2008

A cumulus project: design and implementation

Lizhe Wang

Gregor von Laszewski

Marcel Kunze

Recommended Citation

Online in svn repository: http://code.google.com/p/cyberaide/source/browse/trunk

A cumulus project: design and implementation

Lizhe Wang and Gregor von Laszewski
Service Oriented Cyberinfrastruture Lab, Rochester Institute of Technology
102 Lomb Memorial Drive, Rochester, NY 14623-5608, U.S.
Email: Lizhe.Wang@gmail.com, laszewski@gmail.com

Marcel Kunze and Jie Tao
Steinbuch Centre for Computing, Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1 D-76344 Eggenstein-Leopoldshafen, Germany
Email: Marcel.Kunze@iwr.fzk.de, Jie.Tao@iwr.fzk.de

December 21, 2008

Abstract

The Cloud computing becomes an innovative computing paradigm, which aims to provide reliable, customized and QoS guaranteed computing infrastructures for users. This paper presents our early experience of Cloud computing based on the Cumulus project for compute centers. In this paper, we introduce the Cumulus project with its various aspects, such as design pattern, infrastructure, and middleware.

1 Introduction

The Cloud computing is coined in late of 2007 and currently emerges as a hot topic. A computing Cloud is able to offer flexible dynamic IT infrastructures, QoS guaranteed computing environments and configurable software services. As reported in the Google trends shown in Figure 1, the Cloud computing (the blue line), which is enabled by virtualization technology (the yellow line), has already outpaced the Grid computing [1] (the red line).

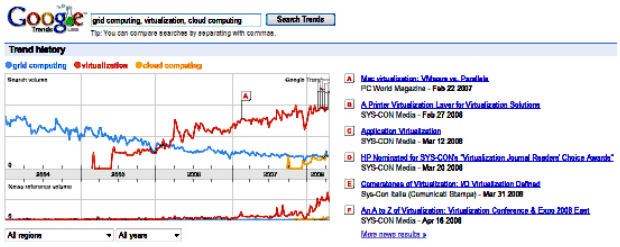

Figure 1: Cloud computing in Google trends

This paper reports our experience on developing a scientific Cloud with some popular Cloud technologies. It is intended to demonstrate flexible design and implementation of a scientific Cloud, which does not bind to any specific underlying technologies. We declare that the Cumulus project enjoys the features such as flexibility, scalability and modularity. The rest of the paper is organized as follows. Section 2 presents related work of Cumulus project. Section 3 overview the design of Cumulus project. Section 4, Section 5, Section 6 and Section 7 show the various aspects of Cumulus project, such as the frontend service, backend service, virtual machine image preparation, and network solution. Finally Section 8 concludes the paper.

2 Related work

2.1 Globus virtual workspace and Nimbus

A virtual workspace [2,3] is an abstraction of a computing environment which are dynamically available to authorized clients via invoking Grid services. Based on Globus virtual workspace services, a cloudkit named Nimbus [6] is developed to build scientific Clouds. The Nimbus contains a frontend Globus service and multiple workspace control agents on host resources for virtual machine deployment.

2.2 OpenNEbula

The OpenNEbula is a virtual infrastructure engine that enables the dynamic deployment and re-allocation of virtual

machines in a pool of physical resources. The OpenNEbula system extends the benefits of virtualization platforms from a single physical resource to a pool of resources, decoupling the server, not only from the physical infrastructure but also from the physical location [8].

The OpenNEbula contains one frontend and multiple backends. The front-end provides users with access interfaces and management functions. The back-ends are installed on Xen servers, where Xen hypervisors are started and virtual machines could be backed. Communications between frontend and backends employ SSH. The OpenNEbula gives users a single access point to deploy virtual machines on a locally distributed infrastructure.

