CS 553 Cloud Computing

Programming Assignment 2

Boyang Li (A20314367)
Xingtan Hu (A20304622)
Zichen Zhang (A20307812)

Team Member's Contributions

1. Virtual Cluster (1 node, 16 nodes) :	Boyang Li, Xingtan Hu
2. Shared-Memory WordCount :	Boyang Li
3. Hadoop Configuration :	Zichen Zhang
4. Hadoop WordCount :	Zichen Zhang
5. Swift Configuration :	Xingtan Hu
6. Swift WordCount :	Boyang Li, Xingtan Hu
7. Performance Analysis:	Boyang Li, Xingtan Hu, Zichen Zhang
8. Sort on Shared-Memory, Hadoop, Swift, and MPI :	Boyang Li, Xingtan Hu, Zichen Zhang
9. Report:	Xingtan Hu

Brief Description

This programming assignment covers the WordCount application implemented in 3 different ways: Python, Hadoop, and Swift. We count the appearance of every words and count the appearance by using a 10 GB file. And the result txt is sorted by its appearance time.

Methodology

For Shared-Memory WordCount, we use python and multi-thread technology to implement.

For Hadoop WordCount, we use Java based on a Hadoop framework and MapReduce.

For Swift WordCount, we use Swift language and MapReduce.

Runtime Environment Settings

Amazon Web Services EC2
AMI - Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-3d50120d
Hadoop 1.2.1
Swift 0.95 RC6
Java 1.7.0_65
Shared-Memory:
EC2 instance type : c3-large
Hadoop part :
EC2 instance type : (master node)c3-large (slave node)m3-medium
Swift part :
EC2 instance type : (headnode)c3-large (worker node)m3-medium
MPI : Open MPI 3.0.4, starcluster

Installation Steps of Virtual Cluster

1. 1 node
1) log in to the AWS, choose EC2
2) launch instance
3) choose Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-3d50120d
4) choose t2.medium
5) choose spot instance
6) add storage to 12GB(enough for Hadoop and Swift WordCount)
7) create a new security group, with rules: SSH, All TCP(Anywhere), All ICMP(Anywhere)

2. 16 nodes
1) finish the step of 1 node
2) create an image of instance established in 1 node
3) create another 15 nodes use the AMI just created
4) use the same configuration

3. difficulties
At first, we didn't notice that the convenient of AMI, we configure the same environment again and again for 16 nodes.
After we used the AMI, it saved a lot of times.

Hadoop Configuration

Edit /etc/hosts file, put the following line in the file:
172.31.44.150 me1
172.31.44.149 me2
172.31.44.151 me3
172.31.44.197 me4
172.31.44.198 me5
172.31.44.208 me6
172.31.44.209 me7
172.31.44.206 me8
172.31.44.204 me9
172.31.44.199 me10
172.31.44.201 me11
172.31.44.205 me12
172.31.44.207 me13
172.31.44.200 me14
172.31.44.202 me15
172.31.44.203 me16
172.31.46.204 me17
Front is private IP, behind is node name.

For every mechine, edit ~/.ssh/config file, put the following line in the file:
Host me1
    HostName 54.68.149.102
    User ubuntu
    IdentityFile ~/CS553-HW2.pem

```
Host me2
   HostName 54.68.42.118
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me3
   HostName 54.69.147.176
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me4
   HostName 54.69.245.188
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me5
   HostName 54.187.56.197
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me6
   HostName 54.191.224.38
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me7
   HostName 54.191.224.164
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me8
   HostName 54.191.224.66
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me9
   HostName 54.191.224.167
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me10
   HostName 54.191.224.39
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me11
   HostName 54.191.224.142
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me12
   HostName 54.191.224.41
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me13
   HostName 54.191.224.33
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me14
```

```
   HostName 54.191.200.150
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me15
   HostName 54.69.122.172
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me16
   HostName 54.191.224.93
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
Host me17
   HostName 54.191.223.192
   User ubuntu
   IdentityFile ~/CS553-HW2.pem
```

HostName is the mechina public IP, IdentityFile is the key, here we use Amazon key pair.

After doing this, all mechines can ssh each other easily.

masters:
this file defines on which machines Hadoop will start secondary NameNodes in our multi-node cluster.

slaves:
this file lists the hosts, one per line, where the Hadoop slave daemons (DataNodes and TaskTrackers) will be run.

core-site.xml
The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.

