HADOOP AND PIG FOR INTERNET CENSUS DATA ANALYSIS

Aleksandar Nikolić, Goran Sladić, Branko Milosavljević, Stevan Gostojić, Zora Konjović
{anikolic, sladicg, mbranko, gostojic, ftn_zora}@uns.ac.rs
Faculty of Technical Sciences, University of Novi Sad

Abstract – *Internet Census 2012 data provides a very comprehensive look at the state of the Internet as a whole. The dataset is a result of extensive network scans covering available public IP space. In this paper we present a platform based on Hadoop, large scale data processing framework, for analysing this dataset. Custom Pig Latin User Defined Functions were developed which present a data specific querying interface allowing faster and more flexible analysis.*

1. INTRODUCTION

Internet Census 2012 [1] was made publicly available in late 2012. The data is the result of comprehensive Internet-wide scans performed by Carna Botnet. Created by anonymous researcher, it exploited insecure credentials on embedded devices.
Many devices on the Internet are unprotected and left with default username and password exposing them to abuse. Carna scanned the Internet for such devices, adding to its botnet and using them for future scans. The botnet consisted of 420 thousand embedded devices which represents 25% of the total number of vulnerable devices. Figure 1 shows geographical distribution of infected embedded devices.

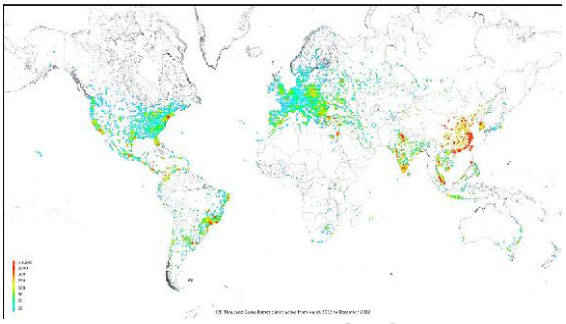

Figure 1. Carna Botnet distribution.

Programs installed on infected devices were non-intrusive and the device would return to its prior state upon reboot. Scanning rate was limited to not overload the device. As all infected devices were available over Internet, botnet could have been designed without the Command & Control server which has the advantage of not having a single point of failure. Scan data was coalesced from infected machines by middle nodes from where it was downloaded by the master server. The bot binary was built for 9 different architectures and was 40 to 60 kilobytes in size.

To discover which IP addresses were actually used on the Internet, Carna Botnet conducted ICMP Ping scans on IPv4 address space available for public use. This scan used a modified version of fping [2] tool. The dataset contains reverse DNS records acquired by running reverse DNS queries for every IP address. Port scan data was acquired by running a version of Nmap[3] for 100 most common ports. To check if the host is alive, Nmap does hostprobe requests which can then be used to roughly determine the host operating system type. The dataset also contains service probe results. By sending a certain type of packet to a port, a service probe, and matching the response to the known database the type of the service can be determined. Smaller vulnerable devices, which could not be infected with a bot, were used to run random traceroute scans. All the scans were conducted from March to December of 2012 and included 420 million IP addresses.

The dataset was released for download over BitTorrent and consists of 568 gigabytes of zpaq archives, segmented by IP address ranges. Zpaq compression was chosen as it offers highest compression ratio. There are total of eight different scan types: ICMP ping, reverse DNS, service probes, host probes, syn scan (open port scan), TCP/IP fingerprints, IP ID sequences and traceroute scans.

Previous scans of this scale focused on a limited subset of services or TCP/IP features. In 2007, Heidemann et al. [4] conducted a first attempt at measuring the population of visible Internet hosts. They present host population results based solely on ICMP echo replies. Analysis of Internet Census 2012 dataset by the original author has shown comparable results. Using Zmap [5], researchers have conducted a large scale study of HTTPS certificates infrastructure. They have performed 110 Internet wide scans over 14 months and analysed trust relationships between certificate authorities [6].

