Advanced Science and Technology Letters
Vol.120 (CCSC 2015), pp.721-726
http://dx.doi.org/10.14257/astl.2015.120.144

HARDMS: Management System for Distributed Clusters Based on Ambari

Changyi Yuan , Ruonan Rao,
School of Software, Shanghai Jiaotong University, China
{w_bear, rnrao}@sjtu.edu.cn

Abstract. In cloud computing environment, various distributed clusters are in use simultaneously. Ambari provides provisioning, managing and monitoring for Apache Hadoop clusters merely. The paper proposes an approach to manage multiple distributed clusters uniformly based on Ambari management model and implements the management system named after HARDMS. Our experimental results show that HARDMS can manage more kinds of distributed clusters, augment new type into management and have minimal performance impact to Ambari.

Keywords: distributed cluster management model Ambari

1 Introduction

Large-scale distributed clusters are widely used as cloud computing platform for a variety of applications and services. In order to take full advantage of machines, different distributed software clusters need to be shared with machines. When the software clusters deployed, they have to be scaled by business requirements, such as a certain business will increase, and the calculated amount will increase similarly, then new nodes have to be deployed and added in the distributed cluster. New computing clusters or storage clusters may be deployed in the machines which some other clusters are running in, therefore, the deploy process cannot influence original clusters.

In this work, we attempt to address this question: How to manage different kinds of distributed systems? Many approaches have been proposed. However, existing approaches only specific to one cluster or a certain class of clusters. The approach we proposed is based on Ambari management model and builds the model for every newly augmented distributed cluster. Our experimental results show that HARDMS can manage more kinds of distributed clusters, argument new type into management and have minimal performance impact on Ambari.

This paper is structured as followed. First, we introduce the related works on distributed cluster management. Next, we describe our design and implementation. Finally, we evaluate our HARDMS and conclude.

ISSN: 2287-1233 ASTL

2 Related Works

In this section, we discuss some operational tools which are generally used to manage distributed system, some management systems to manage a certain kind of distributed clusters and Ambari system which our system based on.

Multiple operational tools exist specially to manage distributed clusters, such as puppet, saltstack, chef, ansible [1, 2, 3, 4]. These tools are able to carry out your management tasks to the distributed clusters with your execution scripts. And they perform excellently in configuration management. However, when encountering the circumstances that mutli-class distributed clusters and multiple instances for one class cluster exist simultaneously, these tools cannot cope with following questions: (1) One operation fails in the process of deploying the cluster, how to go on after the reasons for failure repaired. (2) How to bring a new class of distributed cluster into unified management.

CloudDOE [5] is a user-friendly tool to deploy Hadoop cluster. Bright Cluster Manager [6] which is a commercial software system aims to establish management for a specified class of clusters. These systems are great at management for specified clusters. Nevertheless, with the requirement multiple clusters running on an array of machines, they cannot satisfy.

Ambari [7] is aimed at making Hadoop management simpler for Apache Hadoop clusters. Furthermore, it provides intuitive user interface for normal consumers and RESTful APIs for developers. Ambari is a distributed system in itself, including a master node and many agent nodes. Agent nodes firstly send heartbeat to master node, and master node responses with commands passively. It can support multiple instances and multi-class clusters in Hadoop ecosystem. In accordance with the characteristics of the Hadoop ecosystem, a model was used to manage Apache Hadoop clusters. This paper extends the model and design four modules to support different kinds of distributed clusters out of Hadoop ecosystem which are used for the project smart community. The model divides every cluster into many parts mapping different roles.

3 Design

This section describes system architecture and every module (Interposition Module, Model Builder, Command Module and Execution Module) to design HARDMS for different kinds of distributed clusters.

3.1 System Architecture

To manage different kinds of distributed clusters, HARDMS must model all progresses of these distributed clusters and augment them into the original list which HARMDS is managing. In order to arrive it, the architecture of HARDMS includes four modules: Interposition Module, Model Builder, Command Module and Execution Module, as depicted in Fig 1.

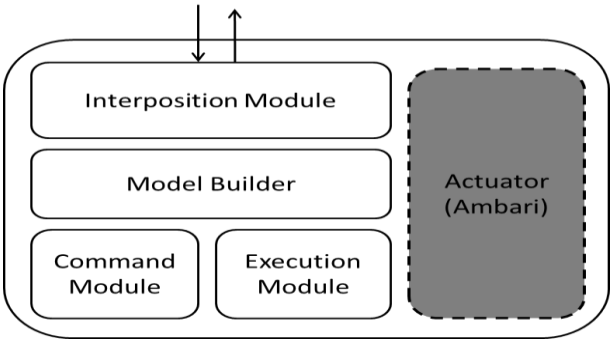

Fig. 1. Architecture of HARDMS

3.2 Interposition Module

The functions of Interposition Module are judging whether the model of specified cluster exists and leading user to build model.

The models of all distributed cluster managed by the system are stored in persistent database. Simultaneously, some models such as Hadoop, Hbase and some others are predefined [8]. The same kind of distributed clusters share one model. If the model needed is not defined in model library, Interposition Module feedbacks user to build model necessarily. Particularly worth mentioning is the model name of every kind of distributed cluster is unique. How it is achieved is described in Model Builder section.

Another function of Interposition Module is leading user to build model. Because the elements of model are incident with one another, for example, Node-Component belongs to Component and Component belongs to Service, then Node-Component and Component must be defined after Service defining, they must be defined in sequence.

3.3 Model Builder

Model Builder builds models for distributed cluster and stores them in the database, and ensures the parts in the model are unique globally. In order to make distributed cluster controllable, possible states of any parts in the model and the transform relations among them need to be maintained. So the operations on cluster do.