Add the following snippets between the <configuration> ... </configuration> tags in the respective configuration XML file:

```
 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>
```

mapred-site.xml
The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map
and reduce task.

```
 <property>
 <name>mapred.job.tracker</name>
 <value>localhost:54311</value>
 <description>The host and port that the MapReduce job tracker runs
 at.  If "local", then jobs are run in-process as a single map
 and reduce task.
 </description>
</property>
```

hdfs-site.xml
Default block replication.
The actual number of replications can be specified when the file is created.
The default is used if replication is not specified in create time.

```
 <property>
 <name>dfs.replication</name>
 <value>1</value>
 <description>Default block replication.
 The actual number of replications can be specified when the file is created.
 The default is used if replication is not specified in create time.
 </description>
</property>
```

From 1 to 16 nodes, copy all three files (core-site.xml, mapred-site.xml, hdfs-site.xml) in the file /hadoop-1.2.1/conf/ in all nodes.

For master node, put mechina name in the file (me1), this is the master node. (only need to modify master node)

For slave node, put the slave nodes that you want in the slaves file. If you want 4 slaves, then put four mechine names in the file. (only need to modify master node)

Formatting hdfs: ~/hadoop-1.2.1/bin/hadoop namenode -format

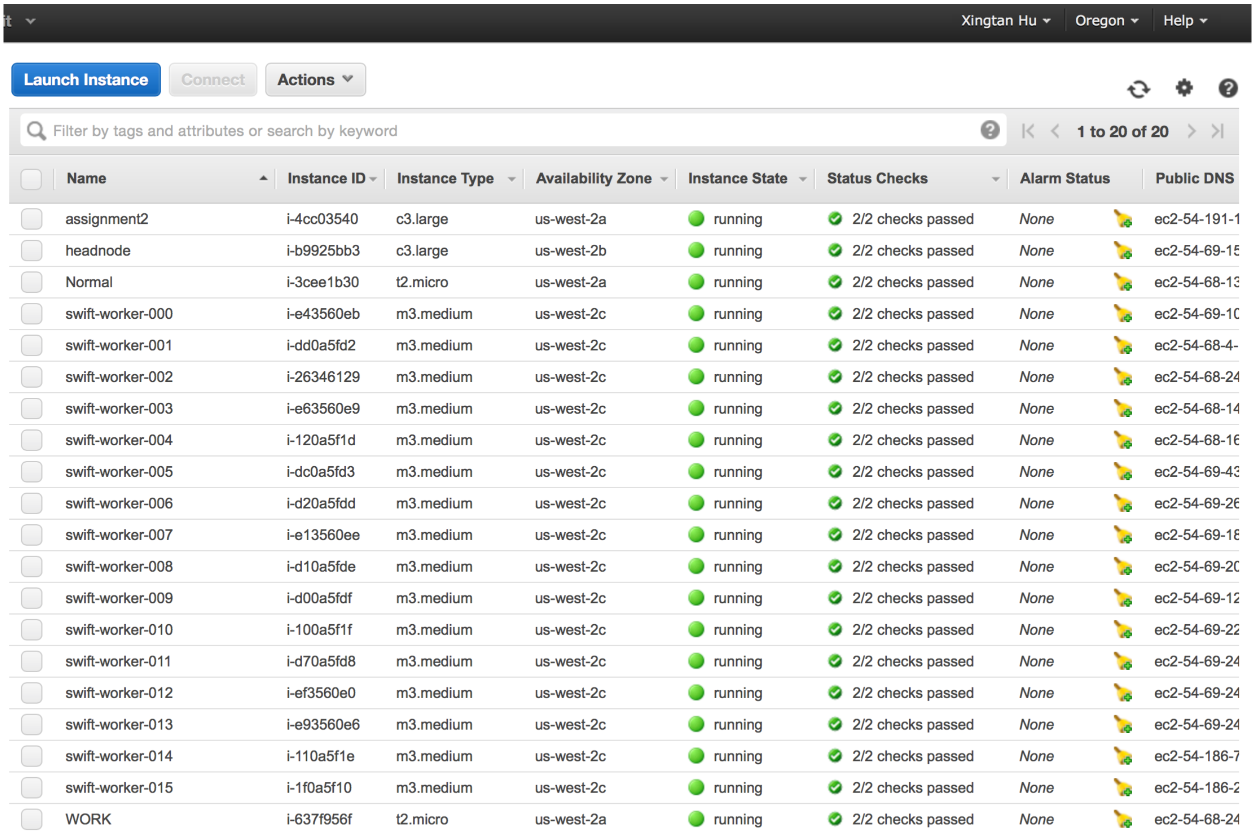
start cluster: ~/hadoop-1.2.1/bin/start-all.sh

create input file: bin/hadoop dfs -mkdir /zzc/input

copy dataset in the file: bin/hadoop dfs -copyFromLocal /home/ubuntu/tmp/wiki10gb /zzc/input run wordcount: bin/hadoop jar wordcount.jar org.wordcount/ZzcWordCount /zzc/input /zzc/output

