writing the obtained join command to a temporary file for later use.
2. The module reads IP addresses sequentially from the second line of the physical machine cluster list and uses batch execution scripts to remotely execute the Swarm join command on target physical machines.
3. Physical cluster list comparison monitoring continuously maintains the Swarm cluster IP list through encapsulated Swarm node inspection commands, comparing it with the physical machine cluster list. When both

lists match in row count and content, and Swarm node reachability shows Active status, all physical machines are considered joined to the Swarm cluster. The system then launches the initial interface and awaits user input for cluster deployment.

4. When new physical machines boot, the physical cluster list updates automatically, triggering the container cloud deployment module to execute the previously stored join command for automatic Swarm cluster integration.

Continuous monitoring through physical cluster list comparison enables automatic maintenance of container clouds with dynamically changing physical machine counts and provides fault tolerance for automatic deployment. Subsequent cluster deployments are implemented on this container cloud.

1.4.2 Overlay Network and Cluster Node Creation

Before launching cluster nodes, an Overlay network must be prepared for cross-host communication among node containers on a private logical network. During automatic deployment, Overlay network creation is achieved by encapsulating network configuration commands and specifying network names.

The deployment also encapsulates Swarm services' declarative model commands to define desired service states, relying on Docker to maintain these states and orchestrate related services. In this work, cluster nodes are orchestrated by master services for master nodes and worker services for worker nodes. Specific commands are as follows:

Based on different roles, each service initially launches one container, but worker node counts elastically scale according to worker service configuration. Both service groups bind to the same Overlay network and consistent NFS directories. The `mpi_{bootstrap}` file defines a set of operational commands that determine container actions upon startup based on the `role` parameter. The master node additionally receives the `host_{number}` parameter as the preset cluster scale. While Swarm services can set container quantities, cluster operation requires automatic cluster list maintenance within containers to ensure the cluster reaches its preset scale before executing user-inputted computation commands.

1.4.3 Cluster Node Discovery and Maintenance

Cluster node discovery is implemented through active IP exposure. The master node starts first and continuously monitors ARP (Address Resolution Protocol) cache. Each worker node initiates an SSH connection to the master node upon startup and accesses the null file in the master node' s dev directory. By maintaining this connection to the null file, the master node detects IP changes of worker nodes through ARP cache, achieving dynamic cluster list maintenance. The SSH connection command is as follows:

Since the cluster scale is preset, a variable is needed to notify the cluster maintenance module when the cluster reaches its target size. This paper uses the `CLUSTER_{{{GET}}_{{READY}}}` variable, where 0 indicates the cluster is not ready and 1 indicates readiness. When the master node detects through continuous monitoring that the cluster has reached its target scale, the variable is set to 1, instructing the master node to stop monitoring the cluster list and launch the initial interface shown in [Figure 5: see original paper] to await user computation commands.

1.4.4 User Interface

[Figure 4: see original paper] Initial interface

Cluster automatic deployment is initiated by the command shown in Example 2 of Figure 4. Users set the cluster scale according to actual needs. Upon completion, the interface remains at the initial screen awaiting user input for cluster computation commands or other operations. By providing a user-friendly interactive interface, users can complete various operations—including cluster computation, cluster termination and deletion, and node status inspection—by providing only minimal necessary parameters.

1.4.5 Fault Tolerance

Automatic cluster deployment may encounter errors due to insufficient hardware resources, master node startup failures or crashes, and network interruptions. If the cluster fails to become ready within a reasonable time, the automatic deployment module performs rollback to release resources. If rollback fails, users can forcibly destroy created services and release all resources by running the command shown in Example 5 of Figure 4. After verifying resource allocation parameters and network reachability, users can re-execute the cluster deployment command for automatic redeployment. Overall, when fault tolerance mechanisms fail at any stage, the default strategy is rollback and resource release.

2 Experimental Design

The previous sections present the automatic deployment method using SAGE-CaL as an example. This chapter tests and compares cluster deployment and computation timeliness across different server quantities and cluster scales, and evaluates the rationality of distributed deployment for cluster computing.

All experimental servers are Sugon TianKuo 620R models running Debian 9.4 stable as the physical machine OS, with Docker 18.03.1-ce and NFS components installed. The base service node runs Docker, NFS server, and a private Docker registry, while other nodes run Docker and NFS client. Images are built using Dockerfile based on Debian 8.6 base images.

Twelve servers are used in the experiments, with the base service node as master and others as workers. Servers connect via Gigabit Ethernet on the same

network segment, with the cluster operating on a private Overlay network. To simulate realistic deployment scenarios, all images are cleared before each experiment to ensure clean-slate execution. To prevent cache effects on computation time, all servers are rebooted after each experiment to clear computation caches. Experimental MS data is sourced from SAGECaL' s built-in test data.

2.2 Timeliness and Rationality

This experiment observes cluster deployment and computation time variations across different server quantities and cluster scales in a container cloud environment. Each deployment group runs five times, with average completion time recorded as the final deployment time. The cluster computation command is as follows:

To reduce computation load, each experiment group processes only one MS dataset with ADMM iterations set to 20. Process count is determined by container count, where each node corresponds to one container and one process. Command execution time is stored in `/root/project/ResultOutput/TimeProcessOutput`. Cluster cache files reside in `/tmp`. MS data to be processed is located in `/root/data/MS/`. Cluster computation uses the 3c196 model, with final results stored in `/root/project/ResultOutput/sm.ms.solutions`.