2.3 OS Farm

The OS Farm [5] is an application written in Python to generate virtual machine images and virtual appliances used with the Xen VMM. Two methods are provided for users to connect to the OS Farm server:

- Web interface

  The Web interface provides various methods to enable users to request virtual machine images. The simplest method is simple request which allows users to select image class and architecture. The advance request gives the possibility to add *yum* packages to the virtual machine image. A drop down menu allows the user to choose packages from a list of predefined *yum* repositories. Using asynchronous Java Script and XML, a further list is returned which allows users to select the corresponding *yum* packages. Support for request via XML descriptions is also available. An image description can be uploaded through the Web interface of the OS Farm server as an XML file. A sample XML description is shown below:

```
<image>
    <name>myvm</name>
    <class>slc4_old</class>
    <architecture>i386</architecture>
    <package>emacs</package>
    <package>unzip</package>
    <group>Base</group>
    <group>Core</group>
</image>
```

- HTTP interface

  The OS Farm also provides the HTTP interface. For example, users can use the *wget* to access the HTTP interface of the OS Farm server. Below is a sample usage of the HTTP interface with the *wget* command:

```
wget
http://www.fzk.de/osfarm/create?
name=vm1&
transfer=http&
class=slc_old&
arch=i386&
filetype=.tar&
group=core&
group=base
```

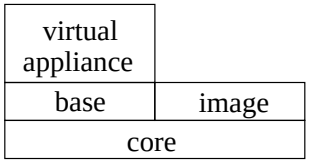

Figure 2: Layers of the OS Farm image generation

The images in the OS Farm are generated in layers which can be cached. The generation of a virtual machine image is divided into three layers or stages. Figure 2 shows the layers of virtual machine image generation:

- The *core* layer is a small functional image with a minimal set of software which is either required to run the image on a VMM or required to satisfy higher level software dependencies.
- The *base* layer provides some additional software needed in order to satisfy requirements for *virtual appliances*.
- the *image* layer is on top of *core* layer and provides user-defined software in addition to the core software.
- A subclass of image is *virtual appliance*, which is an image with an extra set of rules aimed to allow the image to satisfy requirements of the deployment scenario.

3 Design of Cumulus project

The Cumulus project is an on-going Cloud computing project, which intends to provide virtual computing platforms for scientific and engineering applications. The Cumulus project currently is running on distributed IBM blade servers with Oracle Linux and the Xen hypervisors. We design the Cumulus system in a layered architecture (see also Figure 3):

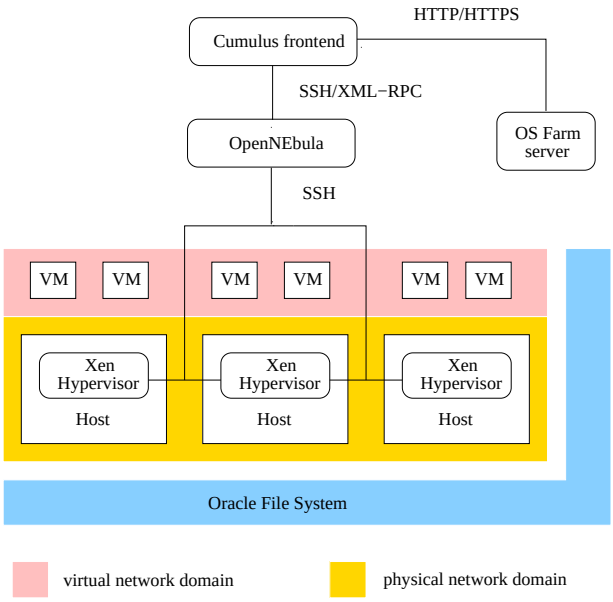

Figure 3: Cumulus architecture

- The Cumulus frontend service resides on the access point of a computing center and accepts users' requirements of virtual machine operations.
- The OpenNEbula works as Local Virtualization Management System (LVMS) on the Cumulus backends. The OpenNEbula communicates with the Cumulus frontend via SSH or XML-RPC.
- The OpenNEbula contains one frontend and multiple backends. The OpenNEbula frontend communicates to its backends and the Xen hypervisors on the backends via SSH for virtual machine manipulation.
- OS Farm
  The OS Farm [5] is a Web server for generating and storing Xen virtual machine images and Virtual Appliances. The OS Farm is used in the Cumulus system as a tool for virtual machine template management.