---

Swift Set&Configuration



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
assignment2	i-4cc03540	c3.large	us-west-2a	running	2/2 checks passed	None	ec2-54-191-1
headnode	i-b9925bb3	c3.large	us-west-2b	running	2/2 checks passed	None	ec2-54-69-15
Normal	i-3cee1b30	t2.micro	us-west-2a	running	2/2 checks passed	None	ec2-54-68-13
swift-worker-000	i-e43560eb	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-10
swift-worker-001	i-dd0a5fd2	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-68-4-
swift-worker-002	i-26346129	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-68-24
swift-worker-003	i-e63560e9	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-68-14
swift-worker-004	i-120a5f1d	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-68-16
swift-worker-005	i-dc0a5fd3	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-43
swift-worker-006	i-d20a5fdd	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-26
swift-worker-007	i-e13560ee	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-18
swift-worker-008	i-d10a5fde	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-20
swift-worker-009	i-d00a5fdf	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-12
swift-worker-010	i-100a5f1f	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-22
swift-worker-011	i-d70a5fd8	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-24
swift-worker-012	i-ef3560e0	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-24
swift-worker-013	i-e93560e6	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-69-24
swift-worker-014	i-110a5f1e	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-186-7
swift-worker-015	i-1f0a5f10	m3.medium	us-west-2c	running	2/2 checks passed	None	ec2-54-186-2
WORK	i-637f956f	t2.micro	us-west-2a	running	2/2 checks passed	None	ec2-54-68-24

You can find all the config files in all submitted file by using ReadMe.

MPI Configuration

```
>>> Configuring SGE...
>>> Configuring NFS exports path(s):
/opt/sge6
>>> Mounting all NFS export path(s) on 1 worker node(s)
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Setting up NFS took 0.088 mins
>>> Installing Sun Grid Engine...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Creating SGE parallel environment 'orte'
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Adding parallel environment 'orte' to queue 'all.q'
>>> Configuring cluster took 1.943 mins
>>> Starting cluster took 3.751 mins

The cluster is now ready to use. To login to the master node
as root, run:

    $ starcluster sshmaster mpicluster

If you're having issues with the cluster you can reboot the
instances and completely reconfigure the cluster from
scratch using:

    $ starcluster restart mpicluster

When you're finished using the cluster and wish to terminate
it and stop paying for service:

    $ starcluster terminate mpicluster

Alternatively, if the cluster uses EBS instances, you can
use the 'stop' command to shutdown all nodes and put them
into a 'stopped' state preserving the EBS volumes backing
the nodes:

    $ starcluster stop mpicluster

WARNING: Any data stored in ephemeral storage (usually /mnt)
will be lost!

You can activate a 'stopped' cluster by passing the -x
option to the 'start' command:

    $ starcluster start -x mpicluster

This will start all 'stopped' nodes and reconfigure the
cluster.
ubuntu@ip-172-31-40-35:~$
```

