Building a 6-node Hadoop Cluster

Xiangwen Wang

July 15, 2015
Blacksburg, VA

Table of Contents

1 Project Overview

In order to handle the huge amount of data we've collected in our human mobility pattern project and virtual currency transaction project, we want to build a multi-user bigdata-analysis platform. After some investigation, we decided to build a Hadoop [1] cluster using ClouderaManager.

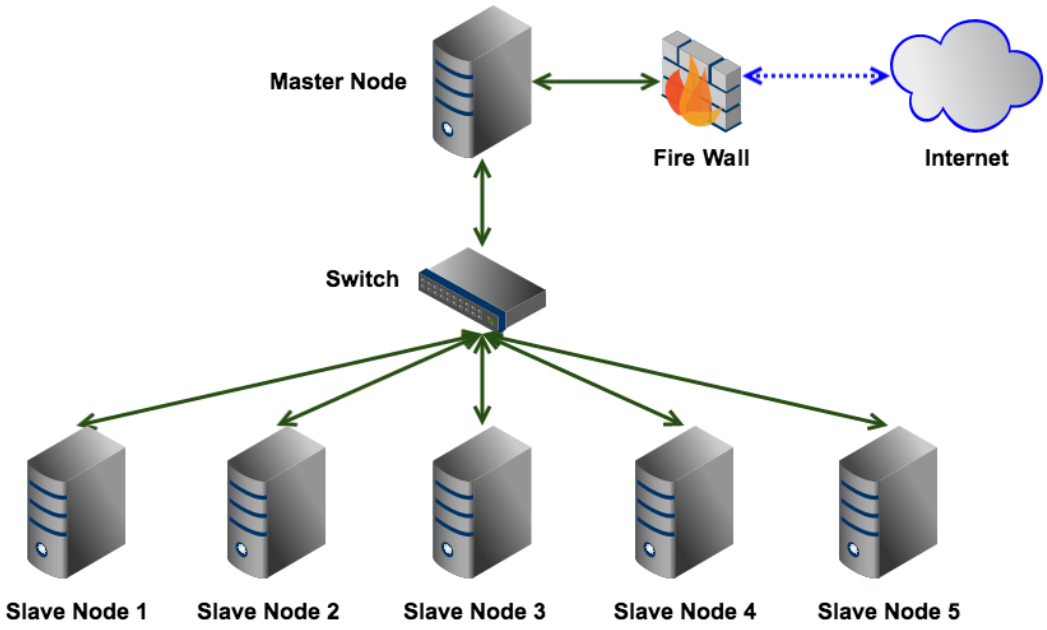
ClouderaManager [2] is a management application provided by Cloudera that could help us easily setup, deploy, and manage CDH (Cloudera Distribution Including Apache Hadoop [3]), an integrated and flexible Hadoop distribution also provided by Cloudera, on our cluster.

Below, we'll show the procedures we build our Hadoop cluster from the very beginning.

2 Hardware installation

2.1 System design

In our cluster, we use 1 Master Node as the manager node and 5 Slave Nodes as the Hadoop nodes. In the internal network, an Ethernet switch connects all the nodes. At the same time, the Master Node has an additional wireless network card to get access to the Internet. And all the Slave Nodes can get access to Internet through the Master Node. However, Data are transferred into or out of the cluster through external hard drives due to the limitation of the bandwidth of the wireless connection.



2.2 Hardware specification

- 1 Master Node and 5 Slave Nodes.
- One quad-core CPU on each node, 24 cores in total.
- 8GB memory on each node, 48GB in total.
- One 750GB enterprise-level hard drive on each node to install the operating system and Hadoop, and one additional 750GB hard drive on Master Node for temporary storage and backup. 5.25TB storage space in total.
- An 8-port Gigabit Ethernet switch connects all the nodes.
- A PCI-E wireless network adapter on Master Node with Internet connection.

3 Cluster installation

3.1 Operating system installation

On each node, we install CentOS as the operating system. Here are the procedures:

1. Choose "New Installation" when installing the CentOS 6.6.
2. Under the "Please name this computer" section, respectively use "master1", "slave1" to "slave5" as the names for the Master Node and Slave Nodes.
3. Use the same Root password for all of the nodes.
4. In the "Which type of installation would you like?" section, choose "Use All Space" option, and also, check the "Review and modify partitioning layout" box at the bottom of that page.
5. Switch the default sizes of the lv_root (/) and lv_home (/home) before disk partitioning. Since the volume of the hard drive is 750GB, we choose 51,200MB for lv_home, and 655,828MB (size may vary for different hard drive manufacturers) for lv_root.
6. Enable the NTP (Network Time Protocol) service to synchronize the time.
7. In the rest sections, use default settings.
8. When finishing the installation, for the Master Node, install necessary packages with following procedures:
    - Get access to the Internet through a wireless network card.
    - Install GCC through yum by running the following command in shell:

      yum –y groupinstall "Development tools"
    - Install Python packages:

      yum –y install python-pip python-devel numpy scipy python-matplotlib
    - Add NTFS support:

      yum –y install epel-release; yum –y insall ntfs-3g
    - Install OpenMPI for parallel computing by:

      yum –y install openmpi-devel

      And editing the .bashrc file by adding the following two lines:

      ```
      export PATH=$PATH:/usr/lib64/openmpi/bin
      export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64/openmpi/lib
      ```

9. On all nodes, to save memory usage, we will only keep the command line login by editing the file /etc/inittab, and change the line "id:5:initdefault:" to "id:3:initdefault:".
10. On all nodes, disable selinux by editing /etc/selinux/config, and changing the line "SELINUX=enforcing" to "SELINUX=disabled".
11. On all nodes, change the swappiness to 5 by running the command:

    sysctl -w vm.swappiness=5

    Also, make the change permanent by editing the file /etc/sysctl.conf, edit the line or add a new line at the bottom as: "vm.swappiness = 5".

