INFO7250 Finial Project

Access Log Analyzer

by Bin Shi

Problem Statement

- API requests from mobile app and browser add-on (Oct, 2017)
- How to start to work on huge amount of data?
- How many visits each day?
- What is the busiest hour?
- Which countries does the traffic come from?
- Which is top 10 URL categories in US?
- What is the trend of the traffic looks like in total, or by country?

Data Set

- 200G (93 files, 2G each, AWS S3)
- **190.239.213.115** - - **[01/Oct/2017:00:00:00 +0000]**
  "GET /axis2/services/WebFilteringService/getCategoryByUrl ?
  app=chrome_antiporn&ver=0.19.7.1
  &**url=https%3A//www.facebook.com**/%3Fstype%3Dlo%26jlou%3DAffAmShI68yNsw-M1-lsS95fsGkzzVgUjyfrS0wKpqjYU_CeCg9VA46WrDXqkYa_nBNdZ9Lx4YOFp0Z8wD_Py2NpH4f1TyNIowTiyRhzZ9lNng%26smuh%3D21435%26lh%3DAc_RQNgTl6mXYKuA
  &**cat=social-networking**
  HTTP/1.1" 200 133 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36"

How to start to work on huge amount of data?

- Random Sampler

```
public static class TheMapper extends Mapper<Object, Text, NullWritable, Text> {

    private static double rate = 0.01; // default value
    private static Random random = new Random();

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        rate = context.getConfiguration().getDouble("rate", rate); // use '-D rate=0.02' to change
        rate = rate > 1.0? rate/100: rate;
    }

    @Override
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        if (random.nextDouble() < rate && value.toString().contains("GET /axis2/")) { // valid log entry
            context.write(NullWritable.get(), value);
        }
    }
}
```

15 files => 1395 (15*93) | 1.3M

```
$ fs -ls -h access-log/sample/subsample
Found 17 items
-rw-r--r--   1 ubuntu supergroup        0 2018-04-26 16:42 access-log/sample/subsample/_SUCCESS
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00000
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00001
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00002
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00003
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00004
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00005
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:41 access-log/sample/subsample/part-m-00006
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00007
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00008
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00009
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00010
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00011
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00012
-rw-r--r--   1 ubuntu supergroup      1.3 M 2018-04-26 16:42 access-log/sample/subsample/part-m-00013
-rw-r--r--   1 ubuntu supergroup    125.7 K 2018-04-26 16:42 access-log/sample/subsample/part-m-00014
-rw-r--r--   1 ubuntu supergroup       30 2018-04-26 16:42 access-log/sample/subsample/part-m-00015
```

InputFormat | Merge ('fs -putmerge' or Reducer)

```
public static class TheReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

    @Override
    public void reduce(NullWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        for (Text v: values) {
            context.write(NullWritable.get(), v);
        }
    }
}
```

$ fs -ls -h access-log/sample/merge

Found 2 items

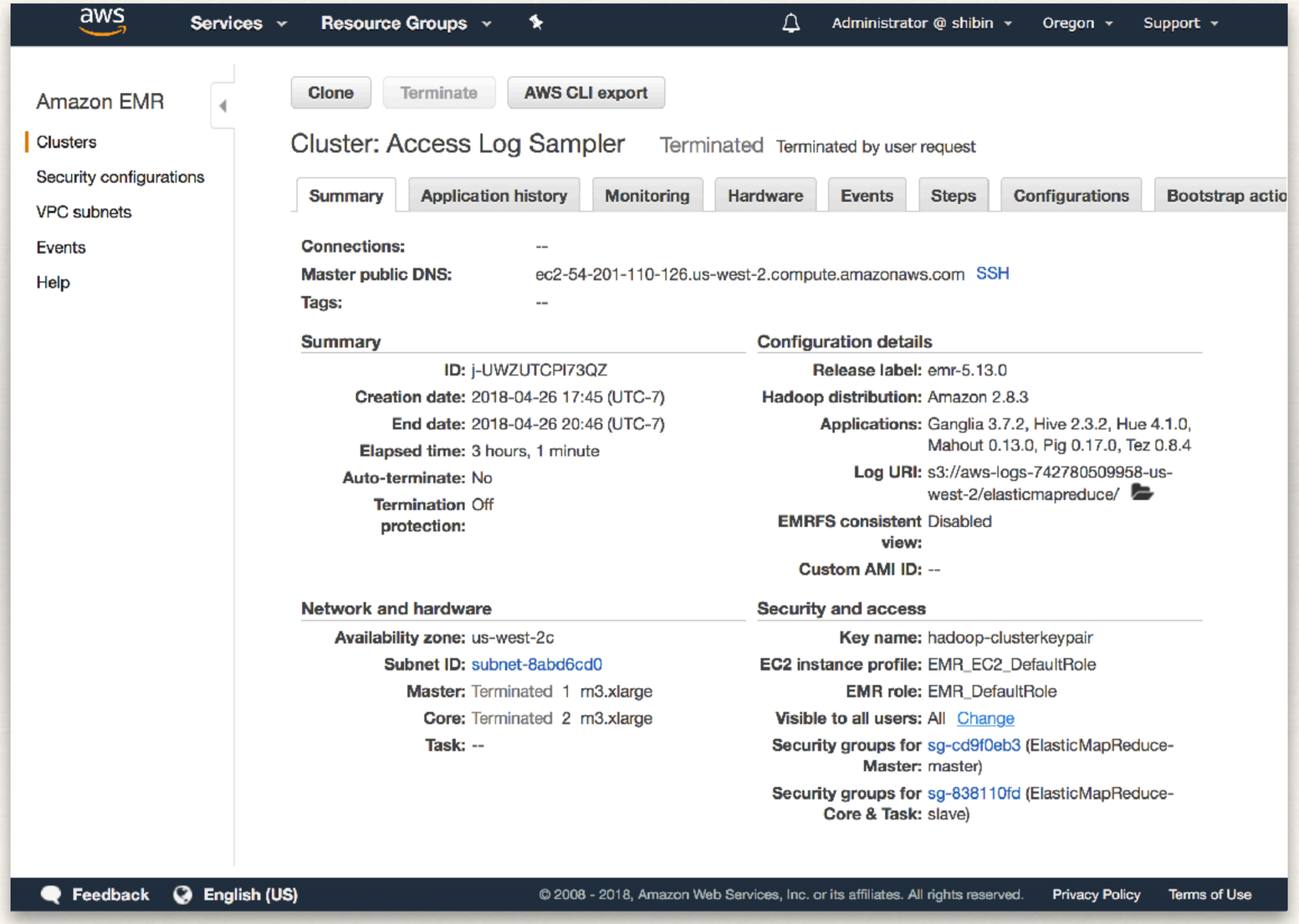
-rw-r--r-- 1 ubuntu supergroup 0 2018-04-26 17:38 access-log/sample/merge/_SUCCESS

