UNIVERSITY OF OSLO
Department of Informatics

Log Analysis as a way to assist Opera mini cluster management decisions

Valeri Cheremetiev

Network and System Administration

Oslo University College

May 19, 2008

UNIVERSITAS OSLOENSIS • MDCCCXI •

Log Analysis as a way to assist Opera mini cluster management decisions

Valeri Cheremetiev

Network and System Administration
Oslo University College

May 19, 2008

Abstract

This thesis considers ways that analysis of Opera mini logs can assist decisions related to global and local load balancing of Opera mini clusters. The analysis is aimed to determine the distribution of traffic with respect to country of origin and server within the cluster over the period of 2 weeks by creating a system for extraction and analysis of log data. Findings show that a large part of traffic originates in Russia with India and Indonesia being second and third. This trend is expected to continue. The total number of requests is shown to be increasing exponentially. The internal load balancing algorithm shows uneven results due to memory effect and high load on the cluster during the peak time.

Acknowledgements

I would like to thank:

- my classmate and collaborator, Stian Østen for Keeping me company during this semester and being there to listen to my at times crazy ideas
- rest of my classmates for cheering me up from time to time
- Kyrre Begnum, Hårek Haugerud and Tore Jonassen at Oslo University College for guidance and advise
- Professor Mark Burgess and rest of the OUC and OU staff that made the Master Programme possible and taught me a lot
- and last but not the least Claudia Eriksen, Trond Aspelund and Sven Ulland at Opera Software for suggesting the topic for this thesis and taking time out of their busy schedules to help me.

Contents

1 Introduction **6**
1.1 Problem statement 6
1.2 Operationalization 6
1.3 Research goals 7
1.4 Structure of this document 7

2 Background and theory **9**
2.1 Opera mini 9
2.1.1 Features 9
2.1.2 Back end 10
2.2 IP GeoLocation 11
2.2.1 Topology and delay based 11
2.2.2 Metadata based 11
2.3 Load Balancing 12
2.4 Log Analysis 14
2.5 Statistical analysis 14
2.5.1 Regression 15
2.5.2 Frequency distributions 16

3 Model and Methodology **17**
3.1 Overview 17
3.2 Equipment 17
3.2.1 Hardware 17
3.2.2 Software 17
3.3 Phase 1 18
3.3.1 Data source 18
3.3.2 Logs 18
3.3.3 Log Processing 18
3.4 Phase 2 21
3.4.1 SQL query examples 22
3.4.2 Result Processing scripts 22
3.5 Phase 3 22
3.5.1 Set commands 23
3.5.2 plot commands 23
3.6 Model 24

4 Results **27**

4.1 Country of Origin 27
4.1.1 Development of requests totals 31
4.1.2 Hourly variations 32
4.2 Transcoders 36
4.2.1 Transcoder request distributions 37
4.2.2 Distribution Height an Width per Country 38

5 Conclusions and Discussion 43
5.1 Discussion of results 43
5.2 Limitations 44
5.3 Further work 45

A Scripts i
A.1 Data Processing i
A.1.1 ipstrip.py i
A.1.2 dist_line.py ii
A.1.3 countryTH.pl iii
A.1.4 cfiltersql.py v
A.1.5 cfiltersql2.py viii
A.2 Data preparation ix
A.2.1 gprep.py ix
A.2.2 distrib_gp.py xi
A.3 Plotting xiii
A.3.1 distrib.py xiii

List of Tables

3.1 Query output example. 22

4.1 Distribution of requests between the top 9 countries and the rest with mean and standard deviation. 29
4.2 Number of requests over 2 weeks,population, number of subscriptions (millions) and mobile penetration as number of subscriptions divided by population. 30

List of Figures

2.1 Opera mini back end structure. The arrows show possible routes for request flow. 10

3.1 The log processing diagram. Arrows show input and output files. 20
3.2 Daily distributions of requests per transcoder 24