3.2 Network configuration

1. For each node, disable NetworkManager service and enable network service by the following command.

   chkconfig NetworkManager off; chkconfig network on

2. For the Master Node, in order to get access to Internet through the wireless network card under command line, we need to let the system automatically connect to a wireless router when the system boots. It can be done with following procedures [8]:

   a. Generate the pass phase with the command
   wpa_passphrase SSID PASSWD >> /etc/wpa_supplicant/wpa_supplicant.conf
   Replace SSID and PASSWD with the wireless network name and password.

   b. Writing following content into a new .sh file, here we use the path /boot/startwlan.sh

```
#!/bin/bash
wpa_supplicant -iwlan0 -B -c /etc/wpa_supplicant/wpa_supplicant.conf
dhclient wlan0
```

   c. Edit the file /etc/rc.d/rc.local, add a new line to the bottom as below:

```
sh /boot/startwlan.sh
```

3. Manually assign an IP address, Gateways, etc., for each node by editing the file /etc/sysconfig/network-scripts/ifcfg-eth0. For the cluster network, we set the IP address of the Master Node to be 192.168.101.101, and the Slave Nodes to be 192.168.101.201 [202. 203. 204, 205].

   a. Following is an example of the ifcfg-eth0 file of the Master Node:

```
DEVICE=eth0
HWADDR=**:**:**:**:**:**
TYPE=Ethernet
UUID=********-****-****-****-************
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=192.168.101.101
NETMAST=255.255.255.0
DNS1=8.8.8.8
DNS2=8.8.4.4
```

b. Following is an example of the ifcfg-eth0 of the Slave Nodes:

```
DEVICE=eth0
HWADDR=**:**:**:**:**:**
TYPE=Ethernet
UUID=********-****-****-****-************
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=192.168.101.201
NETMAST=255.255.255.0
GATEWAY=192.168.101.101
DNS1=8.8.8.8
DNS2=8.8.4.4
```

4. Respectively configure the Hosts file located at /etc/hosts, following is an example:

```
192.168.101.101     master1
192.168.101.201     slave1
192.168.101.202     slave2
192.168.101.203     slave3
192.168.101.204     slave4
192.168.101.205     slave5
```

5. On all nodes, SSHD service needs to be enabled, as well as SSH root login. (Keep default settings.)
6. On each Slave Node, the iptables firewall needs to be disabled with the following command.

   /etc/init.d/iptables stop; chkconfig iptables off
7. On the Master Node, the iptables firewall is enabled, and all incoming traffic from public interface (the wireless network card) is denied except through Ports 22 and 7180, which corresponds to SSH service and ClouderaManager webserver service. Also, a forwarding rule needs to be set up with NAT on Master Node so that the Slave nodes can get access to Internet [6]. Save the following content in a .sh file, such as "/boot/forwards.sh".

```
#!/bin/bash
iptables -P INPUT ACCEPT
iptables -F INPUT
iptables -P OUTPUT ACCEPT
iptables -F OUTPUT
iptables -P FORWARD DROP
iptables -F FORWARD
iptables -t nat -F
iptables -A INPUT -i eth0 -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -i wlan0 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i wlan0 -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -i wlan0 -p icmp -m icmp --icmp-type 8 -j ACCEPT
iptables -A INPUT -i wlan0 -p tcp --dport 7180 -j ACCEPT
iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
iptables -A FORWARD -i wlan0 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth0 -o wlan0 -j ACCEPT
iptables -A INPUT -i wlan0 -j DROP
iptables –nvL
/etc/init.d/iptables save
```

And add the "sh /boot/forwards.sh" as a new line to the bottom of the file /etc/rc.d/rc.local, similar as we did earlier. Also, the IP forwarding needs to be enabled by editing the file /etc/sysctl.conf, and changing the line "net.ipv4.ip_forward = 0" to "net.ipv4.ip_forward = 1".

8. Reboot all the nodes to make apply the changes. Check if the Internet connection is available on the Slave Nodes with Ping.
9. After we configured the internet, respectively generate SSH keys for each node using the "ssh-keygen" command, and combine all the public keys together into a "authorized_keys" file. For each node, copy this file to the folder "~/.ssh/" using the scp command.

