

# Building a 6-node Hadoop Cluster

Xiangwen Wang

July 15, 2015  
Blacksburg, VA

## Table of Contents

<b>1</b>	<b>Project Overview .....</b>	<b>2</b>
<b>2</b>	<b>Hardware installation .....</b>	<b>3</b>
2.1	System design .....	3
2.2	Hardware specification .....	3
<b>3</b>	<b>Cluster installation .....</b>	<b>4</b>
3.1	Operating system installation .....	4
3.2	Network configuration .....	5
<b>4</b>	<b>Hadoop installation .....</b>	<b>8</b>
4.1	Install ClouderaManager on Manager Node .....	8
4.2	Install Hadoop on the cluster .....	8
<b>5</b>	<b>Testing .....</b>	<b>14</b>
<b>6</b>	<b>Result .....</b>	<b>14</b>
<b>7</b>	<b>Summary .....</b>	<b>15</b>
<b>8</b>	<b>References .....</b>	<b>16</b>
<b>9</b>	<b>Appendix .....</b>	<b>17</b>

## 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.

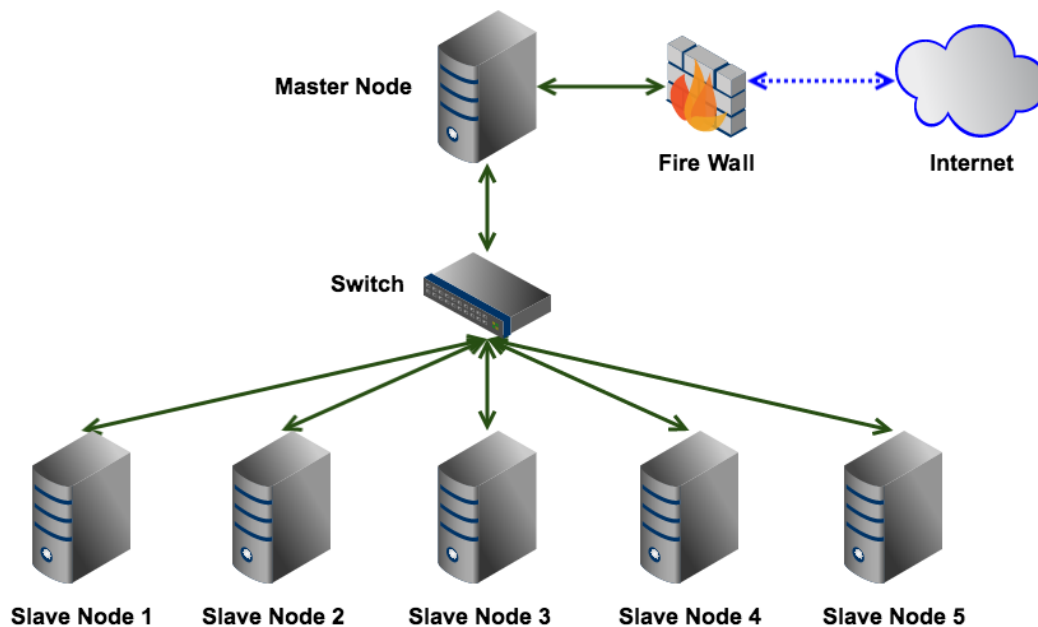
ClouderaManager [2] is a management application provided by Cloudera that could help us easily setup, deploy, and manage CDH (Cloudera Distribution Including Apache Hadoop [3]), an integrated and flexible Hadoop distribution also provided by Cloudera, on our cluster.

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 install 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
NETMASK=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
NETMASK=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.