Internet Census 2012 dataset presents a unique view at the state of the Internet in 2012. Because of its size, in order for us to conduct deeper dataset analysis, a large scale computing platform was required. In this paper we propose storage and computing cluster solution based on Hadoop platform for analysing Internet Census 2012 dataset. Hadoop's distributed file system presents a simple way for storing large amounts of data. Apache Pig is an extensive language especially suited for querying machine generated text files. By

extending Pig with custom User Defined Functions (UDFs) we created a platform for analysing this dataset.

In section 2 Hadoop key concepts and Pig Latin scripting language is introduced. In section 3 concrete cluster organizations is presented. Section 4 shows dataset organization, and section 5 presents custom UDFs and data processing examples.

2. HADOOP AND PIG

Hadoop [7] is a large-scale computation and data storage framework. It is designed to provide high reliability and availability and to run on commodity hardware. It constitutes a system for storing and processing of data in a highly distributed and parallel fashion. Hadoop is comprised of two main points [10]:

- Map/Reduce [11] – a framework for assigning work to the nodes
- HDFS [12] – a distributed file system

Map/Reduce, in a broader sense, represents a programming paradigm for processing large data sets. It consists of two phases. Map phase performs sorting and clustering of data, and Reduce phase performs summation of data in those groups. In Hadoop, Map/Reduce is a way of writing programs running on Hadoop clusters. The Map phase is responsible for breaking up the dataset into independent pieces which are then processed in the Reduce phase [11].

Hadoop distributed file system is designed for high input and output speeds. It is used for storing large input and output files for Hadoop processing. It provides high throughput rates by splitting the files and scattering them throughout the cluster. Knowing where individual pieces of data reside allows for optimized processing distribution [12].

A typical Hadoop Map/Reduce program in Java consists of two classes: A mapper class implementing a *Mapper* interface, and a reducer class implementing *Reducer* interface. Mapper class' job is to process supplied data into key/value pairs which represent intermediate results. Hadoop framework spawns one mapper task for each distributed part of the data. Reducer class' processes intermediate key/value pair results into a smaller set of values that share a key.

Hadoop programs present a very low level data processing interface. Apache Pig [8] represents a higher level platform for creating Map/Reduce programs on top of Hadoop. It consists of a compiler that produces sequences of Map/Reduce programs from its Pig Latin programming language [9]. Pig Latin combines high level declarative querying of SQL and low-level procedural programming. Each Pig Latin program consists of multiple steps, each step being a type of query. As opposed to SQL, Pig is designed to support ad-hoc data analysis. Data can be queried directly without the need for importing into tables. To allow custom data processing, Pig Latin has support for user defined functions (UDFs) which can be used to customize all aspects of data processing. Usual Pig Latin script consists of a LOAD statement that reads data from the HDFS, a series of data transformations and a STORE statement that writes results to the HDFS.

By leveraging Hadoop's HDFS large Internet Census dataset can be distributed and stored on a number of nodes without the need of keeping track of individual pieces. As Internet Census dataset comes in form of textual log files, Apache Pig lends its self as a natural choice for its processing.

3. CLUSTER ORGANIZATION

Every Hadoop cluster is composed of a number of different logical nodes (see Figure 2):

1) *NameNode* – stores location information and metadata about the files in HDFS which is required when retrieving data spread across the cluster.
2) *SecondaryNameNode* – periodically downloads *NameNode* image and creates a checkpoint.
3) *JobTracker* – takes requests from a client and assigns *TaskTrackers* with tasks to be performed.
4) *TaskTracker* – accepts Map and Reduce tasks form the *JobTracker*.
5) *DataNode* – stores files in HDFS and manages file blocks within the node.