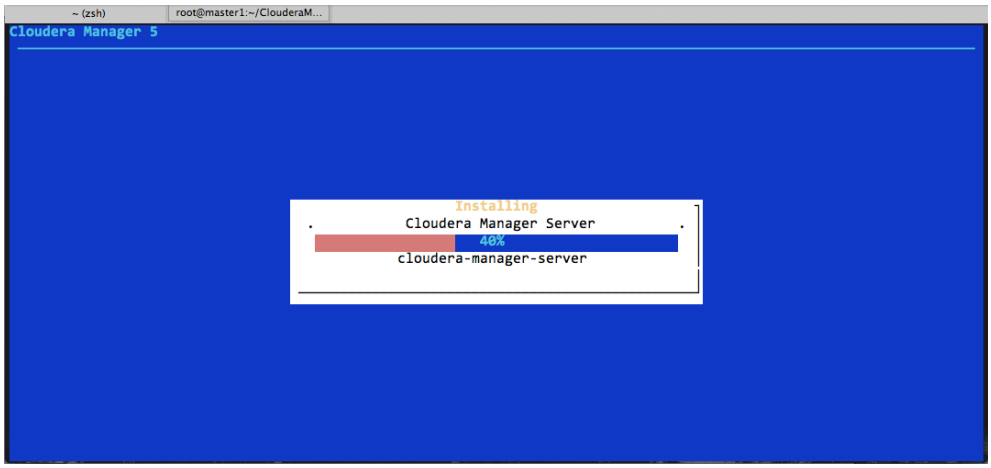
4 Hadoop installation

4.1 Install ClouderaManager on Master Node

On Master Node, download and install ClouderaManager with the commands:

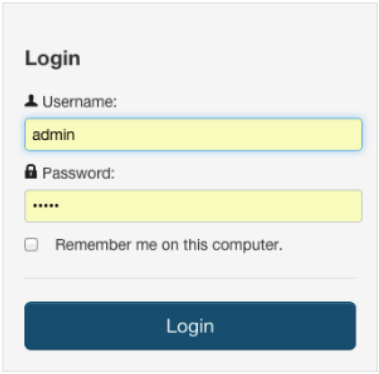
```
wget http://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin;
chmod u+x cloudera-manager-installer.bin; sudo ./cloudera-manager-installer.bin
```

After accepting the Cloudera Express License and Oracle Binary Code License Agreement, the Cloudera Manager 5 installation will start.

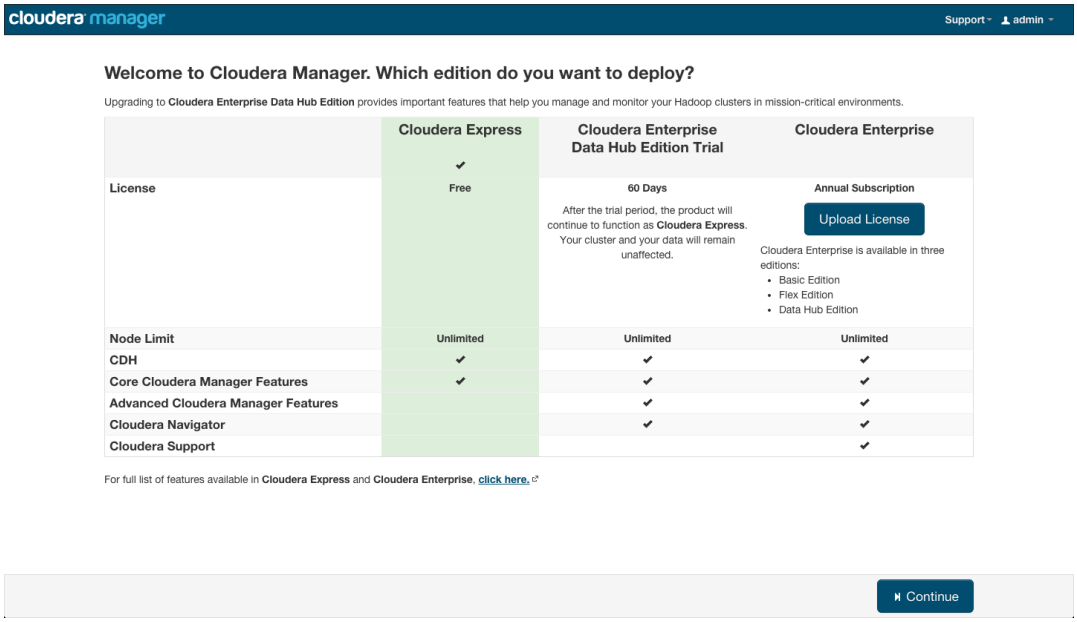


4.2 Install Hadoop on the cluster

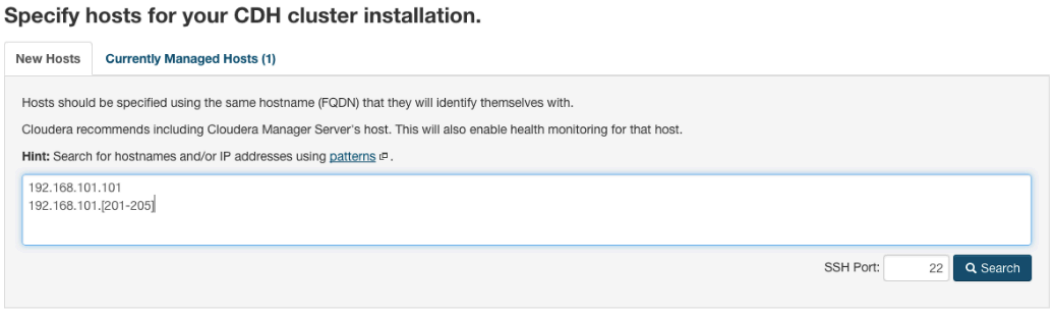
1. If not specified below, we keep the default settings.
2. Use a web browser to install CDH by visiting the master node through port 7180. The default user name and password are both “admin”.