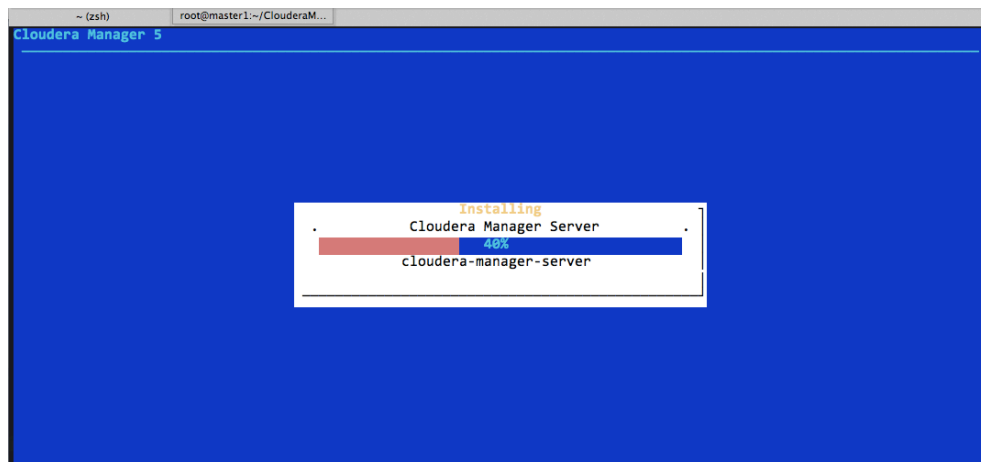
## 4 Hadoop installation

### 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".

A login form for Cloudera Manager. The form has a title "Login". Below the title, there are two input fields: "Username:" and "Password:". The "Username:" field contains the text "admin". The "Password:" field contains six asterisks "\*\*\*\*\*". Below the password field, there is a checkbox labeled "Remember me on this computer." which is currently unchecked. At the bottom of the form, there is a blue button labeled "Login".

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

## Welcome to Cloudera Manager. Which edition do you want to deploy?

Upgrading to Cloudera Enterprise Data Hub Edition provides important features that help you manage and monitor your Hadoop clusters in mission-critical environments.

	Cloudera Express	Cloudera Enterprise Data Hub Edition Trial	Cloudera Enterprise
License	Free	60 Days After the trial period, the product will continue to function as Cloudera Express. Your cluster and your data will remain unaffected.	Annual Subscription <a href="#">Upload License</a> Cloudera Enterprise is available in three editions: • Basic Edition • Flex Edition • Data Hub Edition
Node Limit	Unlimited	Unlimited	Unlimited
CDH	✓	✓	✓
Core Cloudera Manager Features	✓	✓	✓
Advanced Cloudera Manager Features		✓	✓
Cloudera Navigator		✓	✓
Cloudera Support			✓

For full list of features available in Cloudera Express and Cloudera Enterprise, [click here](#).

[Continue](#)

- 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.

### Specify hosts for your CDH cluster installation.

New Hosts

Currently Managed Hosts (1)

Hosts should be specified using the same hostname (FQDN) that they will identify themselves with.

Cloudera recommends including Cloudera Manager Server's host. This will also enable health monitoring for that host.

**Hint:** Search for hostnames and/or IP addresses using [patterns](#).

192.168.101.101

192.168.101.[201-205]

SSH Port:

22

Search

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

## Cluster Installation

### Enable Single User Mode

Only supported for CDH 5.2 and above.

By default, service processes run as distinct users on the system. For example, HDFS DataNodes run as user "hdfs" and HBase RegionServers run as user "hbase." Enabling "single user mode" configures Cloudera Manager to run service processes as a single user, by default "cloudera-scm", thereby prioritizing isolation between managed services and the rest of the system over isolation between the managed services.

The **major benefit** of this option is that the Agent does not run as root. However, this mode complicates installation, which is described fully in the [documentation](#). Most notably, directories which in the regular mode are created automatically by the Agent, must be created manually on every host with appropriate permissions, and sudo (or equivalent) access must be set up for the configured user.

Switching back and forth between single user mode and regular mode is not supported.

Single User Mode

☐

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.

## Cluster Installation

### Provide SSH login credentials.

Root access to your hosts is required to install the Cloudera packages. This installer will connect to your hosts via SSH and log in either directly as root or as another user with password-less sudo/pbrun privileges to become root.

Login To All Hosts As: ☒ root  
☐ Another user

You may connect via password or public-key authentication for the user selected above.

Authentication Method: ☒ All hosts accept same password  
☐ All hosts accept same private key

Enter Password: .....

Confirm Password: .....

SSH Port: 22

Number of Simultaneous Installations: 10 (Running a large number of installations at once can consume large amounts of network bandwidth and other system resources)

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.