This design pattern could bring the scalability and keep the autonomy of compute centers:

- The Cumulus frontend does not depend on any specific LVMS, we use a third party software – the OpenNebula as an example here.
- Compute centers inside computing Clouds could define their own resource management policies such as network solution or virtual machine allocation.
- The Cumulus frontend can also delegate users' requirements to other computing Clouds.

4 Cumulus frontend: re-engineering of the Globus Virtual Workspace Service

The Globus Virtual Workspace Service (GVWS) [3] contains two pieces of software: the GVWS frontend and the GVWS control agents.

The GVWS frontend receives the virtual machine requirements and distributes them to various backend servers via SSH. The GVWS control agent is installed on every backend server and communicates with the the Xen server in order to deploy a virtual workspace.

In principal the GVWS frontend could be employed as a Cloud frontend service. However some limitations of the GVWS are identified :

- Computer centers in general run their own specific LVMS, like OpenNEbula or VMware Virtual Infrastructure, to manage their local infrastructures. However, the GVWS demands to install the workspace control agents directly on the backend servers. This use scenario provided by the GVWS lacks of the generality.
- The GVWS provides three network settings: the *AcceptAndConfigure* mode, the *AllocateAndConfigure* mode and the *Advisory* mode. However users generally do not care about network configurations for virtual machines. Users only demand the virtual machine network address or host name so that they can finally access the virtual machines across the network. We suggest that the network configurations and solutions are transparent to Cloud users and they only controlled by the Cloud backends. This paradigm would be more reasonable considering that the Cloud infrastructure might span across multiple network domains.

The GVWS is re-engineered to adapt it as the Cumulus frontend service. This comprises the following steps:

- Extend the GVWS frontend and remove the control agent, thus allowing the GVWS to work with various virtual machine hypervisors and LVMS, such as OpenNEbula, VMware server and VMware Virtual Infrastructure.
- Support a new networking solution – the *forward* mode. In the *forward* mode, users input no network

configuration information; the backend servers allocate the IP address for the virtual machine and return it to users.

5 OpenNEbula as the Cumulus backend

The OpenNEbula is used to manage the local distributed servers, which provides virtual machine deployment. Currently the OpenNEbula employs the NIS (Network Information System) to manage a common user system and the NFS (Network File System) for virtual machine image management. However it has been widely recognized that the NIS has a major security flaw: it leaves the users' password file accessible by anyone on the entire network. The NFS has some minor performance issues and it does not support concurrency. To employ the OpenNEbula in a more professional way, we merged the OpenNEbula with some modern secure infrastructure solutions like the Lightweight Directory Access Protocol (LDAP) [4] and the Oracle Cluster File System (OCFS) [9] or the Andrew File System (AFS) [7].

The LDAP is an application protocol for querying and modifying directory services over TCP/IP. A directory is a set of objects with similar attributes organized in a logical and hierarchical manner. In the OpenNEbula scenario, the OpenLDAP software is used to implement the user's directory. A common user, *oneadmin*, is authorized on the OpenNEbula frontend and all the OpenNEbula backends.

The OpenNEbula frontend and the backends share directories using the OCFS. The shared space is used to store virtual machine images and to allow the OpenNEbula system to behave as single virtual machine hypervisor.

5.1 Communication between the Cumulus frontend and the OpenNEbula

The communication between the Cumulus frontend and the OpenNEbula service could be either SSH (Secure Shell) or XML-RPC: the Cumulus frontend requires virtual machine operations for example by remotely invoking the OpenNEbula commands via SSH or by sending XML-RPC messages via the OpenNEbula APIs. For other LVMS, like VMware Infrastructure, communication methods should be developed and plugged into the Cumulus frontend.