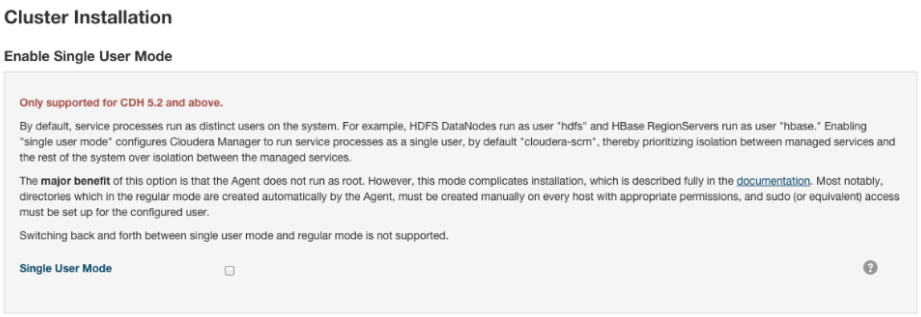
3. In “License” section, select “Cloudera Express” [4] option, since it’s free of charge and it is enough for our 6-node cluster.



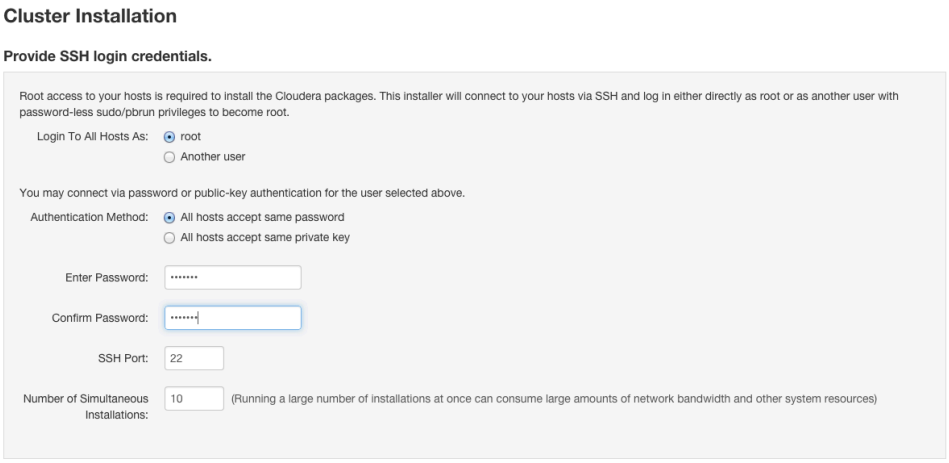
4. First, in the "Specify hosts" section, type in the IP address of each node of the cluster. If you are reinstalling the cluster, the nodes may have already been stored under the "Currently Managed Hosts" tab.



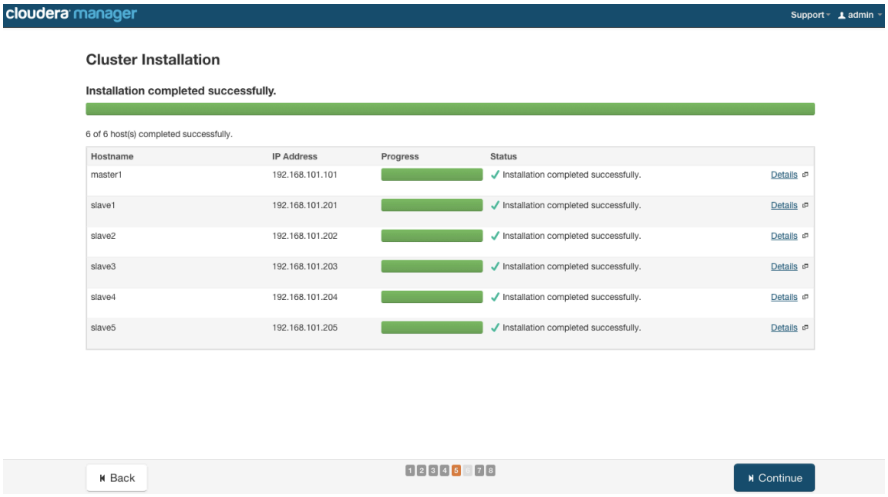
5. The following procedures are separated into two parts, the "Cluster Installation" procedures and the "Cluster Setup" procedures.
6. Under "Cluster Installation", in the "Select Repository" page (Page 1), choose "Use Parcels", and keep the rest with default settings.
7. In "JDK Installation Options" page (Page 2), only check the install JDK box.
8. Under the "Enable Single User Mode" page (Page 3), uncheck (if it is checked) the "Single User Mode" box.

