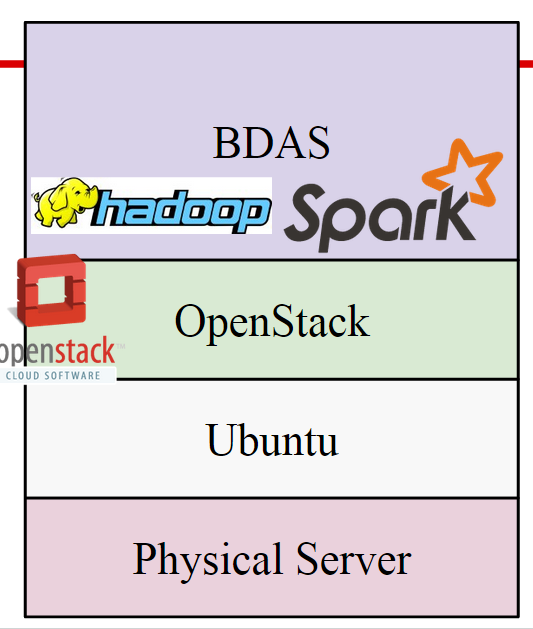
**5.1 System Design**

Because of the tough problems we met in setting up XenServer, we decided to install Openstack based on the Ubuntu instead. Based on the openstack, we then build hadoop and spark. But before configure hadoop and spark, we should set up Sahara which provides users with simple means to provision a Hadoop and a Spark cluster.



***Figure 5.1 The Design using Ubuntu Server as the underlying hypervisor***

Observe that the difference between figure 5.1 and figure 4.1, in the new design hadoop and Spark are considered to be part of BDAS.

**5.2 Setup Methodology**

Openstack is a very complex cloud computing platform. It does provide some ‘plug-and-play’ setup method, but in order to have the big data processing functionality, special configurations are needed.

**5.2.1 General Openstack Setting**

Generally Openstack could be configured via devstack[2]. devstack provides a bunch of shell script and utilities to help you build a local cloud computing platform, as long as you write the configuration file correctly.

Part of the configuration file is presented here.

|  |
| --- |
| **#----NETWORK CONFIGURATION----#**  **#The IP address of the current host of openstack**  **HOST\_IP=10.0.21.5**  **FIXED\_RANGE=192.168.1.0/24**  **FIXED\_NETWORK\_SIZE=256**  **FLOATING\_RANGE=10.0.20.56/29**  **#-- possible setting for sahara --#**  **enable\_service s-proxy s-object s-container s-account**  **SWIFT\_REPLICAS=1**  **#-------AUTO ASSIGN FLOATING IP ENABLE, POSSIBLE PROBLEM HERE-------#**  **EXTRA\_OPTS=(auto\_assign\_floating\_ip=True)**  **LOGFILE=/opt/stack/logs/stack.sh.log**  **VERBOSE=True**  **LOG\_COLOR=False**  **SCREEN\_LOGDIR=/opt/stack/logs**  **ENABLED\_SERVICES+=,sahara** |

It is important to properly set the fixed ip range so that it will not collide with the outside network, otherwise a newly launched instance will be lack of connectivity to the world wide web.

**5.2.2 Sahara Setting in Openstack**

After the general Openstack setting, the question of “how to perform Big Data analysis with Openstack?” came to mind. “Sahara” is Openstack’s answer to big data question. It is an integrated project in the Juno release, and actually started out as something called Savanna.

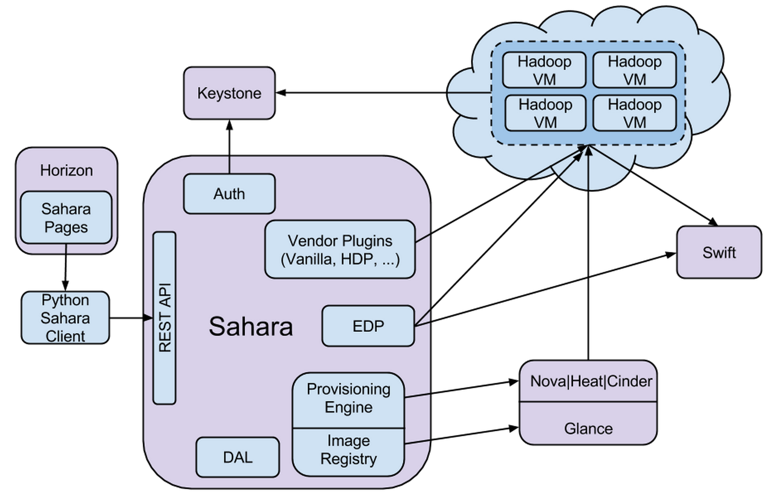
Sahara aims to provide users with simple means to provision a Hadoop cluster at OpenStack by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more.