-rw-r--r-- 1 ubuntu supergroup **18.1 M** 2018-04-26 17:38 access-log/sample/merge/part-r-00000

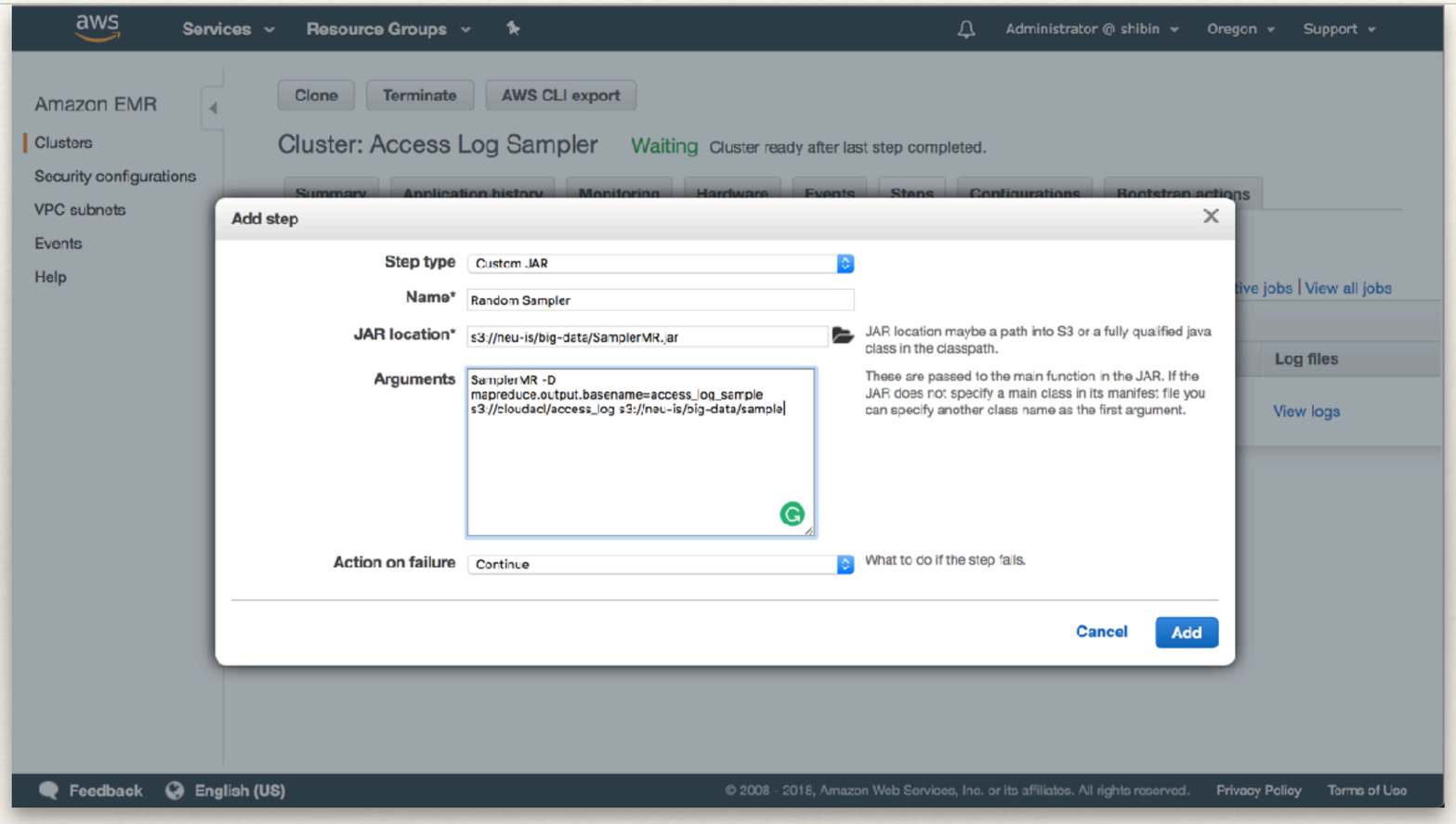
Getting real on AWS EMR

- Master: 1 x m3.xlarge
- Core: 2 x m3.xlarge

- EC2 m3.xlarge:
  - vCPU: 8
  - 16G RAM
  - 80G SSD
  - Network: Up to 10G



Let Hadoop Cluster to Work



- JAR location : s3://neu-is/big-data/SamplerMR.jar
- Arguments : SamplerMR -D mapreduce.output.basename=access_log_sample s3://cloudacl/access_log s3://neu-is/big-data/sample

Working…