**cloudera manager**Support · admin

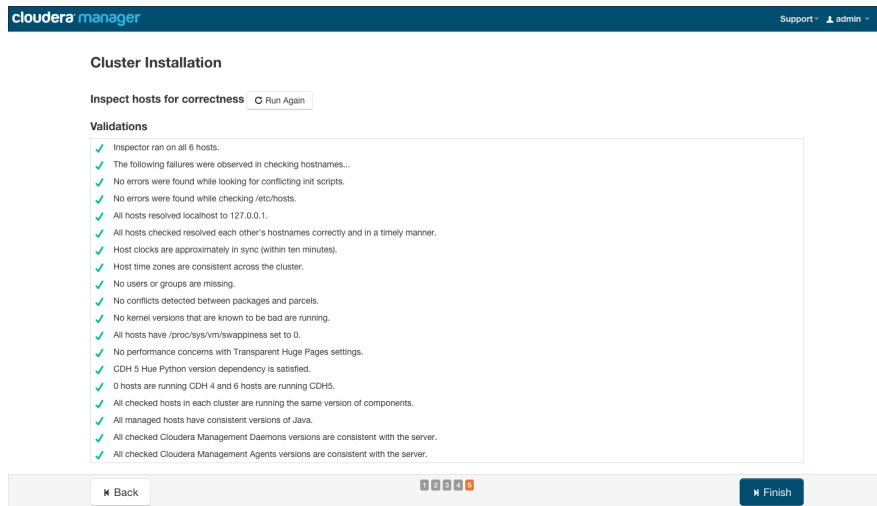
### Cluster Installation

Installation completed successfully.

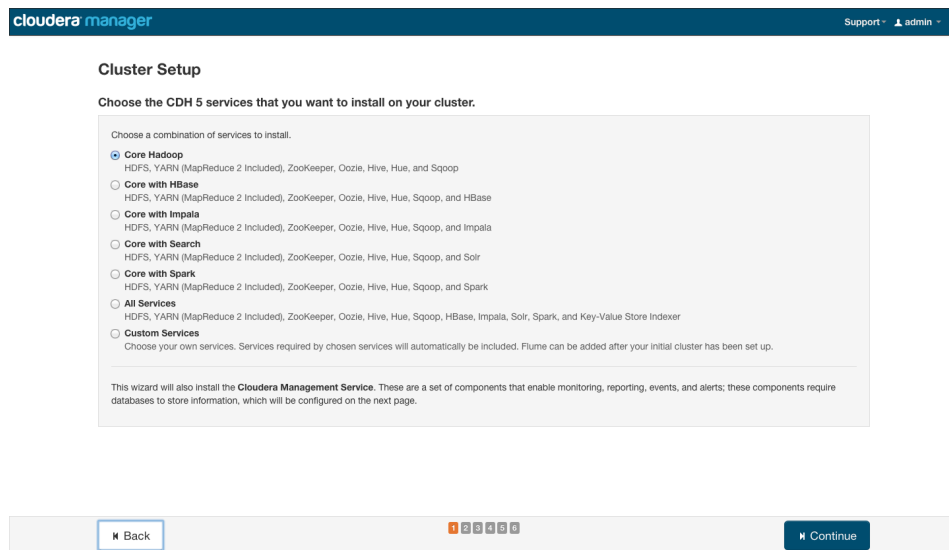
6 of 6 host(s) completed successfully.

Hostname	IP Address	Progress	Status
master1	192.168.101.101	<div></div>	✓ Installation completed successfully. <a href="#">Details</a>
slave1	192.168.101.201	<div></div>	✓ Installation completed successfully. <a href="#">Details</a>
slave2	192.168.101.202	<div></div>	✓ Installation completed successfully. <a href="#">Details</a>
slave3	192.168.101.203	<div></div>	✓ Installation completed successfully. <a href="#">Details</a>
slave4	192.168.101.204	<div></div>	✓ Installation completed successfully. <a href="#">Details</a>
slave5	192.168.101.205	<div></div>	✓ Installation completed successfully. <a href="#">Details</a>

[Back](#)[Continue](#)



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.

**View By Host**

This table is grouped by hosts having the same roles assigned to them.