The key features of Sahara are:

1. Fast Hadoop cluster deployment.
2. Supports different types of jobs: MapReduce, Hive, Pig and Oozie workflows.
3. Data source can be taken from various sources: Swift, HDFS, NoSQL and SQL databases.

**5.2.2.1 OpenStack Configuration**

The architecture of Saharais shown in figure 5.2.



***figure 5.2 The Architecture of Sahara***

There are 6 key components in Sahara. I will introduce components that related to our project.

* Auth -- responsible for client authentication and authorization, communicates with Keystone.
* Vendor Plugins -- mechanisms responsible for configuring and launching Hadoop on provisioned VMs.
* Provisioning Engine and Image Registry -- work with Nova, Heat, Cinder and Glance, which are the key components in OpenStack to provision engine and register images for Sahara.
* EDP(Elastic Data Processing) -- responsible for scheduling and managing Hadoop jobs on clusters provisioned by Sahara. EDP supports Hive, Pig, MapReduce, MapReduce Streaming and Java job types on Hadoop clusters. EDP have access with Swift and HDFS and can work with different types of databases.
* Rest API -- exposes Sahara functionality via REST and supply GUI for the Sahara is located on Horizon.

To setup Sahara upon openstack, there are three basic steps:

1. Put command ‘$echo “enable\_service sahara” ’ into OpenStack configuration file -- “localrc”
2. After enable the Sahara Service, put command ‘$echo “plugins=vanilla,hdp,spark” ’ into configuration file in Sahara -- ‘/etc/sahara/sahara.conf ’
3. Install new special images for data processing with command lines:

|  |
| --- |
| glance image-create --name UbuntuTrustyVanilla2.4.1 \  --disk-format qcow2 --container-format bare --is-public True \  --copy-from http://sahara-files.mirantis.com/sahara-juno-vanilla-2.4.1-ubuntu-14.04.qcow2 |

**5.2.2.2 Hadoop Setup**

After the installation of Openstack and Sahara, we are able to further configure the hadoop and spark cluster.

First, make sure that the hadoop and spark are set to “enable” in the “sahara.conf” file. Then restart sahara, and the “data processing” tag should appear in the console page, like below:



***figure 5.3 The sub-options of Openstack Project***

With the data processing tag appeared, we can configure to launch clusters. Download the hadoop cluster image from the openstack website, make sure the image is the newest version of “vanilla hadoop” image. This image is designed for the hadoop clusters used for cloud computing. After downloading the image, log into the openstack as the admin, and upload the image with the “image -> create image” button. As the image is the qcow2 format, we did a transformation. Although the openstack console page supports QCOW2 format, the NOVA only support the VMDK format image. We converted the image file from QCOW2 to VMDK using the following command:

$ qemu-img convert -f qcow2 -O VMDK xxx.qcow2 xxx.vmdk

After the transformation, we uploaded the .vmdk file to the openstack and created a new image.

However, only by registering the image on openstack is not enough. We still need to register it on the sahara. So using the “register image” button under the “data processing” tag, we registered the image to sahara. Make sure to add tags to the image registered, there are at least two tags, indicating the version of the hadoop and the version of the Ubuntu. We once suffered a lot trying to figure out the reason why our clusters cannot be successfully launched and finally realized we have to tag the image before launching virtual machines.

After configuring the image, we still need to configure the template of launching. We used the “Node Group template” tag to create two group templates, one worker node group and one name node group. The “Availability Zone” “OpenStack Flavor” can be set according to the user’s need, the “Storage Location” however, has to be set as “Ephemeral Drive”. Besides, based on our network configuration, the “Floating IP Pool” need to be configured as “Do not assign floating IPs”. The other configures like “HDFS Parameters” or “ MapReduce Parameters” can be set based on the reality. Another thing that we studied a lot is that in a hadoop cluster, the number of the name node should be exactly one, and the number of the data node should be at least three. If the number of the nodes are configured incorrectly, the cluster cannot be launched.