- 3178 tasks
- 2.3 hours

Cluster: Access Log Sampler Waiting Cluster ready after last step completed.

Summary | Application history | Monitoring | Hardware | Events | Steps | Configurations | Bootstrap actions

Add step | Clone step | Cancel step

Steps > Jobs > Tasks

View all interactive jobs | View all jobs

Tasks for: s-27FS39UK6QMDJ, Job 1524790101458_0001

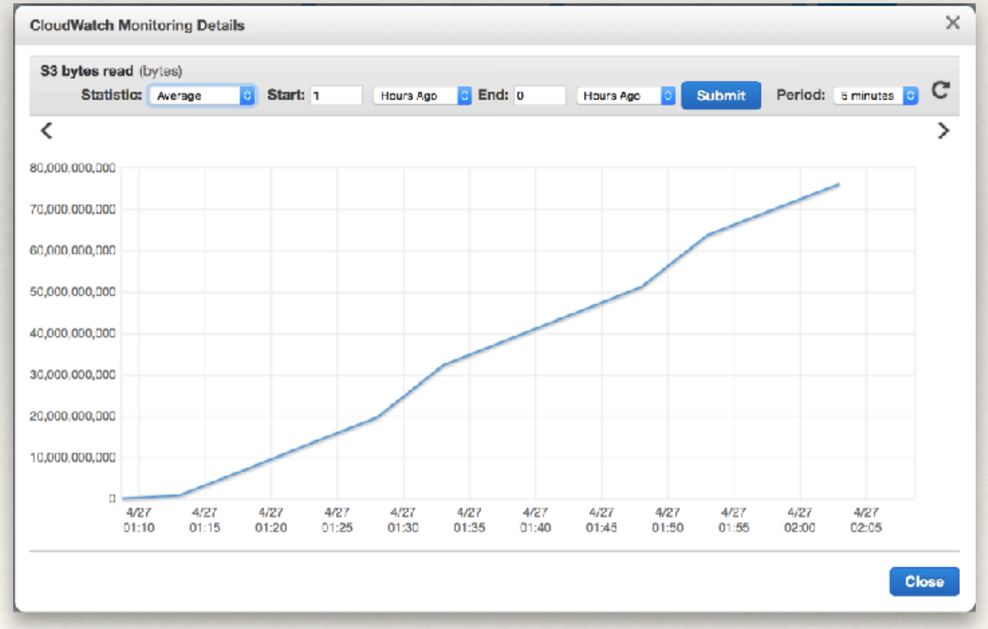
Task summary: 3178 total tasks - 3178 completed, 0 running, 0 failed, 0 pending, 0 cancelled.