Hosts	Count	Existing Roles	Added Roles
master1	1	AP ES HM SM	NN SNN B G HMS HSD HS OS S2S RM JHS S
slave[1-5]	5		DN G NM

Close

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

**cloudera manager** Support - admin

### Cluster Setup

**Progress**

Command	Context	Status	Started at	Ended at
✓ First Run		Finished	May 18, 2015 11:18:05 PM EDT	May 18, 2015 11:25:55 PM EDT

Finished First Run of all services successfully.

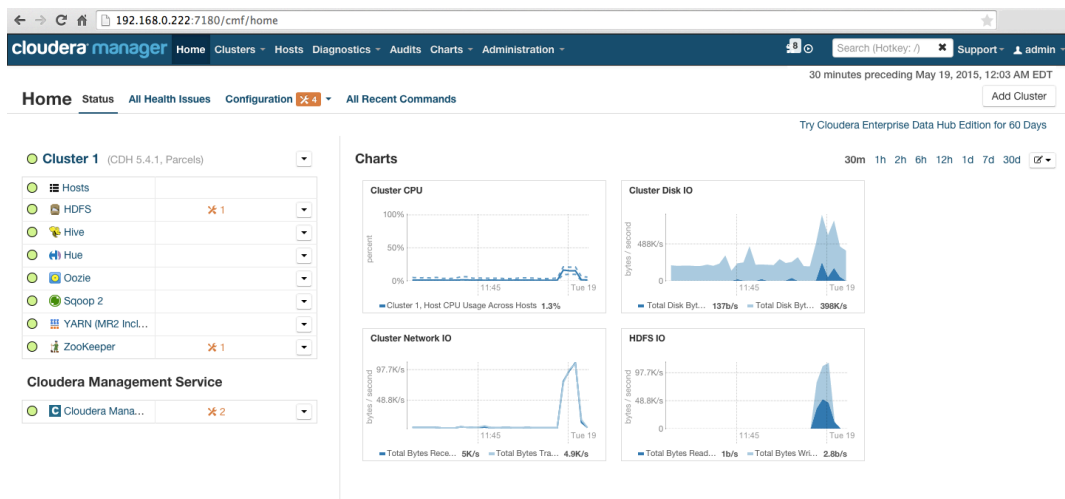
**Command Progress**

Completed 23 of 23 steps.

- ✓ Deploying Client Configuration  
Successfully deployed all client configurations.  
[Details](#)
- ✓ Initializing ZooKeeper Service  
Completed 1 steps successfully.
- ✓ Starting ZooKeeper Service  
Completed 1 steps successfully.  
[Details](#)
- ✓ Checking if the name directories of the NameNode are empty. Formatting HDFS only if empty.  
Successfully formatted NameNode.  
[Details](#)
- ✓ Starting HDFS Service  
Successfully started HDFS service  
[Details](#)
- ✓ Creating HDFS /tmp directory

Back Continue

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.)

cloudera manager Home Clusters Hosts Diagnostics Audits Charts Administration

Search (Hotkey: /) Support admin

Hosts Status Configuration Templates Disks Overview Parcels

Status Add New Hosts to Cluster Host Inspector Re-run Upgrade Wizard

Filters

SEARCH

STATUS

All 6

Good Health

CLUSTERS

CORES

DECOMMISSIONED

LAST HEARTBEAT

LOAD (1 MINUTE)

LOAD (5 MINUTES)

LOAD (15 MINUTES)

Actions for Selected Columns: 9 Selected

Status	Name	IP	Roles	Last Heartbeat	Load Average	Disk Usage	Physical Memory	Swap Space
	master1	192.168.101.101	18 Role(s)	14.78s ago	0.00 0.00 0.00	84 GiB / 687.4 GiB	4 GiB / 7.6 GiB	0 B / 7.7 GiB
	slave1	192.168.101.201	4 Role(s)	14.01s ago	0.03 0.04 0.00	58.2 GiB / 687.4 GiB	844.1 MiB / 7.6 GiB	0 B / 7.7 GiB
	slave2	192.168.101.202	4 Role(s)	13.54s ago	0.00 0.00 0.00	63.6 GiB / 687.4 GiB	833.3 MiB / 7.6 GiB	0 B / 7.7 GiB
	slave3	192.168.101.203	4 Role(s)	12.64s ago	0.00 0.00 0.00	63.2 GiB / 687.4 GiB	842.6 MiB / 7.6 GiB	0 B / 7.7 GiB
	slave4	192.168.101.204	4 Role(s)	2.6s ago	0.00 0.00 0.00	54.2 GiB / 687.4 GiB	949.7 MiB / 7.6 GiB	0 B / 7.7 GiB
	slave5	192.168.101.205	4 Role(s)	9.62s ago	0.00 0.00 0.00	53.9 GiB / 685.4 GiB	903 MiB / 5.6 GiB	0 B / 7.7 GiB