Data Analysis

1. Performance

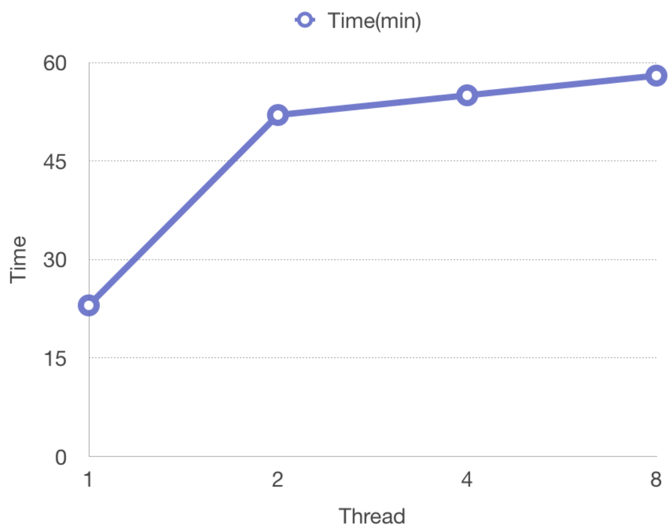
1) Shared-Memory WordCount Performance

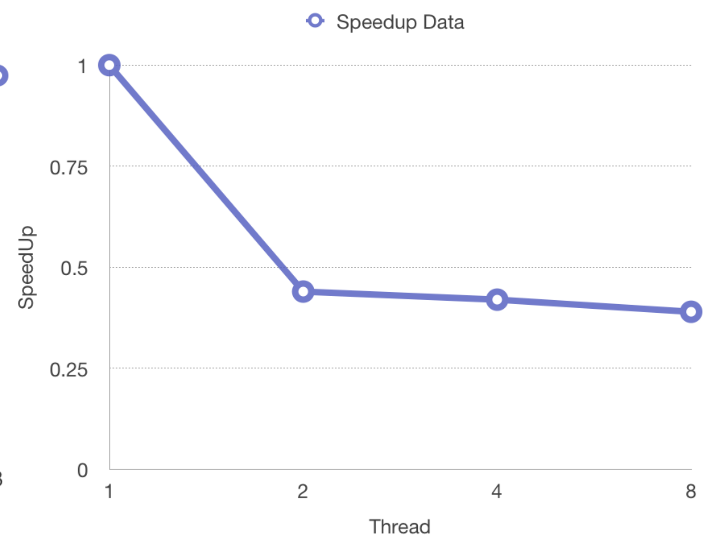
Shared-Memory WordCount Performance

THREAD NO.	TIME(MIN)
1	23
2	52
4	55
8	58

Shared-Memory SpeedUp Data

THREAD NO.	SPEEDUP
1	1
2	0.44
4	0.42
8	0.39



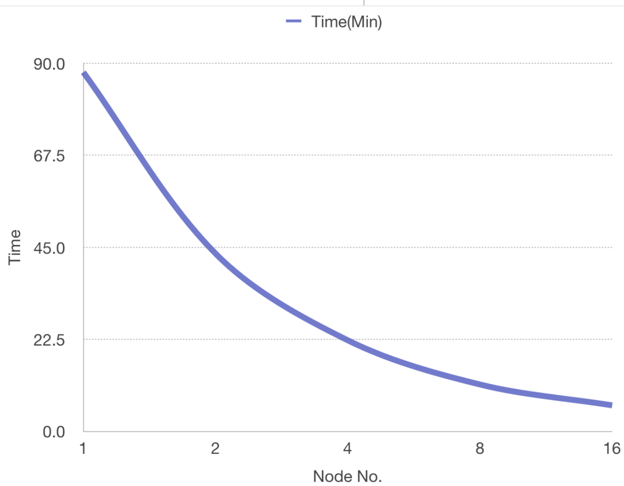


The speedup of shared-memory wordcount is less than 1, which means with more than 1 thread, the program execute slower. The execution-time graph reveals the same thing as speedup graph, that is more threads running concurrently will delay the execution time. The reason is that all threads have to read the file from the disk, which will cause a competition situation, so all these I/O operation will definitely delay the execution time.

2) Hadoop WordCount Performance

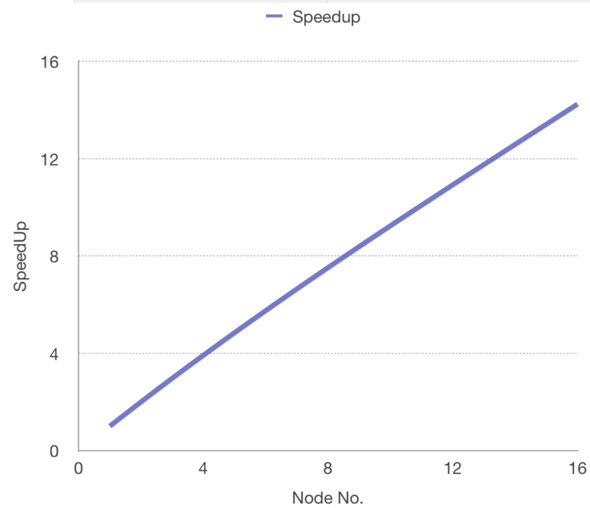
Hadoop WordCount Performance

NODE NO.	TIME
1	87 min 50 sec
2	43 min 26 sec
4	22 min 9 sec
8	11 min 32 sec
16	6 min 10 sec