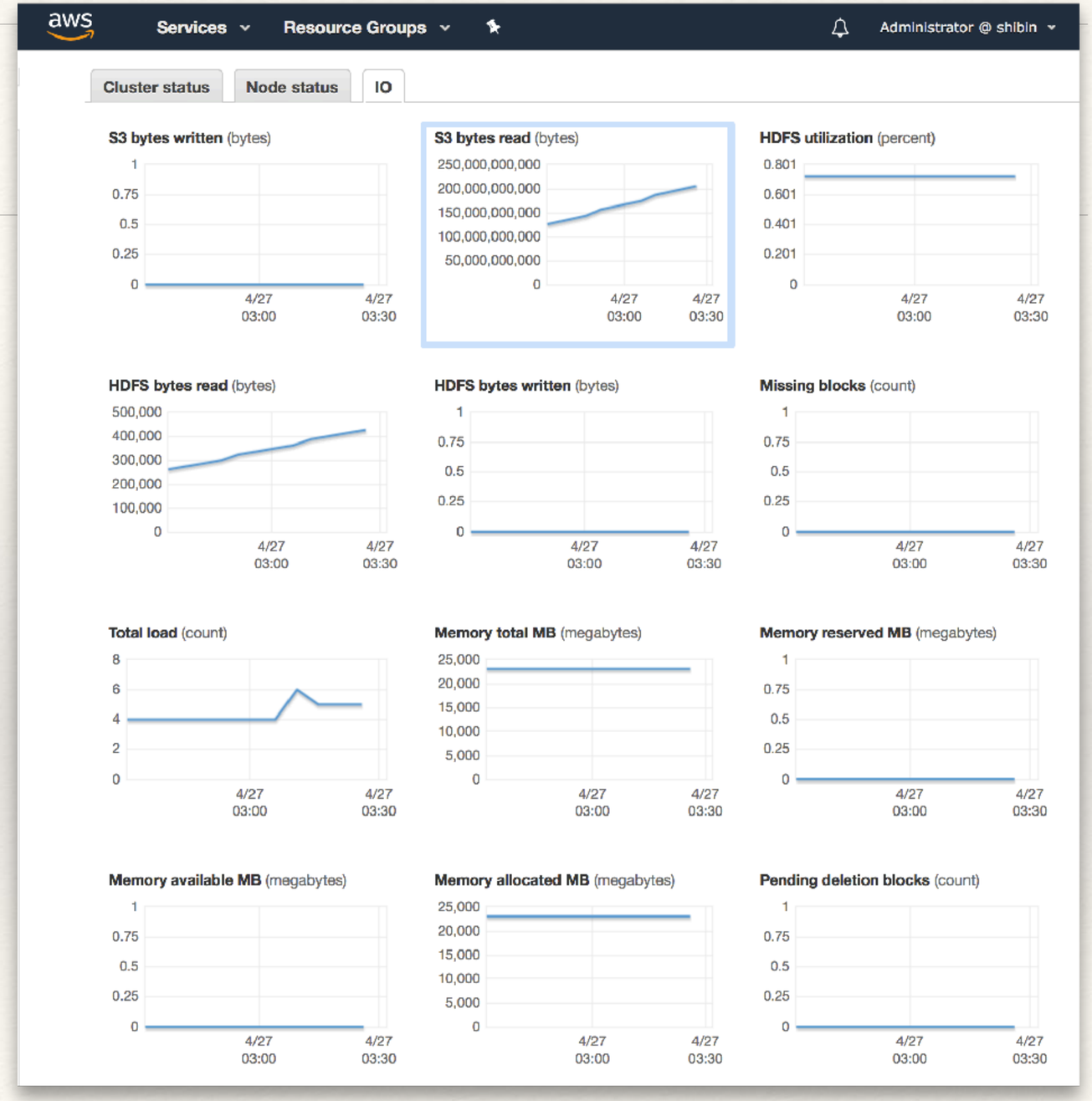
Filter: load more

Task	Type	State	Start time (UTC-7)	Actions
m_000000	MAP	COMPLETED	2018-04-26 18:12:13 (UTC-7)	View attempts
m_000001	MAP	COMPLETED	2018-04-26 18:12:25 (UTC-7)	View attempts
m_000002	MAP	COMPLETED	2018-04-26 18:12:18 (UTC-7)	View attempts
m_000003	MAP	COMPLETED	2018-04-26 18:12:19 (UTC-7)	View attempts
m_000004	MAP	COMPLETED	2018-04-26 18:12:20 (UTC-7)	View attempts
m_000005	MAP	COMPLETED	2018-04-26 18:12:25 (UTC-7)	View attempts
m_000006	MAP	COMPLETED	2018-04-26 18:12:20 (UTC-7)	View attempts
m_000007	MAP	COMPLETED	2018-04-26 18:12:27 (UTC-7)	View attempts
m_000008	MAP	COMPLETED	2018-04-26 18:12:29 (UTC-7)	View attempts
m_000009	MAP	COMPLETED	2018-04-26 18:12:30 (UTC-7)	View attempts
m_000010	MAP	COMPLETED	2018-04-26 18:12:26 (UTC-7)	View attempts
m_000011	MAP	COMPLETED	2018-04-26 18:12:31 (UTC-7)	View attempts
m_000012	MAP	COMPLETED	2018-04-26 18:12:33 (UTC-7)	View attempts
m_000013	MAP	COMPLETED	2018-04-26 18:12:39 (UTC-7)	View attempts
m_000014	MAP	COMPLETED	2018-04-26 18:12:36 (UTC-7)	View attempts
m_000015	MAP	COMPLETED	2018-04-26 18:12:49 (UTC-7)	View attempts
m_000016	MAP	COMPLETED	2018-04-26 18:13:01 (UTC-7)	View attempts
m_000017	MAP	COMPLETED	2018-04-26 18:12:49 (UTC-7)	View attempts
m_000018	MAP	COMPLETED	2018-04-26 18:12:48 (UTC-7)	View attempts
m_000019	MAP	COMPLETED	2018-04-26 18:13:04 (UTC-7)	View attempts