Display 25 Entries

Status	Name	IP	Roles	Last Heartbeat	Load Average	Disk Usage	Physical Memory	Swap Space
	master1	192.168.101.101	18 Role(s) <ul style="list-style-type: none"> <li> HDFS Balancer</li> <li> HDFS NameNode</li> <li> HDFS SecondaryNameNode</li> <li> Hive Gateway</li> <li> Hive Metastore Server</li> <li> HiveServer2</li> <li> Hue Server</li> <li> Cloudera Management Service Alert Publisher</li> <li> Cloudera Management Service Event Server</li> <li> Cloudera Management Service Host Monitor</li> <li> Cloudera Management Service Monitor</li> <li> Oozie Server</li> <li> Spark Gateway</li> <li> Spark History Server</li> <li> Sqoop 2 Server</li> <li> YARN (MR2 Included) JobHistory Server</li> <li> YARN (MR2 Included) ResourceManager</li> <li> ZooKeeper Server</li> </ul>	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) <ul style="list-style-type: none"> <li> HDFS DataNode</li> <li> Hive Gateway</li> <li> Spark Gateway</li> <li> YARN (MR2 Included) NodeManager</li> </ul>	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 10000
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](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

```
1.  #!/usr/bin/env python
2.  import sys
3.  import math
4.
5.
6.  time_gap_limit = 30 # seconds
7.  ajac_dist_limit = 1000 # meters
8.  stop_v_limit = 1.0 # m/s
9.  traj_len_limit = 50 # points
10. resolution_limit = 1000.0 # meters
11. time_between_stops_limit = 600 # seconds
12. stop_count_condition = 180 # seconds
13.
14.
15. def distance(lat1, lon1, lat2, lon2):
16.     R = 6371000
17.     Haversine = (math.sin(math.radians((lat2 - lat1) / 2)))**2 + \
18.         math.cos(math.radians(lat1)) * \
19.         math.cos(math.radians(lat2)) * \
20.         (math.sin(math.radians((lon2 - lon1) / 2)))**2
21.     d = R * 2 * math.atan2(math.sqrt(Haversine), math.sqrt(1 - Haversine))
22.     return d
23.
24.
25. def aver_position(stop_slice):
26.     count = len(stop_slice)
27.     mid_lat, mid_lon = 0, 0
28.     for latlon in stop_slice:
29.         mid_lat += latlon[1] / count
30.         mid_lon += latlon[2] / count
31.     return [mid_lat, mid_lon, stop_slice[0][0], stop_slice[-1][0],
32.         (stop_slice[0][0] + stop_slice[-1][0]) * 0.5]
33.
34.
35. def cal_disp(data_piece):
36.     indicator_list = []
37.     for txyi in data_piece:
38.         indicator_list.append(txyi[3])
39.     if 2.0 in indicator_list:
40.         if 1.0 in indicator_list or 0.0 in indicator_list:
41.             indicator = 2
42.         else:
43.             indicator = 1
44.     else:
45.         indicator = 0
46.     start = 0
47.     stops = []
48.     stop_indicator = 0
49.     error_jump = 1
```

```

50.     ptsnum = len(data_piece)
51.     for i in xrange(1, ptsnum):
52.         delta_d = distance(
53.             data_piece[i - 1][1], data_piece[i - 1][2],
54.             data_piece[i][1], data_piece[i][2])
55.         delta_t = data_piece[i][0] - data_piece[i - 1][0]
56.         if delta_t == 0:
57.             continue
58.         v = delta_d / delta_t
59.         if v < stop_v_limit and v >= 0:
60.             error_jump = 0
61.             if stop_indicator == 0:
62.                 start = i - 1
63.                 stop_indicator = 1
64.             elif not error_jump:
65.                 error_jump = 1
66.             elif stop_indicator == 1:
67.                 if start == 0:
68.                     midposi = aver_position(data_piece[start: i - 1])
69.                     stops.append(midposi)
70.                 elif data_piece[i][0] - data_piece[start][0] > stop_count_condition:
71.                     midposi = aver_position(data_piece[start: i - 1])
72.                     stops.append(midposi)
73.                 stop_indicator = 0
74.             if stop_indicator == 1:
75.                 midposi = aver_position(data_piece[start: ptsnum])
76.                 stops.append(midposi)
77.         d_list = []
78.         # print stops
79.         if len(stops) < 2:
80.             return []
81.         last_stop = stops[0]
82.         for i in xrange(1, len(stops)):
83.             d = distance(
84.                 last_stop[0], last_stop[1], stops[i][0], stops[i][1])
85.             delta_t = stops[i][2] - last_stop[3]
86.             if d > resolution_limit and delta_t > time_between_stops_limit:
87.                 d_list.append([d, stops[i][4] - last_stop[4], indicator])
88.                 last_stop = stops[i]
89.         return d_list
90.
91.
92. def cut_pieces(data):
93.     start = 0
94.     last_t = data[0][0]
95.     d_list = []
96.     for i in xrange(1, len(data)):
97.         if data[i][0] - last_t > time_gap_limit or \
98.            distance(data[i - 1][1], data[i - 1][2],
99.                    data[i][1], data[i][2]) > ajac_dist_limit:
100.             if i - start > traj_len_limit:
101.                 d_list += cal_disp(data[start: i])
102.                 start = i
103.             last_t = data[i][0]
104.         if i + 1 - start > traj_len_limit:
105.             d_list += cal_disp(data[start: i])
106.     return d_list
107.
108.