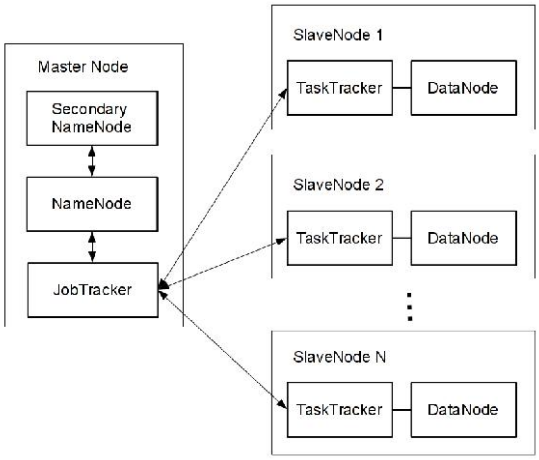

Figure 2. Hadoop cluster components

Hadoop follows a master/slave design. A simplest cluster consists of one master node and one or more slave nodes. The physical master node is composed of *NameNode*, *SecondaryNameNode* and *JobTracker* logical nodes. Slave node consists of *DataNode* and *TaskTracker* logical nodes. Each logical node runs in

its own Java VM. Single point of failure for the Hadoop cluster is *NameNode* which in case of a failure can be restored from *Secundary NameNode* checkpoint. TO reduce the impact of failing *NameNode*, bigger clusters can have a multi-tier, tree-like, structure and can have *NameNode* and *JobTracker* nodes on different physical nodes.

Our cluster consists of one master node running the *NameNode*, *SecundaryNameNode* and *JobTracker* logical nodes and 12 slave nodes running *TaskTracker* and *DataNode* logical nodes. All slave nodes have Intel Core2Duo processors, 2 gigabytes of RAM and 170 gigabytes of hard drive space available, all connected in a local 100Mbps network. In total, there are 24 cores available for parallel data processing and 2 terabytes for HDFS data storage.

4. DATASET ORGANIZATION

Internet Census dataset is organized in a number of file system directories: *hostprobes, synscans, icmp_ping, rdns* and *serviceprobes*. All directories, except *serviceprobes*, contain a number of zpaq archives, one for each public class A network. *Serviceprobes* directory contains gzip archives, for each service probe type, each in turn containing a number of zpaq archives for each class A network. Traceroute results are stored in single zpaq archive.

Direct data processing on zpaq archives is unpractical due to its slow decompression. Our tests have shown that decompressing a single 80 megabyte zpaq archive on our slave node takes up to 40 minutes. Slow decompression speed is due to high compression ratio of dataset archives, which is roughly 18:1.

Estimated size of plain text data is 9 terabytes. Since our cluster has 2 terabytes of HDFS space available, a time/memory compromise has been made by recompressing plain text data into gzip archives. Gzip has been chosen for its decompression speed and ability to query files without fully decompressing them first. Decompression of zpaq archives and recompression into gzip archives has been executed on all 12 cluster nodes. Master node would keep a queue of archives to be processed, distributing two by two to slave nodes for recompression and fetch the results. This step was implemented with common Linux shell tools without reliance on Hadoop cluster. Recompression of h*ostprobes, synscans, icmp_ping* and *rdns* archives took 2 days. Recompression of *serviceprobes* archives additionally took 4 days. Resulting dataset is 1.6 terabytes in size. It has been loaded into HDFS while preserving the same directory structure. Apache Pig has transparent support for loading gzip files.

5. QUERYING THE DATA

Since Internet Census 2012 dataset is domain specific, as it represents network scan results, some conventions can be used to effectively query the data. Each dataset file is organised as a tab separated, columnar text file. Every file consists of one column representing the IP address and one or more columns representing the data for that address. For example, Table 2 shows one excerpt from reverse DNS records. First column is IP address, second is timestamp in Unix Epoch time and third is reverse DNS query response, number in brackets specifying error codes.

IP	Timestamp	Result
108.0.140.255	1336954500	(3)
108.0.141.0	1336886100	(2)
108.0.141.1	1336914900	L100.LSANCA-VFTTP-165.verizon-gni.net

Table 1. Reverse DNS query sample

As such, data can be loaded and organized by the Pig Latin script in a straightforward way as shown in the listing 1.

```
RDNS = LOAD '/rdns/' USING
PigStorage('\t') AS (ip, timestamp,
result);
```

Listing 1. Loading the data

Keyword LOAD expects file or directory location from which to load files, method for parsing each line, and names for columns.