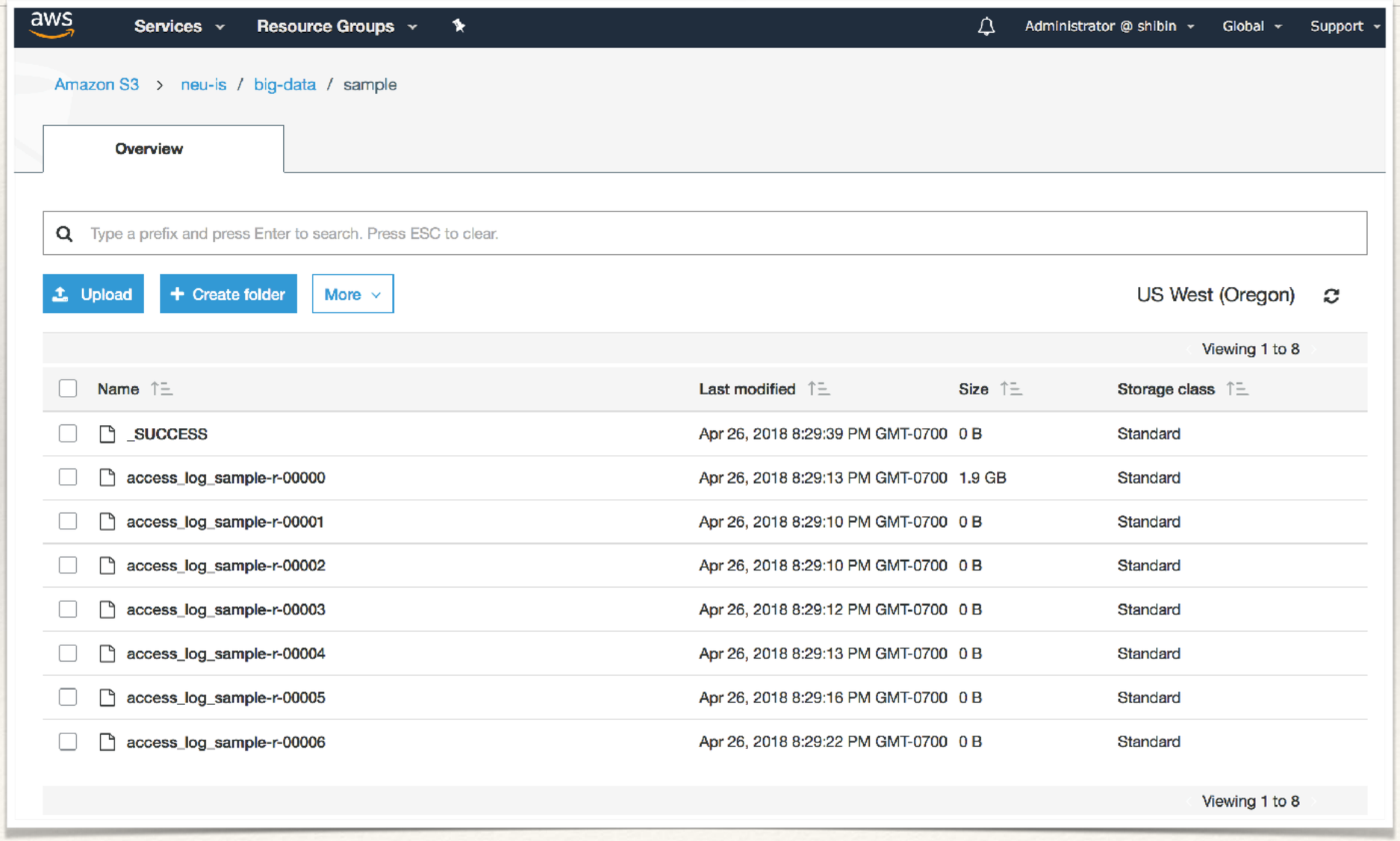
Working hard...

- S3: High read, low write
- (Very) Low HDFS IO





Get expected output, satisfied ?



EMR Cluster Job Summary

- Master: 1 x m3.xlarge
- Core: 2 x m3.xlarge

- 3178 tasks, 2.3 hours
- S3: High read, low write
- (Very) Low HDFS IO
- One output file: 1.9GB

```
Map-Reduce Framework
        Map input records=533889812
        Map output records=5160357
        Map output bytes=2081776950
        Map output materialized bytes=646197228
        Input split bytes=440730
        Combine input records=0
        Combine output records=0
        Reduce input groups=1
        Reduce shuffle bytes=646197228
        Reduce input records=5160357
        Reduce output records=5160357
        Spilled Records=10320714
        Shuffled Maps =22197
        Failed Shuffles=0
        Merged Map outputs=22197
        GC time elapsed (ms)=1446643
        CPU time spent (ms)=16723010
        Physical memory (bytes) snapshot=2097942646784
        Virtual memory (bytes) snapshot=10282272256000
        Total committed heap usage (bytes)=1966614249472
```

How Many Visits Each Day?

- key: date
- value: 1

```
public static class TheMapper extends Mapper<Object, Text, Text, IntWritable> {

    private Text date = new Text();
    private final static IntWritable ONE = new IntWritable(1);

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        // split the string using either ] or [
        String[] tokens = value.toString().split("]|\\[");

        if(tokens !=null && tokens.length > 1) { // exclude index.html
            date.set(LocalDate.parse(tokens[1], formatter).toString());
            context.write(date, ONE);
        }
    }
}
```

Mapper, Reducer, and Combiner. But really?

```
public static class TheReducer
extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context
            throws IOException, InterruptedException {

        int count =
                StreamSupport.stream(values.spliterator(), false)
                .mapToInt(i->i.get())
                .sum();
        context.write(key, new IntWritable(count));
    }
}
```

```
job.setCombinerClass(TheReducer.class);
```

Counter ?