4.1 Totals for number of requests. 28
4.2 Mean percentage of requests for the top 9 countries and the rest with scatter as errorbars. 29
4.3 Number of requests per country for 2 weeks normalized by population and number of cellular subscribtions. 30
4.4 Totals for number of requests with regression. 31
4.5 Development of number of requests per country. 32
4.6 Number of requests per hour for the whole cluster 33
4.7 Number of requests per hour per country. 34
4.8 Number of requests per hour per country, stacked. 35
4.9 Number of requests per transcoder from 4/1/08 till 17/1/08 . . 35
4.10 Number of requests per hour per country, stacked. 37
4.11 Height and width of the request distribution. 40
4.12 Height and width of the request distribution per country. Top 9 countries sorted by total number of requests. 41
4.13 Height and width of the request distribution per country, part2. 41
4.14 Height vs width of the request distribution per country. 42

Preface

This thesis is the final part of a two year Master programme at Oslo University College in collaboration with Oslo University which I have taken part in during the years 2006 through 2008. This thesis is done in cooperation with Opera Software who suggested the subject of research and provided equipment, supervision and working environment. The topic of this thesis is the Opera mini browser and log analysis of it's servers. A thesis regarding a similar subject with a different focus and variables was done in parallel by one of my classmates who used a database created as part of this work to perform his analysis.

Chapter 1

Introduction

In the later years the Internet has experienced immense growth. According to Internet World Stats[1] it has become a part of life of almost one and a half billion people. Usage of mobile phones is even wider with over 3 billion subscribers according to Reuters[2]. Recently several technologies designed to bring those two together have emerged. One of such technologies is Opera mini, an Internet browser for mobile phones. It is not just a browser. It is also a system that is able to fetch any of the web pages hidden behind the 76 million registered domains[3] reformat and compress it and then deliver to any of the aforementioned mobile phones all around the world within seconds. That is a task requiring several server clusters with significant processing power and link capacity. That is why Opera Software is currently expanding their Opera mini systems by creating several such clusters. This is a task requiring making of decisions regarding location of the clusters and configuration of load balancing both between those clusters and internally within them. Making the right decisions can be vital to ensure adequate performance. These decisions must be rooted in knowledge about the current situation in the existing systems. However such knowledge is currently unavailable. This thesis attempts to remedy that.

1.1 Problem statement

The problem statement for this thesis is formulated as follows: I attempt to find out how can analysis of the number of requests relative to location of clients and internal cluster structure help the decisions regarding cluster placement and global and local load balancing for Opera mini.

1.2 Operationalization

To eliminate possible ambiguities I include an operationalization which explains the terms in the problem statement.

- **analysis** - statistical analysis of log data for Opera mini servers.

- **request** - a line in an Opera mini log file pertaining to a communication between an Opera mini server and a client
- **location** - a country
- **internal cluster structure** - pattern of communication between servers within an Opera mini cluster.
- **decision** - planing phase of the implementation
- **cluster** - a set of interconnected nodes running Opera mini software
- **load balancing** - interaction between clusters or single nodes aimed to spread the load between participants.

1.3 Research goals

The main goal of this thesis is to provide assistance for various cluster management decisions for Opera mini. To achieve that I will do the following:

- create a system of scripts capable of parsing Opera mini log files and gather the results in a database
- analyze the results with respect to several variables relevant to cluster management decisions i.e.

  country of origin

  time of day

  transcoder

1.4 Structure of this document

This document consists of 5 parts.

Introduction outlines the motivation for this thesis and then describes the problem statement as a kind of mission objective for this work.

Background that describes the relevant aspects of Opera mini functionality and attempts to give a survey of literature relevant to the work done in this thesis.

Model and methodology has two sections. The first describes in detail how the work was done and the second explains the model used in part of the analysis.

Results presents the results achieved in graphical form and attempts to explain them.

Conclusion and discussion sums up the results and discusses their relevance according to problem statement, then takes a critical look at some of the choices made in the start of the thesis and finally considers what possibilities for future work are available.

Chapter 2

Background and theory

In this section I will attempt to provide necessary theory background regarding Opera mini and various techniques used in this thesis. Opera mini is a quite new system and little to none previous research related to it have been done. There is however some work done on other systems that is related to some aspects of my thesis such as log analysis, IP GeoLocation and load balancing.