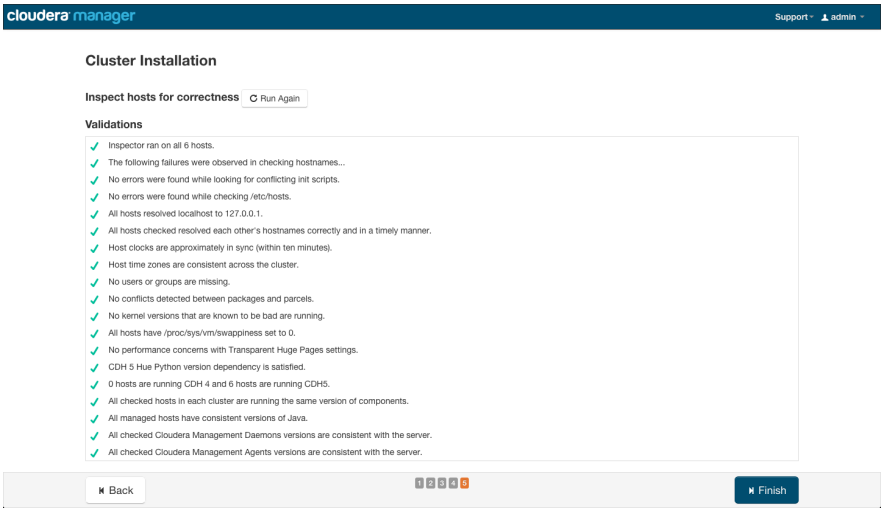


9. In the “Provide SSH login credentials” page (Page 4), type in the root password we set earlier, note that all the six nodes are suggested to use the same root password.

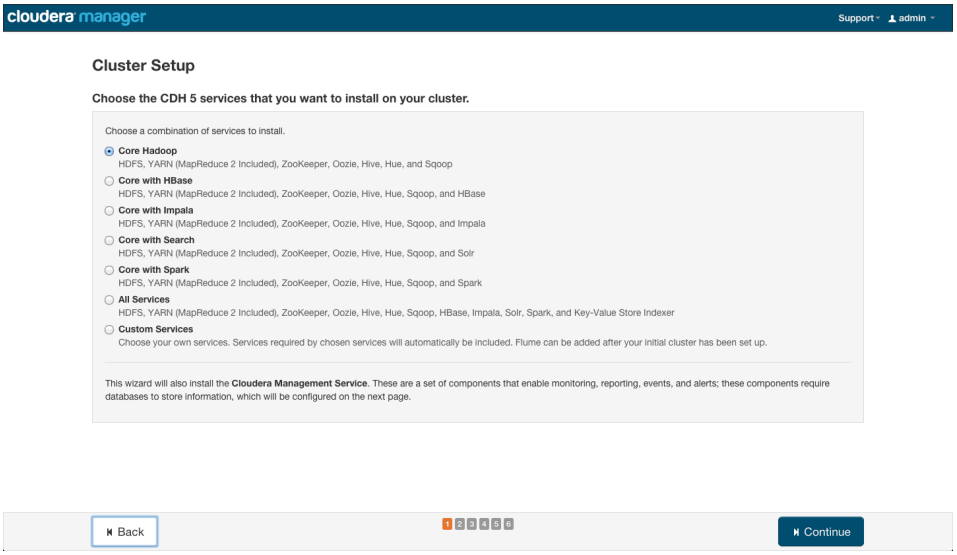


10. Wait the installation to complete; it may take a relatively long time depending on your Internet connection bandwidth. Also it will “Inspect hosts for correctness”, click “Finish” when no error is spotted.