Since we will rarely want to execute queries on the whole IP range, loading only a subset of needed files is an imperative. The usual way of specifying network ranges in network scanning software is CIDR (Classless Inter-Domain Routing) notation relying on network id and netmask. Data is sorted into files for each network with netmask 255.0.0.0, or class A network so only files falling into the network range we are interested in would need to be loaded. For example, to query the data from network 147.91.0.0/16, only /rdns/147.gz file needs to be loaded, and for network 144.0.0.0/5 files from 144.gz to 151.gz should be loaded as all addresses in between belong to the given network. To load a range of files, shell expansion can be used as shown in Listing 2. to substitute directory path in the load statement with a Pig variable *input*.

```
pig -f script.pig -param
input=/rdns/{144..151}.gz
```

Listing 2. Shell expansion example

Additionally, IP addresses need to be filtered for networks smaller than those with 8 bit netmask. User should be able to specify network ID and netmask and

filter results belonging to that network only. A filter in a form of User Defined Function can be used. Filter UDFs need to extend the *FilterFunc* class and implement the *exec* method which returns a boolean value. Listing 3 shows how to use a filter UDF with parameters. To pass parameters to a UDF, the class constructor is used.

Listing 4 shows an excerpt of a UDF for filtering IP addresses by network ID and netmask. The network ID and network mask are set in the constructor before the filter is run. IP address is first converted from string to the integer representation and then verification is made.

```
REGISTER internetCensus.jar;
DEFINE IPBelongs
internetCensus.IPBelongs("147.91.175.
0",”17”);
B = FILTER A by IPBelongs(ip);
```

Listing 3. Filtering IP addresses

```
public class IPBelongs extends FilterFunc {
  int netID;
  int mask;
  public IPBelongs(String netID, String maskBits){
    this.netID = ip2int(netID);
    this.mask =  -1 << (32 – Integer.parseInt(maskBits));
  }
  public int ip2int(String ip){
    Inet4Address ipAddress = (Inet4Address) InetAddress.getByName(ip);
    byte[] ipBytes = ipAddress.getAddress();
    int ipInt = ((ipBytes[0] & 0xFF) << 24) |
    ((ipBytes[1] & 0xFF) << 16) |
    ((ipBytes[2] & 0xFF) << 8)  |
    ((ipBytes[3] & 0xFF) << 0);
    return ipInt;
  }
  public Boolean exec(Tuple input) {
    if (input == null || input.size() == 0)
    return null;
    try {
      Object value = input.get(0);
      int ip = ip2int(value);
      return ((netID & mask) == (ip & mask));
    } catch (ExecException ee) {
      throw WrappedIOException.wrap(ee);
    }
  }
}
```

Listing 4. IP address filter

While performing data analysis on *synscan* and *service probes* data, certain ports and port ranges might be of special interest. A filter UDF similar to the IP address one can used to filter ports. We adopted Nmap's style of specifying port ranges, meaning the user can specify a single port, an array of comma separated ports or a port range with starting and ending port. UDF constructors can only be passed strings as parameters so the constructor is responsible of generating a list of ports. Filter usage is shown in listing 5.

```
REGISTER internetCensus.jar;
DEFINE FilterPort
internetCensus.FilterPort("1024-
3306”)
B = FILTER A by FilterPort(ip);
```

Listing 5. Filtering results by ports

Service probe and IP fingerprint results have specific service and Operating System fingerprints for each scanned service and IP address which are stored in Nmap fingerprints format. To match these fingerprints to Nmap's database external programs need to be executed. Nmatch, a tool supplied with Internet Census dataset, can be used to match service probes. Fingermatch can be used to match IP fingerprints to OS. To facilitate the execution of external programs, Pig Latin has the notion of streams. STREAM operator is used to send data through external script or program. Only limitation is that external program has to accept input on STDIN and write output on STDOUT. Listing 6 shows an example of streaming service probe fingerprints to Nmatch. Each fingerprint is passed to Nmatch over STDIN; results are printed on STDOUT and added to C.

```
A = LOAD 'service_probes/80-
GetRequest/' USING PigStorage('\t')
AS (ip, timestamp, state, result);
B = FOREACH A GENERATE result;
C = STREAM B THROUGH './nmatch
GetRequest tcp';
```