2.1 Opera mini

Opera mini is a technology that attempts to bring the full web to Java ME enabled mobile devices. The main difference from conventional browsers is that the Opera mini does not access the content directly, but instead contacts special servers called transcoders that fetch the web pages, process them and then deliver them to the mobile device. This process includes compressing the data to speed up the transfer by a factor of 2 or 3 and reformatting the pages to better suit the low resolution screens.[4] Since version 3 the connection is encrypted.[5] Opera mini was initially released in 2005. In 2007 Opera mini came preinstalled on 40 million mobile phones.[6] According to NetApplications.com Opera mini's marked share was 0.03% of all browsers both desktop and mobile.[7] Current version is 4.1 released 13.05.2008. All previous versions starting with 2.0 are still operational.

2.1.1 Features

Opera mini features include[4]:

- smooth scrolling and a virtual mouse pointer
- landscape mode
- several levels of image quality
- bookmark synchronization via myOpera site
- custom search engines

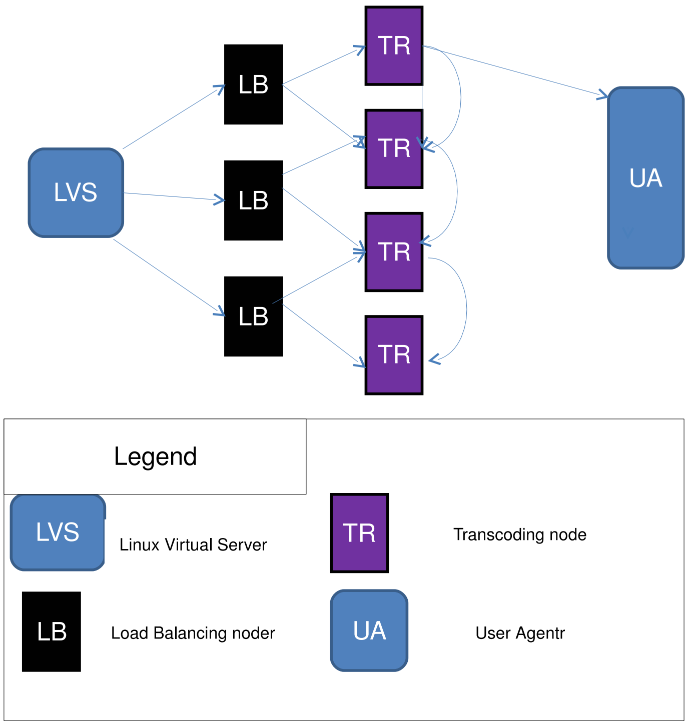


Figure 2.1: Opera mini back end structure. The arrows show possible routes for request flow.

- file down- and uploading with native file system support

2.1.2 Back end

Opera mini back end is built to allow for redundancy and parallel processing. The key parts are:

- Linux Virtual Server(LVS)
- Load Balancing nodes(LB)
- Transcoding nodes (TR)

Linux Virtual Server(LVS) A system performing dumb load balancing transparently. [8] The LVS hands the incoming requests to a set of Load Balancing nodes(LB) in a round robin fashion. The LBs are running Opera mini software that is performing load balancing between Transcoding nodes (TRs) according to an undocumented algorithm. The LBs are also transparent for the client. TRs can receive requests from the LBs or directly from User Agents (UAs) i.e. mobile phones. Upon receiving such a request the TR will fetch the requested web page from the Internet, process it and then send the result directly to the UA Figure 2.1 shows the flow of a request through the system starting with LVS. The LBs can force the requests to be forwarded from one TR to another ruled by the load balancing algorithm. The algorithm has persistency/memory effect, that is the requests from same client are likely to be sent to the same TR. This is achieved by using a cookie to identify clients. In current version of Opera mini this cookie is reset each time the client is restarted.

Opera mini client application can locate it's servers by using a set url. To implement global load balancing this url can be tailored to the users location specified at the time of downloading the client. DNS can also be used to direct clients requesting IP for this url to different locations. This url refers the client to a load balancer which can then choose to redirect the client to a different server.

2.2 IP GeoLocation

IP GeoLocation is a process of determining a hosts geographical location using it's IP address. It's currently used in web applications to tailor content based on user location. There are 2 techniques used for obtaining IP GeoLocation information. One consists of gathering metadata provided by ISPs, another involves use of network topology and estimating distance to unknown nodes by measuring the delay to known nodes and triangulation.