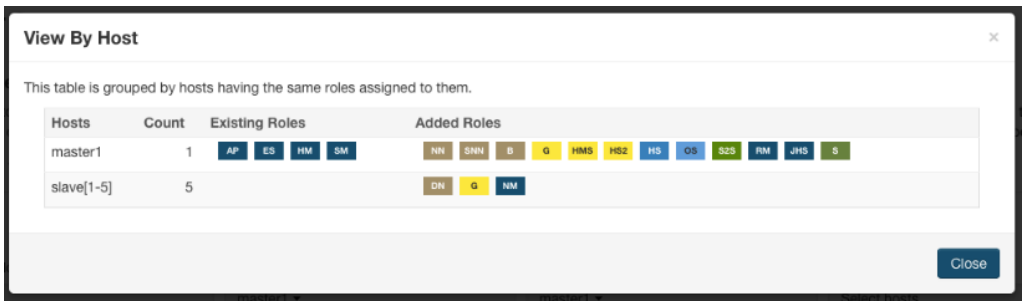




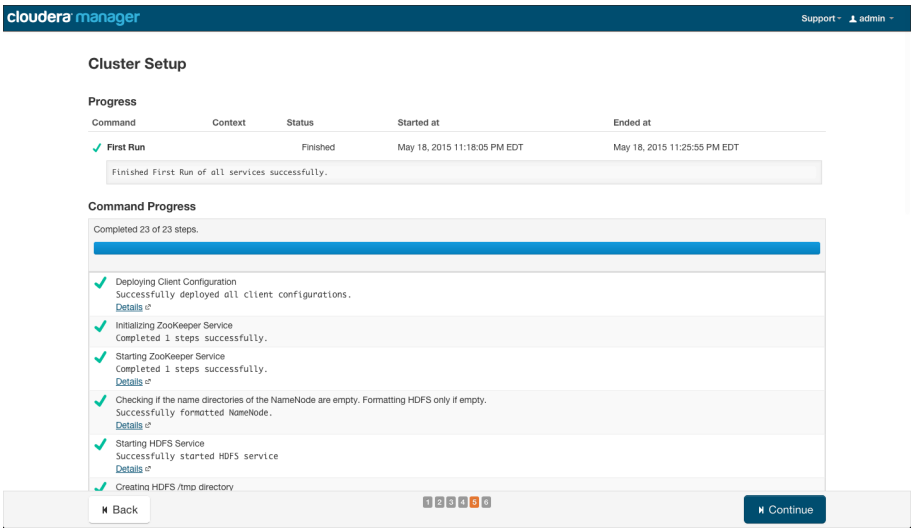
11. Then are the "Cluster Setup" procedures. In the "Choose the CDH 5 services that you want to install on your cluster" page, choose the corresponding combination. Here since we only need to use HDFS and YARN services, we can either choose the "Custom Services" option or just choose "Core Hadoop". If you want to play with other services provided by CDH 5, such as Spark and Solr, you can choose the "All Services" option.



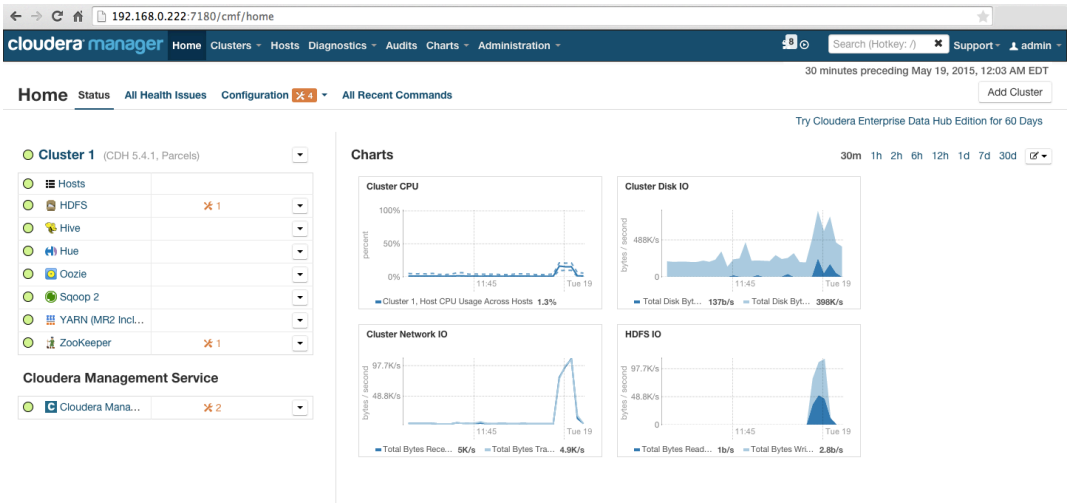
12. Next, we need to assign the role for each node, here we will choose the master node as the NameNode, SecondaryNameNode, and choose the five slave nodes as the Data Nodes. Then manually select the corresponding services for each node. Here is an example.



13. Waiting for the installation to complete. It may take very long time.

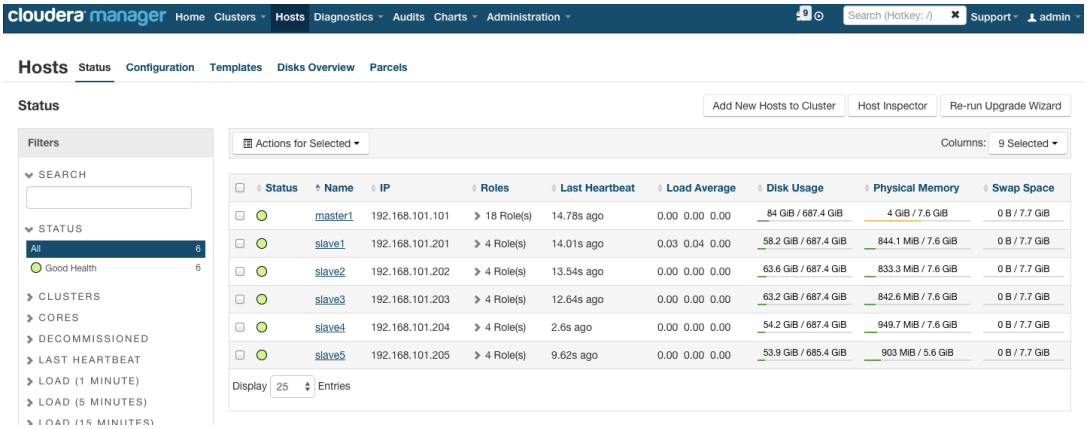


14. When the setup processes complete, login to the Cloudera Manager home page to inspect the cluster.



15. Below is a later snapshot of the host status when we only start the HDFS and YARN services, and have uploaded some data on it.

(Note that, here we added the Spark service to our cluster but we didn't start it. So the slave nodes now have 4 roles and master node has 18 nodes, including the Cloudera Management Services.)