- communication with SSH

  Originally, the GVWS frontend issues its commands to its backends via SSH. Therefore the first attempt to communicate the Cumulus frontend with the OpenNEbula via SSH. However the SSH communication is not supported by current OpenNEbula implementation. The OpenNebula Command Line interface (CLI) is the interface used when issuing SSH commands. It identifies the virtual machines running on the OpenNEbula with an identification number when a virtual machine is started in the context of the OpenNEbula. As a consequence this identification number should be returned to the Cumulus frontend for later use, for example, to pause, resume and reboot the just created virtual machine. The OpenNEbula uses the terminal as the standard output to print the identification number. The SSH program, following Unix conventions, can only get the return code after it is executed. It cannot get the command line output, which is needed to be parsed to obtain virtual machine's identification number. Therefore, the SSH communication is not supported in the current implementation context.

- communication with XML-RPC

  The OpenNEbula exposes the XML-RPC interfaces for users to access its services. The OpenNebula's features could therefore be accessed from the client programs. This method in some aspects is superior to the previous CLI implementation since easy recovery of the virtual machine's identification number is possible. Upon creation of virtual machines, the XML-RPC call returns a parameter with the identification number of the created virtual machine. Also all other functions return the status of the executed commands and therefore a better control and interaction with the OpenNebula server are possible.

6 OS Farm for virtual machine image management

The Cumulus can generate virtual machine images for users by using the OS Farm.

An OS Farm client is embedded in the Cumulus frontend to generate virtual machine images. When Cumulus users send virtual machine requirements to the Cumulus frontend, the frontend invoke the OS Farm client, which thereafter sends HTTP requests to the OS Farm server. The virtual machine images generated by the OS Farm are transferred to the Cumulus virtual machine repository, which is implemented via the OCFS. Then users could manipulate virtual machines after they are stared by the Cumulus backends (see also Figure 4).

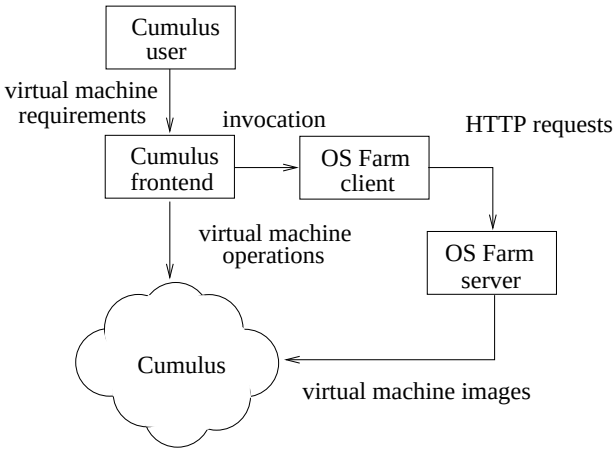

Figure 4: Using OS Farm in the Cumulus

The OS Farm client is implemented as follows:

1. The client gets virtual machine parameters from the Cumulus frontend, such as virtual machine name, transfer protocol, virtual machine class name, virtual machine architecture, virtual machine image file format, and image generation layer. An example of these parameters are "myvm, http, slc4_old, i386, .tar, core"

2. The client then constructs a HTTP request and send it to the OS Farm server. For example, a *wget* URL could be built like this:

```
wgeturl = osfarmserver +
"/create?name=" +name +
"& transfer =" + transferprotocol +
"& class =" + class +
"& arch =" + arch +
"& filetype =" + filetype
```

3. Finally the client obtains the virtual machine image from the OS Farm server and stores it in the Cumulus virtual machine repository. A connection is formed between the OS Farm server and the client with the help of *java.net.URL* class. It is demanded to read and write the coming data from the OS Farm server simultaneously due to the large size of the virtual machine image files (200MB – 2000MB). It is not possible to store such large image files in the default heap space of the Java virtual machine, which is normally around 128 MB.

7 Networking solution

We provide a new networking solution – the *forward* mode. Users do not have to specify the network configuration. The backend servers could implement the network solution for the incoming virtual machine requests. The *forward* mode could play an important role when the Cloud infrastructure span across multiple network domains.