2.2.1 Topology and delay based

This approach uses a set of so-called landmarks i.e. nodes with known location. Such landmarks can be active or passive. Active ones are used to send probes such as ICMP ping messages and passive ones are only able to reply to such probes.

Several GeoLocation methods have been proposed. GeoPing and Constrain Based Geolocation[9] suggest that there is a correlation between geographical distance to a landmark and the delay to it. By measuring the delay between several such landmarks and target hosts with unknown location they are able to create an address space that can be used to further improve the precision.

GeoCluster uses information gathered from BGP prefixes to add to locations of known landmarks.

Topology based GeoLocation proposed by [10] uses routers nearby landmarks to further improve the precision of GeoLocation.

All of these methods are limited by their choice of constraint, that is delay. However there are several factors affecting delay that are not dependant on geographical locations such as faulty nodes, load balancing, delays introduced by router processing. Additionally relying on landmarks falls short when there is insufficient number of such landmarks to determine the location with any degree of precision.[10, 9, 11] At the moment all of this techniques are in development stage and can not be used in practice.

2.2.2 Metadata based

There are several types of metadata available:

- WHOIS records

- DNS loc records
- DNS hostname parsing rules[10]

WHOIS records are maintained by network information centers such, RIPE, ARIN, APNIC and AFRINIC. The original purpose of these records was to provide contact information for persons responsible for assigned IP address ranges including location. Currently there is a number of information services using WHOIS protocol.[12] The information is provided in human-readable form and formating varies complicating automation of queries.

RFC 1876[13] suggested use of LOC field in DNS records to store information about location of hosts including latitude and longitude. This has not yet been adapted widely enough for practical use.

There are several commercial and free databases that use one or more of these techniques to provide location information for IP addresses.

NetGeo [14] is a tool by CAIDA that combines all of these techniques to create a database available online through use of Perl scripts or http requests. The information provided is extensive and specifies location on city level or even provides coordinates. Unfortunately it has not been maintained for several years and thus contains a large number of errors due to relocated or reassigned IPs.

MaxMind [15] uses WHOIS records and user entered information to create a database of location information with Country and City level and coordinates where available. The databases are updated every month. They are available in binary format and can be accessed through APIs in a number of programming languages.

One of the disadvantages of metadata based IP GeoLocation is lack of precision as the information pertaining to registration may not necessarily be the same as the actual host location. Additionally the information is assigned and gathered manually and is thus subject to human error. However the accuracy of metadata based databases on the country level is usually enough for most purposes provided they are maintained.

2.3 Load Balancing

Wikipedia[16] defines load balancing as "a technique to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, throughput, or response time". Common goals of load balancing are server affinity, scalability high availability/failover achieved through redundancy. Server affinity means that server state affects load balancing. Scalability means that the service can accommodate a varying number of users. High availability means that if one server fails the service is maintained. [17] Additionally when there are multiple possible locations of servers and clients there is an additional goal to consider, that is minimizing distance between clients and servers. This distance could be geographical or dependant on network topology such as number of hops. Some load balancing algorithms[18] can consider other issues such as:

- minimizing the cost of bandwidth
- catering to different capacity demands of the clients
- avoiding using maximum capacity of the servers to prevent overloading
- maintaining state information
- catering to client preferences in regards to servers e.g. using “stable marriage” techniques

There are several approaches to implementing load balancing:

Client based In this system load balancing is implemented in the client. In the simplest versions the clients merely maintain a set of server addresses which they chose from using some algorithm such as round robin to send the requests to each of the servers in turn or weighted round robin where the number of requests to be sent to each of the server is proportional to e.g. server capacity. More advanced versions allow for high availability and server affinity by having the clients update their serverlist from a source that is aware of the state of the servers. The disadvantage of this scheme is that it introduces overhead of updating and makes the system more complex.

DNS round robin In this scenario all the clients get is a url. In DNS system this url can be resolved to a list of IPs. When the clients poll the DNS system for an IP it returns a new one each time. The advantage of this compared to storing the list in the client is that the list is much easier to change. The system itself is also easier to setup as there is no need to design the load balancing system. Disadvantages are that by default this scenario does not allow for server affinity or failover and DNS caching may affect the results.[17] This can also be augmented by a system that is aware of server state and changes the list accordingly. The problem with that is that DNS information takes time to propagate so the reaction time is uncontrollable.