**5.2.2.3 Spark Setup**

Spark is a fast and general engine for large-scale data processing. It has a lot of advantages as the followings:

(1)Speed: Run programs up to 100x faster than Hadoop.

(2)Ease of Use: write applications quickly in Java, Scala or Python.

(3)Generality: combine SQL, streaming and complex analytics.

(4)Runs Everywhere: Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, HBase, S3, etc.

When compared to Hadoop, Spark is a in-memory version of Hadoop. Hadoop use HDFS while Spark use streaming.

When setting up Spark, we should first download the Spark cluster image from the openstack website. We install new special images with command lines as follows:

|  |
| --- |
| glance image-create --name UbuntuTrustySpark1.0 \  --disk-format qcow2 --container-format bare --is-public True \  --copy-from \  http://sahara-files.mirantis.com/sahara-juno-spark-1.0.0-ubuntu-14.04.qcow2 |

Then the steps are similar with setting up Hadoop. We need to register it both on the Spark and Sahara. Also make sure to add tags to the image registered. After configuring the image, it’s time to configure the template of launching. We used the “Node Group template” tag to create two group templates, one worker node group and one name node group. Then used the “Cluster Template” to create two cluster templates. And then we can create cluster. Later, we use the command ‘ssh -i xxx.key ubuntu@xxx’ to connect to Spark. At first, We cannot ssh to ubuntu successfully. Then we found that we should wait for a long time because creating cluster took much time. It was necessary for us to download the scala language first before running spark shell. So we use the command to download and configure Scala:

|  |
| --- |
| $ wget http://www.scala-lang.org/files/archive/scala-2.10.4tgz  $ tar xvf scala-2.10.4.tgz  $ ln -s scala-2.10.4 scala  $ export SCALA\_HOME=/home/ubuntu/scala  $ export PATH=$SCALA\_HOME/bin:$PATH |

We then use a ‘Hello World’ program to test whether the Scala is there.

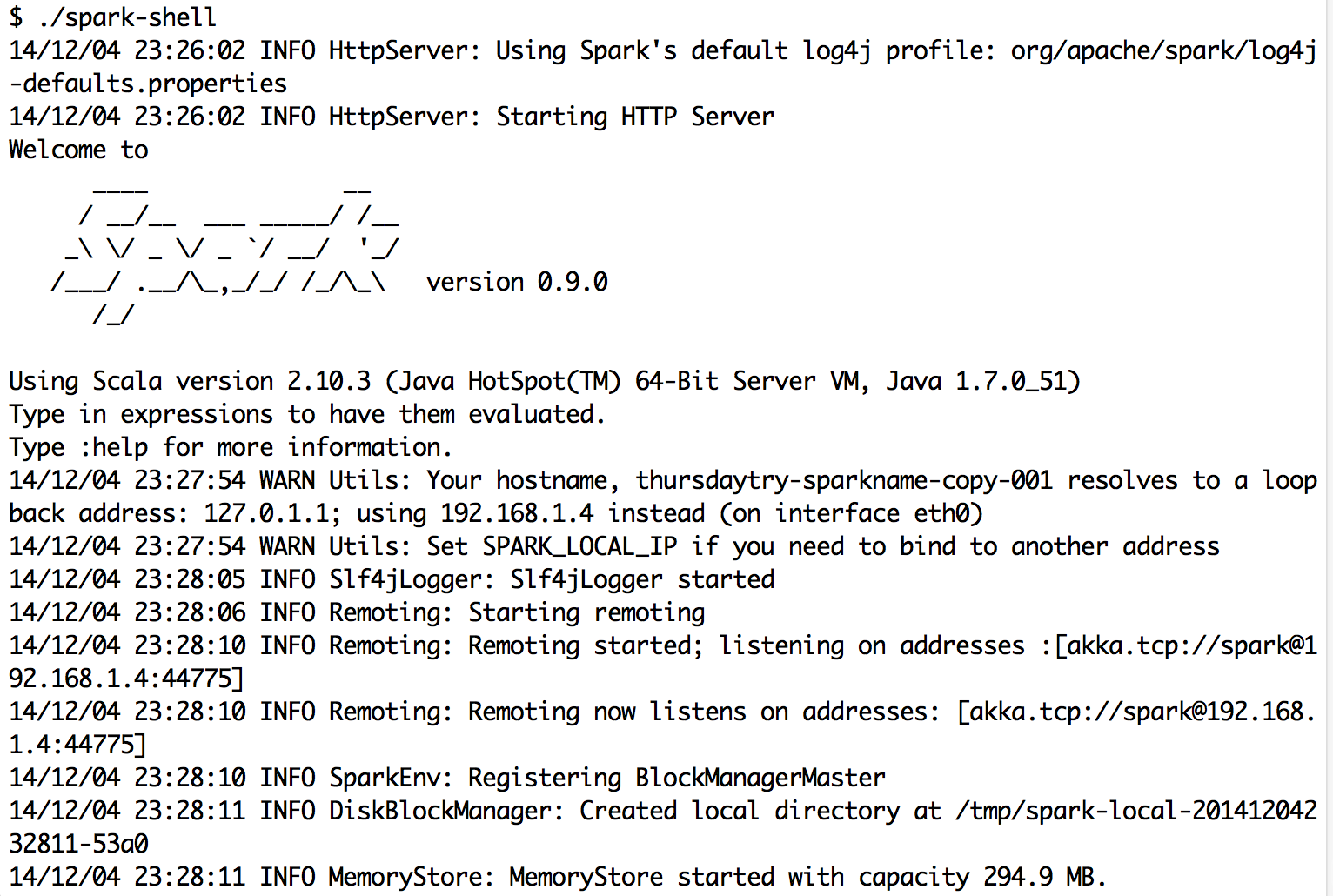
|  |
| --- |
| scala>object HelloWorld{  | def main(args: Array[String]) {  | println(“Hello, world!”)  | }  | }  defined module HelloWorld  scala >HelloWorld.main(null)  Hello, world! |