```

```

109. def GPXparse(content):
110.     content1 = content.split(',')[::-1]
111.     data = []
112.     for strtxyi in content1:
113.         txyi = strtxyi.split(' ')
114.         data.append(
115.             [int(txyi[0]), float(txyi[1]), float(txyi[2]), int(txyi[3])])
116.     if len(data) < traj_len_limit:
117.         return 1
118.     d_list = cut_pieces(data)
119.     if d_list:
120.         for d_pair in d_list:
121.             print '0' * (8 - int(math.log10(d_pair[0]))) + \
122.                 '%.8f\t%.2f\t%d' % (d_pair[0], d_pair[1], d_pair[2])
123.     return 1
124.
125.
126. def read_input(file):
127.     for line in file:
128.         yield line.strip()
129.
130.
131. def main():
132.     content = read_input(sys.stdin)
133.     for line in content:
134.         GPXparse(line)
135.
136.
137. if __name__ == '__main__':
138.     main()

```

The corresponding Reducer.py

```

1.  #!/usr/bin/env python
2.  import sys
3.  import math
4.
5.  maxloglen = 100
6.  indicator_num = 3
7.  logres = 10.0
8.  min_resolution = 1000.0 # min = 1km
9.  base = 10.0 # or use math.e
10.
11.
12. def read_input(file):
13.     for line in file:
14.         yield line.strip()
15.
16.
17. def main():
18.     content = read_input(sys.stdin)
19.     lastline = ''
20.     distrib = []
21.
22.     for i in xrange(indicator_num):
23.         distrib.append([0.0] * maxloglen)
24.

```

```

25.     for line in content:
26.         line = line.strip()
27.         if line == lastline:
28.             continue
29.         lastline = line
30.         content = line.split()
31.         indicator = int(content[2])
32.         d = float(content[0])
33.         d_index = int(math.log(d / min_resolution, base) * logres)
34.         distrib[indicator][d_index] += 1.0
35.
36.     for i in xrange(indicator_num):
37.         count = sum(distrib[i])
38.         print i
39.         for j in xrange(maxloglen):
40.             if distrib[i][j] > 0.5:
41.                 x = math.pow(base, (j + 0.5) / logres)
42.                 p = distrib[i][j] / count / (math.pow(base, (j + 1.0) / logres) -
43.                                                math.pow(base, float(j) / logres))
44.                 print '%.12f\t%.12f' % (x, p)
45.
46.
47. if __name__ == '__main__':
48.     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 &

```