Load balancer based This method uses a hardware or software based load balancing system that acts as a front for all the servers. Clients need only know the address of this system. This system forwards the user requests to the servers. It can be aware of server state and thus provide server affinity and failover.[17] The disadvantages of this system are it’s complexity and possible reduced scalability as the system may easily become a bottleneck.

Distributed Recently some research[19] has been done concerning load balancing algorithms that involve both servers and clients to provide all of the advantages of the methods mentioned above and be more effective. This however introduces a new level of complexity.

2.4 Log Analysis

As I have already mentioned there are no published studies covering Opera mini log analysis. However log analysis for other services is covered in some detail. Some of the issues with those can be applied to Opera mini and my thesis. The service type that is closest to Opera mini are web and proxy servers. In much the same way they can serve users from all over the world that are largely anonymous, while the traffic amount can range from low to extremely high. Some of the data commonly found in Log files is source of the request, time of access, protocol, destination (i.e. web resource) and it's size.[20] The purpose of such analysis can be to summarize that information to gain quantitative knowledge about utilization and to make predictions about future development and plan server capacity. Log files often contain a large amount of data that is not relevant to the analysis which makes some degree of pruning necessary. A number of tools such as webviz and wwwstat[21, 22] have been created for this purpose. Some of those include analysis capabilities such as calculating number of requests and temporal analysis and some provide graphical representations of data.[21] As research shows the traffic on such services consists of 95% html and images[23]. This number is likely to be even higher for Opera mini due to the limitations of files mobile devices can process. [20] notes following challenges in analysis of results gained through log analysis:

- being unable to discover identities of the users with certainty,
- impossibility of gathering qualitative data such as purpose of a request
- having historical data per user being limited by servers perception of a user session which can be wrong and does not include other servers than the current
- the number of requests not reflecting the actual usage due to caching
- automated requests being indistinguishable from actual user requests
- lack of accuracy in geographical placement of users.

2.5 Statistical analysis

There are mainly 2 statistical techniques that I use in this thesis, one of them, regression is used for identifying relationships and creating predictions, the other involves describing findings using distributions.

2.5.1 Regression

Regression or curve fitting is used to investigate relationships between variables to determine whether one of them is dependent on the other.[24] The dependent variable is modeled as a function of one or more independent variables attempting to give a "best fit" of the data using the least squares method. [25]. Linear regression uses formulas of following form:

$$y = a + bt + E$$

where y is the dependent variable, t is the independent one, a is a parameter that determines the 'starting value' of the dependent variable also known as y-intercept as it is the y coordinate of a point where the graph of this function intercepts y axis. b is the parameter determining the slope of that graph and E is the "noise" representing the unaccounted factors affecting the dependent variable. Regression analysis usually assumes that E is zero on average.[24]

The analysis itself consists of finding such values for a and b that the graph of this function is as close as possible to observed values of y. This "closeness" is determined by using the least squares method. The name of the method refers to the criterion used to describe the closeness. The best fitting line is defined as the one that gives smallest sum of squared residuals i.e. differences between observed and predicted values.[26]

Thus the formula for the parameters becomes:

$$b = \frac{\Sigma(t_i - \bar{t})(y_i - \bar{y})}{\Sigma(t_i - \bar{t})^2}, ' \ a = \bar{y} - b\bar{t}$$

Where $\bar{t}$ is the mean of t values and $\bar{y}$ is the mean of y values. To measure how well the strength of correlation between t and y a correlation coefficient r is used. It can be found using the formula:

$$r = \frac{n\Sigma(ty_i) - \Sigma x_i \Sigma y_i}{\sqrt{\left[n\Sigma t_i^2 - (\Sigma t_i)^2\right]\left[n\Sigma y_i^2 - (\Sigma y_i)^2\right]}}$$

r^2 is a measure of how much of the observed data is accounted for by the function [27]

$r^2 \equiv 1 - \frac{SS_{err}}{SS_{tot}}$ where SS_{err} is the sum of squared residuals and SS_{tot} is the sum of squared deviations from the mean in the observed data.

Exponential regression

Exponential regression is used when the dependent variable is expected to change exponentially in relation to the independent variable. The function used is:

$$y = ae^{bt}$$

which using properties of logarithms can be transformed to

$$lny = lna + bt$$

Which is similar to linear regression and can be calculated as such using logarithms of y values.