Start small (Sample data)

- 2017-09-30 1
- 2017-10-01 136999
- 2017-10-02 159346
- 2017-10-03 163462
- 2017-10-04 163093
- 2017-10-05 161678
- 2017-10-06 149585
- 2017-10-07 137522
- 2017-10-08 141987
- 2017-10-09 169257
- 2017-10-10 195056
- 2017-10-11 188498
- 2017-10-12 186526
- 2017-10-13 185550
- 2017-10-14 171802
- 2017-10-15 168730
- 2017-10-16 196601
- 2017-10-17 195836
- 2017-10-18 186674
- 2017-10-19 184203
- 2017-10-20 171713
- 2017-10-21 147815
- 2017-10-22 145077
- 2017-10-23 176996
- 2017-10-24 174732
- 2017-10-25 174483
- 2017-10-26 172623
- 2017-10-27 159105
- 2017-10-28 134655
- 2017-10-29 133806
- 2017-10-30 166398
- 2017-10-31 160547
- 2017-11-01 1

Map input records=5160357
Map output records=5160357
Map output bytes=77405355
Map output materialized bytes=4754
Input split bytes=2368
Combine input records=5160357
Combine output records=274
Reduce input groups=33
Reduce shuffle bytes=4754
Reduce input records=274
Reduce output records=33
Spilled Records=548
Shuffled Maps =16
Failed Shuffles=0
Merged Map outputs=16
GC time elapsed (ms)=2424
CPU time spent (ms)=130340
Physical memory (bytes) snapshot=8153784320
Virtual memory (bytes) snapshot=76471353344
Total committed heap usage (bytes)=6979846144
Peak Map Physical memory (bytes)=522702848
Peak Map Virtual memory (bytes)=4438278144
Peak Reduce Physical memory (bytes)=217370624
Peak Reduce Virtual memory (bytes)=5548224512

Achieve big (full data)

2017-09-30	85
2017-10-01	14368780
2017-10-02	16549503
2017-10-03	16960781
2017-10-04	16946224
2017-10-05	16771744
2017-10-06	15571863
2017-10-07	14307579
2017-10-08	14786606
2017-10-09	17524765
2017-10-10	20052298
2017-10-11	19373614
2017-10-12	19278255
2017-10-13	19146386
2017-10-14	17759991
2017-10-15	17487473
2017-10-16	20116080
2017-10-17	20097309
2017-10-18	19183608
2017-10-19	19010848
2017-10-20	17755039
2017-10-21	15376696
2017-10-22	15165334
2017-10-23	18254586
2017-10-24	18111448
2017-10-25	18018861
2017-10-26	17806645
2017-10-27	16435430
2017-10-28	14032539
2017-10-29	13974421
2017-10-30	17094559
2017-10-31	16570390
2017-11-01	50

Map-Reduce Framework

Map input records=533889812
Map output records=533889790
Map output bytes=8008346850
Map output materialized bytes=2034821
Input split bytes=440730
Combine input records=533889790
Combine output records=3301
Reduce input groups=33
Reduce shuffle bytes=2034821
Reduce input records=3301
Reduce output records=33
Spilled Records=6602
Shuffled Maps =123669
Failed Shuffles=0
Merged Map outputs=123669
GC time elapsed (ms)=1891193
CPU time spent (ms)=41694260
Physical memory (bytes) snapshot=2315772375040
Virtual memory (bytes) snapshot=10434531799040
Total committed heap usage (bytes)=2227441238016

Powerful EMR, reach soft limit (20 EC2)

Clone | Terminate | AWS CLI export

Cluster: Enhanced Cluster Running Running step

Summary | Application history | Monitoring | Hardware | Events | Steps | Configurations | Bootstrap actions

Connections: Enable Web Connection – Resource Manager ... (Vie
Master public DNS: ec2-34-211-154-195.us-west-2.compute.amazonaws
Tags: -- View All / Edit

Summary
ID: j-ZB0SRE3CTAOE
Creation date: 2018-04-15 12:34 (UTC-7)
Elapsed time: 20 minutes
Auto-terminate: No
Termination protection: Off Change

Configuration detail
Release label:
Hadoop distribution:
Applications:
Log URI:
EMRFS consistent view:
Custom AMI ID:

Network and hardware
Availability zone: us-west-2a
Subnet ID: subnet-5715f52e
Master: Running 1 units
Core: Running 80 units
Task: Running 48 units

Security and access
Key name:
EC2 instance profile:
EMR role:
Visible to all users:
Security groups for Master:
Security groups for Core & Task:

Node type	Fleet instance types	Target capacity (vCores)	Advanced Spot options
Master Master - 1	**m3.xlarge** 8 vCore, 15 GiB memory, 80 SSD GB storage EBS Storage: none Maximum Spot price: % On-Demand 100 Add / remove instance types to fleet	On-demand Spot *The master fleet consists of one EC2 instance*	**Defined duration** 1 hour **Provisioning timeout** Terminate cluster after 60 min. of Spot unavailability
Core Core - 2	**m3.xlarge** 8 vCore, 15 GiB memory, 80 SSD GB storage EBS Storage: none Maximum Spot price: % On-Demand 50 Add / remove instance types to fleet	0 On-demand vCores 80 Spot vCores **80 Total vCores**	**Defined duration** 1 hour **Provisioning timeout** Terminate cluster after 50 min. of Spot unavailability
Task ✖ Task - 3	**m3.xlarge** 8 vCore, 15 GiB memory, 80 SSD GB storage EBS Storage: none Maximum Spot price: % On-Demand 50 Add / remove instance types to fleet	0 On-demand vCores 48 Spot vCores **48 Total vCores**	**Defined duration** Not set **Provisioning timeout** Terminate cluster after 50 min. of Spot unavailability

I want to know more, but how!

- What is the busiest hour?
- Which countries does the traffic come from?
- Which is top 10 URL categories in US?

Pre-processing using Pig

```
-- map IP -> country, city using using GeoLiteCity.dat http://dev.maxmind.com/geoip/legacy/install/city/
register /home/hadoop/resource/pig-udf-0.0.1-SNAPSHOT.jar
register /home/hadoop/resource/geoip-api-1.3.1.jar
-- the .dat should be available in HDFS so that each node could get and use it locally
fs -get cloudacl/resource/GeoLiteCity.dat
a = LOAD '$INPUT' AS (line:chararray);
b = FOREACH a GENERATE flatten(REGEX_EXTRACT_ALL(line, '(.*?) .*?\\[(.*?)\\].*?&cat=(.*?) .*')) AS (ip:chararray,
dt:chararray, cat:chararray);
c = FILTER b BY ip IS NOT null;
-- get country geoinformation
d = FOREACH c generate ip, com.example.pig.GetCountry(ip) AS country, ToString(ToDate(dt, 'dd/MMM/yyyy:HH:mm:ss
+0000'), 'yyyy-MM-dd HH:00:00') AS dt, cat;

e = FILTER d BY country IS NOT null;
-- aggregate using country, date and catetory
f = GROUP e BY (country, dt, cat);
g = FOREACH f GENERATE flatten(group), COUNT(e);
-- save the output
STORE g INTO '$OUTPUT';
```

Pig UDF

- register UDF before using
- register dependent jars
- Pig will ship registered jars to backend

```
public class PigUDF extends EvalFunc<Tuple> {
    public String exec(Tuple t) throws IOException {
        if (cl == null) {
            cl = new LookupService("GeoLiteCity.dat",
                    LookupService.GEOIP_MEMORY_CACHE);
        }
        Location loc = cl.getLocation((String) t.get(0));
        if (loc == null) {
            return null;
        }
        return loc.countryCode;
    }
}
```

Hive: Store Structured Data

```
create table access_log(country string, dt timestamp, cat string, count int)
row format serde 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe' with
serdeproperties('field.delim'='\t','timestamp.formats'='yyyy-MM-dd HH:mm:ss')
stored as textfile;

-- hadoop fs -put part* /user/hive/warehouse/access_log

create table access_log_partitioned(country string, dt timestamp, cat string, count int)
partitioned by(d date) row format serde
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe' with
serdeproperties('field.delim'='\t', 'timestamp.formats'='yyyy-MM-dd HH:mm:ss') stored as
textfile;

set hive.exec.dynamic.partition.mode=nonstrict;
insert overwrite table access_log_partitioned partition(d) select country, dt, cat, count,
cast(dt as date) from access_log;
show partitions access_log_partitioned;
```

Table, who does not like it

CN	2017-10-05 12:00:00	travel	100
CN	2017-10-05 13:00:00	religion	5
CN	2017-10-05 13:00:00	personal-site-and-blog	75
CN	2017-10-05 14:00:00	business-and-economy	34
CN	2017-10-05 14:00:00	internet-communication	12
US	2017-10-26 08:00:00	travel	89
US	2017-10-26 09:00:00	religion	234
US	2017-10-26 09:00:00	online-storage	46
US	2017-10-26 09:00:00	alcohol-and-tobacco	11
US	2017-10-26 09:00:00	entertainment-and-art	220
US	2017-10-26 09:00:00	personal-site-and-blog	147

```
hive> desc access_log_partitioned;

country             string
dt                  timestamp
cat                 string
count               int
d                   date

# Partition Information
# col_name          data_type         comment
d                   date
```

Again, how many visits per day?

```
select d, sum(count) from access_log_partitioned group by d order by d;
```

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.37 sec   HDFS Read: 106280680 HDFS Write: 901 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.46 sec   HDFS Read: 6430 HDFS Write: 1131 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 830 msec
```

2017-09-30	85
2017-10-01	14339144
2017-10-02	16512951
2017-10-03	16925291
2017-10-04	16910908
2017-10-05	16730374
2017-10-06	15530652
2017-10-07	14269510
2017-10-08	14752002
2017-10-09	17487247
2017-10-10	20013993

2017-10-11	19331226
2017-10-12	19232466
2017-10-13	19102088
2017-10-14	17720183
2017-10-15	17445980
2017-10-16	20074808
2017-10-17	20052138
2017-10-18	19141494
2017-10-19	18967594
2017-10-20	17711539

2017-10-21	15345248
2017-10-22	15116395
2017-10-23	18212291
2017-10-24	18061024
2017-10-25	17973108
2017-10-26	17766064
2017-10-27	16394004
2017-10-28	13993784
2017-10-29	13934136
2017-10-30	17047771
2017-10-31	16525742
2017-11-01	50

Time taken: **47.109 seconds**, Fetched: 33 row(s)

What is the busiest hour?