Every type of distributed cluster is an instance of Service, and the name must be unique. Component, Operation and other parts are in the same way. Along with cluster quantity increasing, there is more risk that two parts have same name. As long as the name of Service is unique, and the other parts belonging to this cluster are named after Service name prefix, the names of all parts are unique and distinct meaning.

The processes of cluster running on different hosts may be in different states which

are important to users and management system. And the transformation of process states in its life cycle is assured. Therefore, Model Builder builds finite-state machine for clusters.

3.4 Command Module

Command Module generates commands for operations of distributed cluster management. The command content contains the script type (which is supported by the hosts), script location, input parameter file location and other information. And actual script files and input parameter files are managed by Execution Module. The commands are serialized in JSON format to transfer to agents. Accordingly, agents can parse the messages received in JSON format and execute the script.

3.5 Execution Module

As mentioned in Command Module section, script files and input parameter files are managed by Execution Module. Script files can be shell script, python script or puppet script and input parameter files are strings in JSON format. Script files and input parameter files will be transferred when Server responds the heartbeats of agents. Therefore, script type is determined by agent environment.

4 Implementation

This section describes the specific implementation strategy used in HARDMS for different distributed clusters. On account of easy to implement Interposition Module, concentration will be on the other three modules.

4.1 Foundation

We implemented our system based on the Ambari 1.6.1. We use the same model for cluster management described in section Ⅱ. Model Builder is implemented in Java. Both Command Module and Execution Module are implemented in Java and Python (Server part implemented by Java and agent part implemented by Python).

4.2 Model Builder Implementation

As mentioned earlier, new management model is added to model container with new distributed cluster being brought into management. In order to make the system acquire that a new cluster type is added as soon as possible and not be affected the normal use, JMX (Java Management Extensions) technique is applied. Two MBean interfaces are defined, respectively mapping Service and Role in Ambari.

4.3 Command Module Implementation

For all distributed clusters, five commands (install, uninstall, start, stop, service_check) must be implemented. They are named in the form of ServiceName-Command, for instance, MONGOD-START means starting mongod command. For other customized commands, JMX technique is used again, which augments commands dynamically. And for all commands, it needs to modify role_command_order.json file which uses command defined to define sequence.

4.4 Execution Module Implementation

Script files are actually performed to complete commands generated by Command Module. Python script was served as target script for Execution Module in our implementation. And target scripts generated are placed in the resource directory. Template design pattern is used to implement universal operations. For example, install operation is divided into three operations: preinstall, install, postinstall. Operation preinstall does checks for the components to be installed. And operation install completes installation. And operation postinstall does clear-up work after installation. For other operations, just ensure executor can execute successfully.

5 Evaluation

In this section, we evaluate (1) the kinds of distributed cluster can be managed by HARDMS, (2) request latency for HARDMS and Ambari.

HARDMS proposed above is being applied in smart community service platform. It proves that the management system achieves the goal.

Fig 2 shows kind quantity which HARDMS and Ambari can manage distributed clusters. HARDMS can manage more clusters as more models augmented. However, Ambari can only manage clusters in Hadoop ecosystem.

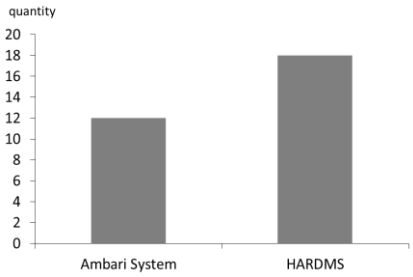

Fig. 2. kind quantity Ambari and HARDMS

Fig 3 shows the latency in Ambari system and HARDMS when user makes a request.

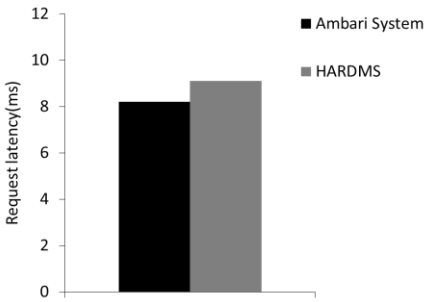

Fig. 3. request latency for Amari and HARDMS

Experiment results show that HARDMS can manage generally different kinds of distributed clusters, have minimal performance impact to Ambari.

6 Conclusion

This paper proposes an approach to extend Hadoop management system Ambari to support more different kinds of distributed clusters. As long as the model of the distributed cluster which will be managed by the system can be established successfully, it is possible to manage the cluster by extending Ambari. And it proves feasible by evaluation experiments.

References

1. Baker M, Fox G C, Yau H W. A review of commercial and research cluster management software[J]. 1996.
2. Hosmer B. Getting started with salt stack--the other configuration management system built with python[J]. Linux journal, 2012, 2012(223): 3.
3. Turnbull J. Pulling strings with puppet: configuration management made easy[M]. Springer, 2008.
4. Mohaan M, Raithatha R. Learning Ansible[M]. Packt Publishing Ltd, 2014.
5. Chung W C, Chen C C, Ho J M, et al. CloudDOE: A User-Friendly Tool for Deploying Hadoop Clouds and Analyzing High-Throughput Sequencing Data with MapReduce[J]. 2014.
6. http://www.brightcomputing.com/Bright-Cluster-Manager.
7. Wadkar S, Siddalingaiah M. Apache Ambari[M]//Pro Apache Hadoop. Apress, 2014: 399-401.
8. Aravinth M S S, Shanmugapriyaa M S, Sowmya M S, et al. An Efficient HADOOP Frameworks SQOOP and Ambari for Big Data Processing[J]. International Journal for Innovative Research in Science and Technology, 2015, 1(10): 252-255.