Status	Name	IP	Roles	Last Heartbeat	Load Average	Disk Usage	Physical Memory	Swap Space
●	master1	192.168.101.101	18 Role(s) HDFS Balancer HDFS NameNode HDFS SecondaryNameNode Hive Gateway Hive Metastore Server HiveServer2 Hue Server Cloudera Management Service Alert Publisher Cloudera Management Service Event Server Cloudera Management Service Host Monitor Cloudera Management Service Service Monitor Oozie Server Spark Gateway Spark History Server Sqoop 2 Server YARN (MR2 Included) JobHistory Server YARN (MR2 Included) ResourceManager ZooKeeper Server	13.27s ago	0.12 0.12 0.05	84 GiB / 687.4 GiB	4 GiB / 7.6 GiB	0 B / 7.7 GiB
●	slave1	192.168.101.201	4 Role(s) HDFS DataNode Hive Gateway Spark Gateway YARN (MR2 Included) NodeManager	12.6s ago	0.00 0.00 0.00	58.2 GiB / 687.4 GiB	844 MiB / 7.6 GiB	0 B / 7.7 GiB

5 Testing

Run the PiTest program to test if the installation is successful [5].

> *sudo –u hdfs hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 20 10000*

And here is the running result.

```
[root@master1 /]# sudo -u hdfs hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 20 1
0000
Number of Maps  = 20
Samples per Map = 100000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
        File Input Format Counters
                Bytes Read=2360
        File Output Format Counters
                Bytes Written=97
Job Finished in 26.841 seconds
Estimated value of Pi is 3.14150400000000000000
```

We can see that it passed the test.

6 Result

We've analyzed a human movement dataset, which contains quarter million real-world GPS trajectories, on this cluster, by running Python scripts using Hadoop Streaming [9]. The dataset is about 25.6GB, and it took about 15 minutes to finish running the MapReduce program. Comparing with running a similar program with Open MPI on a Intel Core i7-4790K (quad-core, 4.4 GHz) desktop with four processes, which took about 2 hours, our Hadoop cluster shows great advantages.

7 Summary

In general, our cluster building is successful. Now we can write and run our own MapReduce programs on this homemade cluster, which we have full control. The system is still scalable, and we plan to add one more slave node into the cluster in next couple of days. Also, adding more storage devices is on our agenda. The total cost of the cluster is about $800. On the other hand, the power consumption of this cluster is relative low; it takes about 600 Watts when fully loaded.

But due to the limitation of funding (the equipment is self-funded), we have to use consumer-grade desktops with many second-hand components, and we have to put the cluster at a corner in our apartment, which make the cluster to be somehow unstable. Every now and then, there may be one node goes down due to hardware failure (mainly caused by home moisture). Even though Hadoop provides redundancy for up to 2-node failure, it is still a crucial problem to our cluster. Also, for the master node, we did neither use an independent secondary name node, nor did we use raid1 for storage. Thus, if the master node went down or a hard drive failure happened on the master node, we will lose all of our data. Even though we used an independent hard drive for data backup and temporary storage, the system is still insecure.

The biggest bottleneck of this platform is at the memory side. 8GB is in general not enough for slave nodes, not mention the master node. During programming, we have to modify our MapReduce scripts several times to reduce the memory usage to avoid the “Java heap space errors”.

8 References

[1] The Apache Software Foundation, “Welcome to Apache Hadoop”. https://hadoop.apache.org/, accessed on 05/20/2015.
[2] Cloudera, “Cloudera Manager”, http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise/cloudera-manager.html, accessed on 05/20/2015.
[3] Cloudera, “CDH”, http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html, accessed on 05/20/2015.
[4] Cloudera, “Cloudera Express”, http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-express.html, accessed on 05/20/2015.
[5] Cloudera, “Testing the Installation”, http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cm_ig_testing_the_install.html, accessed on 05/20/2015.
[6] Frank Rischner, “Setting up a Hadoop Cluster with Cloudera Manager and Impala”, https://www.stthomas.edu/media/engineeringsoftware/files/gpspdfs/coe4bd/Rischner.pdf, accessed on 05/20/2015.
[7] Avinash Kumar, “Hadoop, introduction, installation and administration”, https://docs.google.com/file/d/0Bx6N95pJhrROblJiaEJ0dHpwVmc, accessed on 05/20/2015.
[8] BeatriceTorracca, “How to use a WiFi interface”, https://wiki.debian.org/WiFi/HowToUse, accessed on 05/20/2015.
[9] Michael Noll, “Writing an Hadoop MapReduce Program in Python”, http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/, accessed on 05/20/2015.

9 Appendix

Here we attach the python scripts of the mapper and reducer we used in our human mobility project. The programs automatically calculate the displacement distribution from real-world GPS trajectories.