Sugon TianKuo 620R servers feature dual 4-core CPUs and 4GB memory, with 12 servers available total. Each cluster node launches one process during computation, meaning one process maps to one CPU core for cluster scales up to 96 nodes. Cluster computation time measures from command start to result acquisition. Overall deployment time includes image download from the private registry to cluster readiness. Cluster creation time measures deployment duration when images are already present on servers.

Statistical table of experimental results

The results show cluster creation time is extremely short. Including SAGECaL image transfer time, a 1000-node cluster deploys within one minute—far exceeding traditional manual deployment capabilities. As server quantity increases, computational capacity improves and cluster computation time decreases. However, when process count exceeds available CPU cores, computation time increases slightly due to extensive process switching.

[Figure 5: see original paper] Cluster deployment time
[Figure 6: see original paper] Deployment time & calculation time

As shown in Figure 5, image network transfer time still dominates the deployment process, with noticeable network transmission overhead during image distribution. However, Figure 6 reveals that for 1000-node clusters, overall automatic deployment time accounts for only 10% of total time compared to cluster computation time. Thus, deployment time overhead is not the primary factor; the performance bottleneck remains hardware computational capacity. Dis-

tributed integration of additional computational resources through automatic deployment methods proves rational for deploying computation-intensive applications like SAGECaL in container clouds.

3 Conclusion and Outlook

Automatic deployment methods based on container technology in cloud environments improve deployment efficiency while conveniently integrating additional computational resources, enabling astronomical application software to run across different hardware resources after a single compilation. Experimental results demonstrate that our proposed method significantly enhances cluster deployment efficiency, allowing users to focus on astronomical data processing.

To address massive data processing demands, some astronomical applications now support MPI-based distributed parallel computing. The container-based automatic deployment method provides robust support for all MPI-based astronomical applications, representing a general-purpose solution.

Current MPI cluster automatic deployment methods suffer from complete deployment failure if the master node fails to start first or crashes. While resource scheduling algorithms can allocate node locations during deployment, such scheduling only suits nodes of equal status. MPI clusters with master-worker hierarchies require improved scheduling mechanisms. Future work will explore Zookeeper or P2P algorithms combined with real-time dynamic performance collection and comparison of physical machines to develop a fault-tolerant, self-evolving automatic deployment method. This approach would enable equal-status initial node deployment, with nodes self-evolving into appropriate roles based on environmental performance and deployment conditions for dynamic automatic deployment.

We thank the National Astronomical Observatories-China Alibaba Cloud Astronomy Big Data Joint Research Center for supporting this work.

References

1. Chen Tairan, Wang Wei, Wang Feng, et al. The Study and Application of a High Performance UVFITS Assembly System Based on MPI[J]. Astronomical Research & Technology—Publications of National Astronomical Observatories of China, 2016, 13(2):184-189.

2. Shi Congming, Zhang Xiaoli, et al. Design and Implementation of the MUSER Negative Database Interfaces[J]. Astronomical Research & Technology—Publications of National Astronomical Observatories of China, 2018, 15(02):169-175.

3. Morris, D., et al., Use of Docker for deployment and testing of astronomy software. Astronomy & Computing, 2017. 20.

4. Young, M.D., S. Hayashi, and A. Gopu. StarDock: shipping customized computing environments to the data. in Software and Cyberinfrastructure for Astronomy IV. 2016.

5. Merkel, D., Docker: lightweight Linux containers for consistent development and deployment. 2014. 2014(239).

6. Rosado, T. and J. Bernardino, An overview of openstack architecture. 2014: ACM. 366-367.

7. Wei, S., et al., OpenCluster: A Flexible Distributed Computing Framework for Astronomical Data Processing. Publications of the Astronomical Society of the Pacific, 2016. 129(972): p. 024001.

8. Kazemi, S., et al., Radio interferometric calibration using the SAGE algorithm. Monthly Notices of the Royal Astronomical Society, 2011. 414(2): p. 1656-1666.

9. Kumazaki, K., S. Yatawatta, and S. Zaroubi. Performance of SAGECal calibration. in General Assembly and Scientific Symposium. 2014.

10. Naik, N. Building a virtual system of systems using docker swarm in multiple clouds. in IEEE International Symposium on Systems Engineering. 2016.

11. Nguyen, N. and D. Bein. Distributed MPI cluster with Docker Swarm mode. in Computing and Communication Workshop and Conference. 2017.

Automatic deployment method of astronomical application software based on container technology

Yao Kun[1] Dai Wei[1]* Yang Qiuping[1] Mei Ying[23] Shi Congming[1] Wang Feng[123]
[1]Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China
[2]Yunnan Observatories, Chinese Academy of Sciences, Kunming 650011, China
[3]Center for Astrophysics, Guangzhou University, Guangzhou 510006, China

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