Search engine for large collection of proxy logs

Emmanuel Benazera emmanuel.benazera@seeks.pro
SEEKS SAS, 36 rue Jacques Babinet,
31000 Toulouse France

1. Description

Steria faces very large ($> 50G$ per day) amounts of proxy log collections from which to draw useful reports. Fast generation of user reports requires fast processing and indexing of logs. This documents briefly describes and documents a proof of concept for processing the logs and building a searchable index.

The purpose of the system is to first collapse large amounts of logs into smaller amounts (i.e. by aggregating log records for every user, visited domain name, ...). Second, the system indexes those collapsed logs in such a manner as to make queries of the index very fast (i.e. below the second).

The built system comprehends :

- C++ map-reduce code (see details below) for heavily multithreaded log collapsing and indexing ;
- A pre-configured Apache Solr Cloud instance with four shards, acting as the search engine itself.

2. Architecture

The architecture is built on map-reduce (MR) for log processing and Apache Solr for building a distributed and scalable search engine.

2.1 Map Reduce with Metis

The architecture uses a two step process built as MR jobs :

- log collapsing ;
- log indexing.

Both jobs have been implemented in C++ with Metis (`http://pdos.csail.mit.edu/metis/`). Metis takes full advantage of multi-core machines, thus leading to very high performance on a single server (though it doesn't scale horizontally across several machines).

The code is available in the Metis repository, see usage in the section below.

For compiling the code, go to the Methis reposity then run

```
./configure
make
```

2.2 Big Data search engine with Solr

Apache Solr Cloud is a high performance scalable search engine. It has been preconfigured for using a four shards index (i.e. an index split in four), with the ability to scale across machines.

3. Usage

Below are the two steps

3.1 Log collapsing

Assuming that the logs are located in /home/steria/logs

```
cd Metis
./obj/bluecoat_log_proc /home/steria/logs/* -b -o logs_out.json -j
```

This collapses every log file in reposity /home/steria/logs and produces a JSON file of collapsed logs, ready to be indexed with Solr.
Two collapsing schemes are implemented : per day and per hour, default being per day. To activate per hour collapsing, add -t 1, that is run the command

```
./obj/bluecoat_log_proc /home/steria/logs/* -b -o logs_out.json -j -t 1
```

3.1.1 Tips & Performances

This MR job runs in memory. The required memory by each MR job is directly proportional to the size of the log file it processes. For this reason it is recommended to test the log collapsing program on files of different sizes, then split the largest log files accordingly.
A convenient way to split file is to use the split program on Unix systems :

```
split -l 1500000 log1
```

will split the log1 file into smaller files of 1.5M lines each.
Another way to lower the memory trace is to limit the number of map jobs, as follows :

```
cd Metis
./obj/bluecoat_log_proc /home/steria/logs/* -b -o logs_out.json -j -m 10
```

This will tell the program to use 10 mapping jobs. Note that the output remains exactly the same indpendently of the number of mapping jobs.

3.2 Starting the search engine

If the Solr is already running, skip this step.
To start Solr Cloud :

```
cd apache-solr-4.0.0
./jetty.sh start 4
```

To stop Solr Cloud :

```
cd apache-solr-4.0.0
./jetty.sh stop 4
```

To get a count of the number of elements in the index :

```
cd apache-solr-4.0.0
./count_results.sh
```

To erase the index :

```
cd apache-solr-4.0.0
./clear_index.sh
./clear_index.sh
```

Note the need to call clear_index.sh twice.

3.3 Log indexing

Indexing the log is achieved with a MR job in order to easily maximize the capability of the server :

```
cd Metis
./obj/solr_commit logs_out.json -m 10
```

This will index the collapsed logs into Solr.
It has been observed that the program can put Solr under too much stress and thus generate failures. For this reason, the program reports the number of successful calls.
In order to limit the stress on the Solr indexer, it is recommended to limit the number of mapping jobs. Thus the example above uses a 10 mapping jobs call.

3.4 Search & Queries

Once some data have been indexed, queries can be issued.
Examples of queries :

```
curl "http://localhost:8984/solr/collection1/select?q=url_host:google.com*&rows=0"
```

reports records of users having visited any URL from google.com.

```
curl "http://localhost:8984/solr/collection1/select?q=username:hal&rows=0"
```

reports records of user hal.

```
curl "http://localhost:8984/solr/collection1/select?q=username:hal \
        &logstamp=2012-02-03T00:00:00Z+2DAYSrows=0"
```

reports records of user hal between the 3rd and 5th of February 2012. See `http://lucene.apache.org/solr/api-4_0_0-BETA/org/apache/solr/schema/DateField.html` for more information on how to query date fields with Solr.

Note that Solr supports the service of queries while indexing. Therefore in production, new logs can be indexed without disrupting service.