Correlation coefficient is given by:

$$r = \frac{\Sigma(x_i - \bar{x})(logy_i - lo\bar{g}y)}{\sqrt{\left[\Sigma(x_i - \bar{x})^2\right]\left[\Sigma(logy_i - lo\bar{g}y)^2\right]}}$$

2.5.2 Frequency distributions

Frequency distribution is a tabulation of raw data. The data is spread between a number of ″bins“ or classes. The number of elements within boundarys of each bin is then calculated.[28] The point of this data manipulation is to simplify managing the data. The important qualities of distributions are their mean and median. Mean is the arithmetic mean of the data calculated as:

$$\bar{x} = \frac{1}{n}\Sigma x_i$$

Median is the number that separates the distribution into two parts with equal number of bins. The difference between mean and median determines whether the distribution is skewed.

Scatter is a measure of variation in the measurements i.e. width of the distribution.[29]

Chapter 3

Model and Methodology

3.1 Overview

Practical work on this thesis consisted of 3 phases: data acquisition by log filtering with IP GeoLocation and storage of resulting data in a database, preparation of data for analysis by querying the database and rearranging the data, statistical calculations and graphical analysis using gnuplot.

3.2 Equipment

3.2.1 Hardware

The equipment used in the practical work on this thesis consisted of:

- server Intel Xeon dual core 1.6GHz, 3GB ram, 2x 160GB SATA disks in raid 0, running Debian ”etch” used for phase 1 log storage and parsing
- workstation PC with 3GHz Intel Pentium, 1 GB ram, running Debian ”Lenny”

  the equipment was provided by Opera Software

3.2.2 Software

Most of the log parsing and operations on the database were done using Python 2.5. Multithreaded implementation of GeoLocation script for logs was written in Perl. MaxMind GeoLite package and it’s Perl and Python APIs were used for GeoLocation. Python SQLite module was used for creation of the database and interaction with it. Gnuplot and gnuplot.py module were used for graphical analysis.

3.3 Phase 1

3.3.1 Data source

Opera software provided me with 2 choices for data acquisition: live capture of traffic from all currently operational Opera mini servers and traffic logs from the transcoders. The advantages of live capture would have been: more current data and more controll over time variable as well as possibility to explore the loadbalancing algorithm used in more detail. The disadvantages involved included inherent problems in capturing data at 1Gbps speed such as -

- the kernel using conventional hardware may be unable to capture all the traffic resulting in packet loss and therefore incomplete data[30, 31]
- not being able to capture more than first 40 bytes of the packet(header) and thus being restricted in available metadata
- ”noise”, that is irrelevant traffic being captured alongside the traffic I was interested in.
- packet fragmentation complicating the filtering

Log files on the other hand provide complete data about each request with a lot off additional variables with the disadvantage of being restricted to one period of time and having to deal over 20 gigabytes of logfiles per day. The completeness of data was the deciding factor in choosing the logs as source of data.

3.3.2 Logs

Due to the change in log format I had to adapt to two different log structures. The difference between them was mostly in format and not in the data contained so I will only cover the most relevant one here.

Each of the log files contained the data for 1 day and 1 transcoder resulting in 54 to 56 files for each day. Each request to a transcoder resulted in 3 lines in a log file, cluster name, a format string naming the fields in the data string and the data string itself, containing 36 fields. The fields include both data originating from the transcoder such as timestamp, processing time, data size etc. and from client such as handset name, client version, language etc. There were 2 fields for source address, one being the IP that actually contacted the transcoder and optionally others that forwarded the request. The URL requested was presented exactly as received from the client with exception of URLs that transcoder failed to process which were marked by a b: infront of them.