Listing 6. Using STREAM operator

Presented UDFs can be used to easily analyse the dataset and extract meaningful results.

```
ALL_IPS = LOAD '/synscan/147.gz'
USING PigStorage('\t') AS (ip,
timestamp, state, reason, type,
ports);
REGISTER internetCensus.jar;
DEFINE IPBelongs
internetCensus.IPBelongs
      ("147.91.175.0",”23”);
NETMASK_IPS = FILTER A by
IPBelongs(ip);
DEFINE FilterPort
internetCensus.FilterPort ("53");
PORT_53 = FILTER NETMASK_IPS by
      FilterPort(ip);
UDP_PORT_53 = FILTER PORT_53 by type
== 'udp';
OPEN_UDP_PORT_53 = FILTER UDP_PORT_53
by    state == 'open';
STORE OPEN_UDP_PORT_53 INTO '/output'
      USING PigStorage ('\t');
```

Listing 7. Complete Pig Latin Script

Listing 7 presents a complete Pig Latin script that enumerates hosts in 147.91.175.0/23 network that have UDP port 53 opened, indicative of DNS server. Script in Listing 7 starts by loading the file containing data of interest. Each result is filtered multiple times. First, filtering only IP addresses we are interested in using IPBelongs UDF, then filtering only IP addresses that have port 53 in the results. Remaining IP addresses are filtered by the type (TCP or UDP) of port and finally by port state. Results are written back to HDFS using STORE operator.

6. CONCLUSION

Large amount of data contained in Internet Census 2012 dataset presents a unique snapshot of the state of the Internet as a whole in 2012. With the rising adoption of IPv6 Internet scans of this scale and completeness will soon be impossible. The dataset was created by an anonymous researcher by exploiting insecure embedded Internet devices. Analyzing over 9 terabytes of raw text files, representing the logs of Internet-wide scans, has proven to be a difficult task.

To speed up and distribute the analysis workload and data storage we created a Hadoop cluster with 12 slave nodes, totaling 24 CPU cores and over 2 terabytes of storage. In order to get fast processing speeds but still retain the dataset size on a manageable level, original zpaq archived dataset was recompressed into gzip archives. We utilized Pig and extended Pig Latin with custom UDFs to present how this platform can be used to extract meaningful information in a straightforward manner.

Presented platform can be a basis for data mining tools which can be used to perform extensive analysis and uncover more subtle properties of the Internet.

REFERENCES

[1] Port scanning /0 using insecure embedded devices, http://internetcensus2012.bitbucket.org/l.

[2] Fping, http://fping.sourceforge.net/.

[3] Nmap, https://nmap.org

[4] J. Heidemann, Exploring Visible Internet Hosts through Census and Survey, 2007.

[5] Z. Durumeric, ZMap: Fast Internet-Wide Scanning and its Security Applications, 22nd USENIX Security Symposium, August 2013.

[6] Z. Durumeric, Analysis of the HTTPS Certificate Ecosystem, Proceedings of the 13th Internet Measurement Conference, October 2013.

[7] Hadoop, https://hadoop.apache.org/

[8] Apache Pig, https://pig.apache.org/

[9] C. Olston, Pig Latin: A Not-So-Foreign Language for Data Processing, Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008.

[10] Hadoop Documentation, http://hadoop.apache.org/docs/current

[11] Hadoop MapReduce – Hadoop Documentation, https://wiki.apache.org/hadoop/MapReduce

[12] Hadoop HDFS – Hadoop Documentation, https://wiki.apache.org/hadoop/HDFS