When virtual machine requirements arrive at the Cumulus frontend, the frontend just forwards the requirements to the Cumulus backend. The backend sets the virtual machine network configuration and returns the network configurations to the frontend. The frontend thereafter again forwards the configurations to users. Finally users could access the virtual machine via the network configurations, normally the IP address or the host name of the virtual machine. Specifically, three methods to allocate virtual machine networking are implemented in the Cumulus backend:

- lease IP addresses from an IP address pool
  In this solution, a file named "ipPool" stores, as deducted from the name, a pool of IP addresses. In this context, when the fontend submits the virtual machine to the backend, the first available IP address in the file will be taken and be parsed as a Xen parameter. Since the OpenNebula (via Xen) already supports setting a virtual machine IP address, it is available that on virtual machine's creation the virtual machine is configured with the provided IP address. Other networking parameters, such as gateway, subnet mask, DNS and broadcast, can be specified in OpenNebula's configuration files. This solution already provides functions to avoid reusing an already consumed IP address and assures that a new IP address is used for every new virtual machine. The administrator of the backend is only required to declare in the file the available IPs.

- identify a virtual machine with the host name
  In this implementation, the host name parameter uniquely identifies a virtual machine. When the frontend sends a virtual machine to the backend, the backend set the host name for the virtual machine, then returns it to the frontend for later reference. However, the OpenNebula is not yet capable of asking the Xen server to set a host name. It is demanded to ask the OpenNebula to directly submit this parameter to the Xen hypervisor by setting the OpenNebula's "RAW" variable. This variable is used to directly pass to the Xen server, on virtual machine's creation, any additional Xen parameter users

would like to add. Moreover, this solution permits no further configuration of any additional files, since the implementation already generates a unique host name on virtual machine's creation time. The backend can return the host name to users for later reference.

- get IPs from a central DHCP server
  In this implementation, when the frontend sends virtual machines to the backend, the backend sets the networking item in the configuration file with "DHCP". Then the virtual machine is started, listens to the central DHCP server and gets the network configurations. However, the problem here is how to get the IP address of the started virtual machine and returns it to the frontend. Since every virtual machine has its own unique MAC address, one solution is to query the DHCP server to obtain the virtual machine's IP address by providing the MAC address. On successful reply, the backend would forward the IP address to the frontend, so that user could access the virtual machine. A possible drawback related to this solution is its dependence on certain specific networking configuration. Depending on administrator's policies of the backend, some queries to the DHCP server might be banned.

8 Conclusion and outlook

This paper presents a prototype of scientific Cloud – the Cumulus project. The Cumulus project aims to build a scalable scientific cloud: Cumulus could adopt to various virtual machine hypervisors as backends, delegates users' requirements to other Clouds or existing Grid testbed, and provision desired computing infrastructure on demand. The implementation justifies the resource re-organization and service orchestration for Cloud computing.

References

[1] I. Foster and C. Kesselman. *The Grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 1998.

[2] K. Keahey, K. Doering, and I. Foster. From sandbox to playground: dynamic virtual environments in the Grid. In *Proceedings of the 5th International Workshop on Grid Computing*, pages 34–42, 2004.

[3] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: achieving quality of service and quality of life in the Grid. *Scientific Programming*, 13(4):265–275, 2005.

[4] V. A. Koutsonikola and A. Vakali. LDAP: framework, practices, and trends. *IEEE Internet Computing*, 8(5):66–72, 2004.

[5] OS Farm project [URL]. http://cern.ch/osfarm /, access on July 2008.

[6] Nimbus Project [URL]. http://workspace.globus.org/clouds/nimbus.html/, access on June 2008.

[7] Open AFS [URL]. http://www.openafs.org/, access on Sep. 2008.

[8] OpenNEbula Project [URL]. http://www.opennebula.org/, access on Apr. 2008.

[9] Oracle Cluster File System [URL]. http://oss.oracle.com/projects/ocfs/, access on June 2008.