```
select hour(dt), sum(count) as ct from access_log_partitioned
group by hour(dt) order by ct desc limit 1;
```

```
MapReduce Total cumulative CPU time: 3 seconds 130 msec
Ended Job = job_1524803915024_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 11.06 sec   HDFS Read: 106280703 HDFS Write: 644 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.13 sec   HDFS Read: 6243 HDFS Write: 111 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 190 msec
OK
16    27671406
Time taken: 63.47 seconds, Fetched: 1 row(s)
```

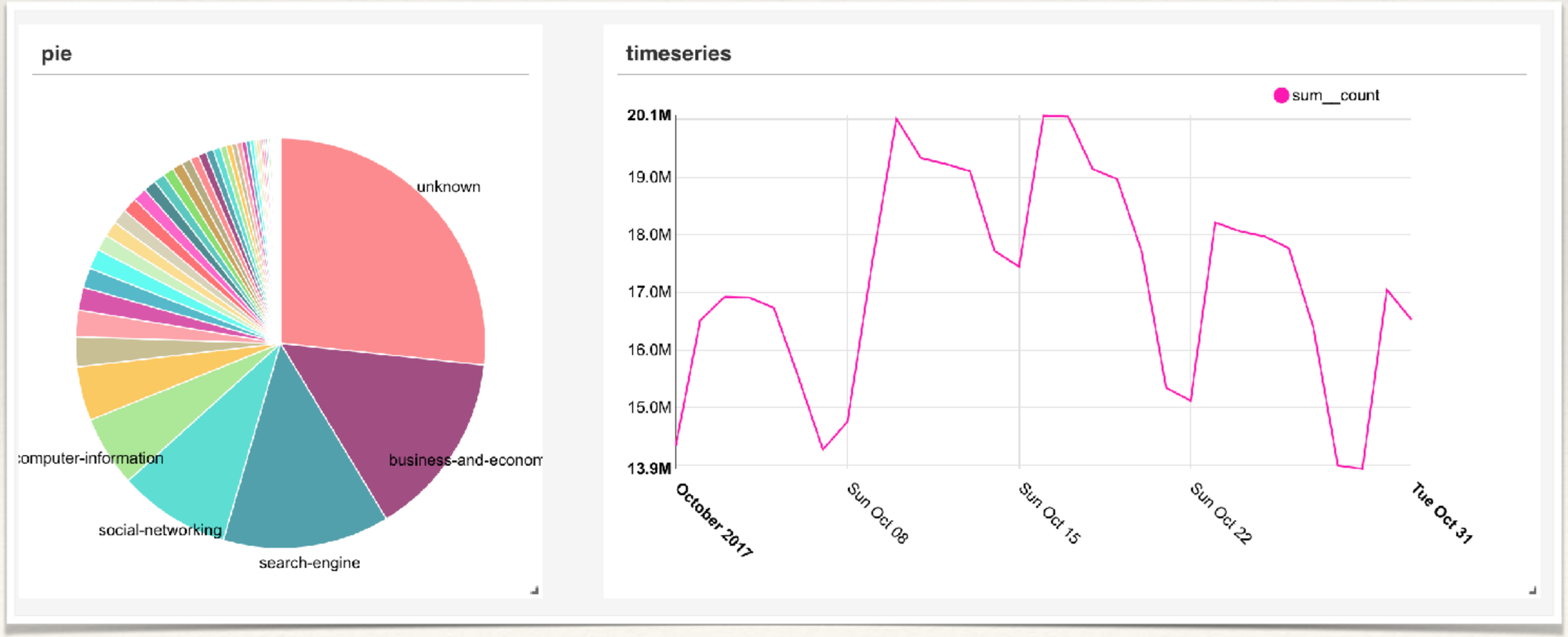
Which top 10 countries have most total visits?

```
select country, sum(count) as s from access_log_partitioned
group by country order by s desc limit 10;
```

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.03 sec   HDFS Read: 106280696 HDFS Write: 5417 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 5.28 sec   HDFS Read: 11123 HDFS Write: 327 SUCCESS
Total MapReduce CPU Time Spent: 15 seconds 310 msec
OK
US    90079276
PE    85283216
PH    50572854
MX    23807685
CO    21391952
IT    18015359
BO    17605652
VE    14599674
EC    12835092
AR    12022976
Time taken: 67.37 seconds, Fetched: 10 row(s)
```

Getting bored, let's see something super!

- Which is top 10 URL categories in US?
- What is the trend of the traffic looks like in total, or by country?



Visualize Data using Superset