Mapper.py

```
#!/usr/bin/env python
import sys
import math


time_gap_limit = 30  # seconds
ajac_dist_limit = 1000  # meters
stop_v_limit = 1.0  # m/s
traj_len_limit = 50  # points
resolution_limit = 1000.0  # meters
time_between_stops_limit = 600  # seconds
stop_count_condition = 180  # seconds


def distance(lat1, lon1, lat2, lon2):
    R = 6371000
    Haversine = (math.sin(math.radians((lat2 - lat1) / 2)))**2 + \
        math.cos(math.radians(lat1)) * \
        math.cos(math.radians(lat2)) * \
        (math.sin(math.radians((lon2 - lon1) / 2)))**2
    d = R * 2 * math.atan2(math.sqrt(Haversine), math.sqrt(1 - Haversine))
    return d


def aver_position(stop_slice):
    count = len(stop_slice)
    mid_lat, mid_lon = 0, 0
    for latlon in stop_slice:
        mid_lat += latlon[1] / count
        mid_lon += latlon[2] / count
    return [mid_lat, mid_lon, stop_slice[0][0], stop_slice[-1][0],
            (stop_slice[0][0] + stop_slice[-1][0]) * 0.5]


def cal_disp(data_piece):
    indicator_list = []
    for txyi in data_piece:
        indicator_list.append(txyi[3])
    if 2.0 in indicator_list:
        if 1.0 in indicator_list or 0.0 in indicator_list:
            indicator = 2
        else:
            indicator = 1
    else:
        indicator = 0
    start = 0
    stops = []
    stop_indicator = 0
    error_jump = 1
```

```
    ptsnum = len(data_piece)
    for i in xrange(1, ptsnum):
        delta_d = distance(
            data_piece[i - 1][1], data_piece[i - 1][2],
            data_piece[i][1], data_piece[i][2])
        delta_t = data_piece[i][0] - data_piece[i - 1][0]
        if delta_t == 0:
            continue
        v = delta_d / delta_t
        if v < stop_v_limit and v >= 0:
            error_jump = 0
            if stop_indicator == 0:
                start = i - 1
                stop_indicator = 1
        elif not error_jump:
            error_jump = 1
        elif stop_indicator == 1:
            if start == 0:
                midposi = aver_position(data_piece[start: i - 1])
                stops.append(midposi)
            elif data_piece[i][0] - data_piece[start][0] > stop_count_condition:
                midposi = aver_position(data_piece[start: i - 1])
                stops.append(midposi)
            stop_indicator = 0
    if stop_indicator == 1:
        midposi = aver_position(data_piece[start: ptsnum])
        stops.append(midposi)
    d_list = []
    # print stops
    if len(stops) < 2:
        return []
    last_stop = stops[0]
    for i in xrange(1, len(stops)):
        d = distance(
            last_stop[0], last_stop[1], stops[i][0], stops[i][1])
        delta_t = stops[i][2] - last_stop[3]
        if d > resolution_limit and delta_t > time_between_stops_limit:
            d_list.append([d, stops[i][4] - last_stop[4], indicator])
            last_stop = stops[i]
    return d_list


def cut_pieces(data):
    start = 0
    last_t = data[0][0]
    d_list = []
    for i in xrange(1, len(data)):
        if data[i][0] - last_t > time_gap_limit or \
                distance(data[i - 1][1], data[i - 1][2],
                         data[i][1], data[i][2]) > ajac_dist_limit:
            if i - start > traj_len_limit:
                d_list += cal_disp(data[start: i])
            start = i
        last_t = data[i][0]
    if i + 1 - start > traj_len_limit:
        d_list += cal_disp(data[start: i])
    return d_list


```

```
def GPXparse(content):
    content1 = content.split(',')[:-1]
    data = []
    for strtxyi in content1:
        txyi = strtxyi.split(' ')
        data.append(
            [int(txyi[0]), float(txyi[1]), float(txyi[2]), int(txyi[3])])
    if len(data) < traj_len_limit:
        return 1
    d_list = cut_pieces(data)
    if d_list:
        for d_pair in d_list:
            print '0' * (8 - int(math.log10(d_pair[0]))) + \
                '%.8f\t%.2f\t%d' % (d_pair[0], d_pair[1], d_pair[2])
    return 1


def read_input(file):
    for line in file:
        yield line.strip()


def main():
    content = read_input(sys.stdin)
    for line in content:
        GPXparse(line)


if __name__ == '__main__':
    main()
```

The corresponding Reducer.py

```
#!/usr/bin/env python
import sys
import math

maxloglen = 100
indicator_num = 3
logres = 10.0
min_resolution = 1000.0  # min = 1km
base = 10.0  # or use math.e


def read_input(file):
    for line in file:
        yield line.strip()


def main():
    content = read_input(sys.stdin)
    lastline = ''
    distrib = []

    for i in xrange(indicator_num):
        distrib.append([0.0] * maxloglen)

```

```
    for line in content:
        line = line.strip()
        if line == lastline:
            continue
        lastline = line
        content = line.split()
        indicator = int(content[2])
        d = float(content[0])
        d_index = int(math.log(d / min_resolution, base) * logres)
        distrib[indicator][d_index] += 1.0

    for i in xrange(indicator_num):
        count = sum(distrib[i])
        print i
        for j in xrange(maxloglen):
            if distrib[i][j] > 0.5:
                x = math.pow(base, (j + 0.5) / logres)
                p = distrib[i][j] / count / (math.pow(base, (j + 1.0) / logres) -
                                             math.pow(base, float(j) / logres))
                print '%.12f\t%.12f' % (x, p)


if __name__ == '__main__':
    main()
```

And below is the command for running the MapReduce programs, note that, we only use one reducer here. The /GPX_DATA and /GPX_RES_D below are respectively the input path and output path on HDFS.

nohup sudo -u hdfs hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-mr1.jar -D mapred.reduce.tasks=1 -file Mapper.py -mapper Mapper.py -file Reducer.py -reducer Reducer.py -input /GPX_DATA/data.dat -output /GPX_RES_D &