3.3.3 Log Processing

Log processing consisted of four tasks:

1. reading of files, extraction and processing of fields

2. GeoLocation of the source IP
3. GeoLocation of the destination URL
4. adding each of the requests to database

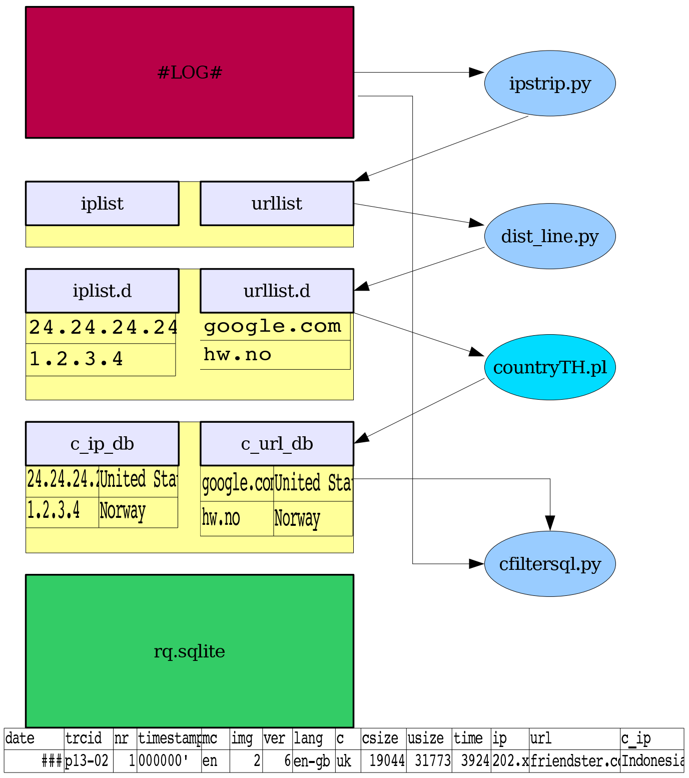
I decided to use python for log processing tasks as it is the language I am most proficient in which also provides efficient functions for dealing with textfiles while being sufficiently fast.[32] First and fourth of these tasks can be solved directly in python with the use of pysqlite.dbapi package. The second and third required use of some external GeoLocation technology. I considered 3 approaches to doing this.

- Direct WHOIS lookups
- NetGeo web database
- MaxMind GeoLite database

After researching and testing these approaches I found the first option too time-consuming and unsuitable for a large number of lookups, option 2 far too inaccurate and decided to settle on option three, see chapter 2 for more information about the alternatives. MaxMind has a python API so all four tasks could be done in a single script. However this proved to be inefficient because GeoLite lookups took considerably more time then logfile processing and database input, additionally the nature of the log files implied recurring IP and URL addresses so time could be saved by first gathering all of them, doing lookups, storing the results in a file and then using those files for lookups during processing. This line of thought resulted in using 4 scripts for log processing. Figure 2 shows the process in detail 4.4. Script ipstrip.py extracts IPs and URLs and stores them in the files iplist and urllist, dist_line.py is then run on each of those files to eliminate recurring lines, countryTH.pl then does a lookup and assigns a country to each of the addresses, cfiltersql.py then goes through the log files again and using information acquired earlier finalizes request information and inputs it into the database. Full code of those scripts can be found in Appendix A.1-4. Here I will cover some excerpts from those scripts which I find important.

Script highlights

ipstrip.py As mentioned above each request in the log will sometimes have several forwarded IPs in addition to origin IP. This can happen due to interaction with proxy servers at the users ISP/Mobile Operator. In that case origin IP would often be a reserved IP and forwarded IP i.e. IP of an interface at the ISP would have to be used. Similar situation happens when a request is passed between Opera mini servers as part of internal load balancing. Then origin IP would then be that of an Opera mini load balancer so again forwarded IP would have to be used. Several IP addresses would end up in the forwarded field if both these cases happen and/or there were multiple proxies. In all



date	trcid	nr	timestamp	mc	img	ver	lang	c	csize	usize	time	ip	url	c_ip
###	p13-02	1	000000'	en	2	6	en-gb	uk	19044	31773	3924	202.x	friendster.c	Indonesia

Figure 3.1: The log processing diagram. Arrows show input and output files.