This result confirms that we install scala successfully.

Then it is time to install and launch Spark.

|  |
| --- |
| $ wget http://d3kbcqa49mib13.cloudfront.net/spark-0.9.0-incubating-bin-cdh4.tgz  $ tar xvf spark-0.9.0-incubating-bin-cdh4.tgz  $ ln -s spark-0.9.0-incubating-bin-cdh4.tgz spark  $ cd bin  $ ./spark-shell |

Figure 5.4 shows launching Spark successfully.



***figure. 5.4 Spark launched successfully***

Apache Spark is a ***cluster computing engine***. It abstracts away the underlying distributed storage and cluster management aspects. The main abstraction for computations in Spark is [***Resilient Distributed Dataset(RDD)***](https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf). Due to its simplified programming interface, it unifies computational styles which were spread out in otherwise traditional Hadoop stack. MapReduce model is composed of following stages: Map-->shuffle and sort -->Reduce. But RDD does not restrict us in traditional MR way. This is a MapReduce example in Spark:

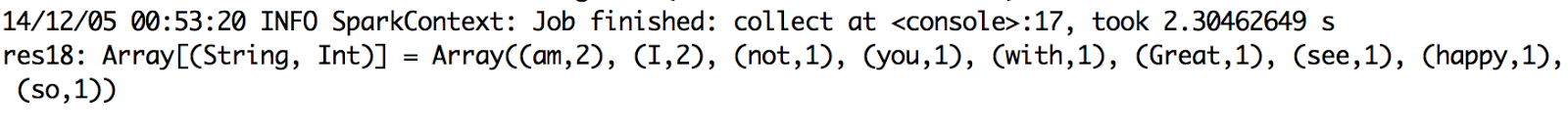
(1) Input a text as followings:

|  |
| --- |
| I am so happy with you  Great I am not see |

(2)Use the following command:

|  |
| --- |
| scala> val textFile = sc.textFile(“home/ubuntu/spark/input.txt”)  scala> val wordCounts = textFile.flatMap(line => line.split(“ ”)).map(word => (word, 1)).reduceByKey((a,b) => a + b)  scala> wordCounts.collect() |

Figure 5.5 shows Result.



***figure. 5.5 Result of the program***

From the result, we can find that running MapReduce in Spark is easier and faster.