Hadoop WordCount SpeedUp Data

Node No.	Speedup
1	1
2	2.01
4	3.97
8	7.62
16	14.24



According to the execution-time graph and SpeedUp graph, the Hadoop wordcount program is quite obeying the practical phenomenon. When we run the Hadoop wordcount with 1 or 2 threads, the execution time is quite large. Hadoop needs to set up and configure many things but this level of concurrency is not able to make up the time consumption. However, when number of threads become larger, the execution time becomes much less because high level of concurrency.

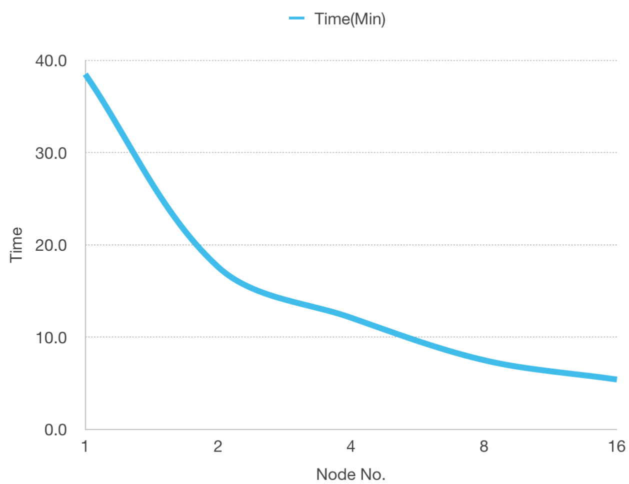
3) Swift WordCount Performance

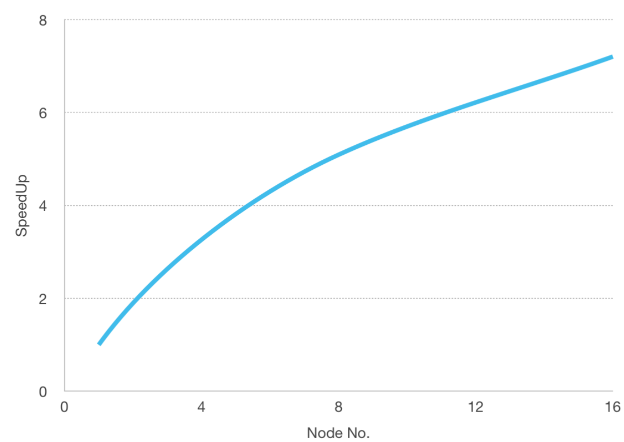
Swift WordCount Performance

NODE NO.	TIME
1	38 min 24 sec
2	17 min 41 sec
4	12 min 6 sec
8	7 min 32 sec
16	5 min 20 sec

Swift WordCount SpeedUp Data

Node No.	Speedup
1	1
2	2.02
4	3.17
8	5.09
16	7.20





The performances of swift and Hadoop programs are quite familiar. They both map and reduce the file automatically. So when the concurrency level is low, the running time could be large because many configurations to be done. But as the concurrency level becoming higher, the running time will reduce significantly because the amortized time on each thread is small with more nodes.

2. Comparison of Three Methods(Shared-Memory, Swift, Hadoop WordCount)

Performance Comparison

THREAD(NODE) NO.	SHARED-MEMORY TIME(MIN)	HADOOP TIME(MIN)	SWIFT TIME(MIN)
1	23	87 min 50 sec	38 min 24 sec
16(node only for hadoop & swift)	N/A	6 min 10 sec	5 min 20 sec

For 1 thread, shared-memory takes less time than Hadoop and Swift. As we mentioned above, shared-memory has the highest efficiency.
For 16 threads, swift has a better performance than Hadoop.
From all the performance data we collected, Swift(16 nodes) has the best performance, which compared to Hadoop(16 nodes) and shared-memory(1 thread).

Conclusion

After several weeks' hard-work, we finally get used to use Hadoop and Swift. After taking long time configuration, we can implement MapReduce on AWS EC2 cluster by using Hadoop and Swift. Through the comparison of performance, we can find a balance between time and cost on build virtual cluster, which makes calculations more effective.