other cases there would be no forwarded IP and origin IP would be used. Accordingly my script uses first IP in field 36 (XFF or x-forwarded) if present and field 34 otherwise. The GeoLocation of URL turned out to be the most time consuming process due to the fact that the GeoLite database only contained IPs so each URL would have to be resolved via a dns lookup first. Which is why I wanted to minimize the number of URLs that needed resolving. As a result I decided to treat URLs with country code top level domain as belonging to that country and only consider URLs with generic top level domain such as .com .net .org etc. Furthermore I decided to consider three top levels of the domain name as more then that would often result in too many duplicate lookups and using only two would cause problems with certain countries which use second level domains as an additional qualifier such as ”co.uk” and ”com.br” which are very common according to domain survey by Internet Systems Consortium.[3] A large number(5%) of URLs in the logfiles were mistyped, incomprehensible or used wrong encoding. To filter out the correct ones following regular expression was used:

```
"(?:http[s]?://|www.|[WAPwap].)(.*?)(?:/|\s|[A-Z]| :|$)"
```

This means that correct URLs had to start with either ”http(s)”, ”www” or ”wap” with domain name ended by either slash, colon, whitespace, uppercase letter or endline. This resulted in only correct domain names being output, but introduced another source of errors as about 0.3% of correct URLs would fall outside of it's scope due to the regex being unable to distinguish them from malformed ones. See appendix A.1.1 for full code.

countryTH.pl As mentioned above each of URL lookups required a dns lookup. Dns lookup time varies a lot due to network related issues such as request propagation time and dns server being busy. Processing each of those lookups serially could potentially slow the whole process down. Great improvement could be achieved by processing several URLs simultaneously so as to avoid waiting for slow requests. To achieve this I decided to use several threads. Having more experience with thread system in Perl I decided to use that here. Number of threads to be used had to be specified beforehand so I tested it with several different numbers. The greatest increase in performance happend when going from 2 to 5 threads and there was very little gain using more than 15 threads. I've also created a system to track progress of the thread by having them output number of requests processed to standard error. See appendix A.1.3 for full code

cfiltersql.py This script combines the results from the previous ones, gets additional data from the files and inputs data into an SQLite database. SQLite was chosen as it is simple, yet powerfull and also the most python-friendly database format.[33] This script extracts more data from the logs than this thesis uses. This is due to the database being used in a parallel project as well as the possibility that it and the resulting database could be used for further research into the subject. For a listing of all the fields see the script comments in appendix A.4. The script uses a table to converts between country names and top level domains using a list from IANA and ISC [3, 34]. See appendix A.1.4 for full code

cfiltersql_brief.py Later I have modified the cfilter script to adjust to changed requirements. This version was designed to make the database as small (and thus as fast) as possible, by leaving out the fields that would not be used at this time. This version is also designed for more user friendliness so it splits the timestamp into hour, minute and second fields. Skipping the URL GeoLocation made the time saved by preprocessing GeoLocation data less meaningfull so that was also sacrificed for simplicity and IP country is resolved directly. In other words this version takes a little more time to run, but does not require additional interaction. See appendix A.1.5 for full code

3.4 Phase 2

Phase 2 consisted of using the database created in phase 1 to acquire data that could be used for analysis by using sql queries. Then the data could be further processed by using python/NumPy.

Canada	p04-07	2008-01-04	544
China	p04-07	2008-01-04	2860
Costa Rica	p04-07	2008-01-04	406
Cote D'Ivoire	p04-07	2008-01-04	156
Croatia	p04-07	2008-01-04	236

Table 3.1: Query output example.

3.4.1 SQL query examples

Lacking a proper sql server I have used the available machines to execute the queries which made the process quite time consuming with queries taking from several minutes to several hours depending on complexity. The database design I implemented in phase one was necessarily simple so the queries shown here should be understandable for anyone with basic sql knowledge. Most of them consist of a select statement with an aggregate function such as count() and one or more levels of grouping to order the data appropriately. For example this statement

```
SELECT CIP,COUNT( DISTINCT IP) FROM RQ GROUP BY CIP;
```

produces a list of all countries(CIP - country for IP) in the main table (RQ) and counts the number of unique (DISTINCT) IP addresses from this country.
and

```
SELECT TRCID,CIP,DATE,HOUR,COUNT(TRCID) AS QQ FROM \
RQ GROUP BY CIP,TRCID,DATE,HOUR HAVING QQ>100;
```

Produces a four-level list where number of requests is counted for each combination of transcoderID, country, date and hour.

3.4.2 Result Processing scripts

The queries above produced results that had each combination of ordering parameters 3.1 on its own line and thus were ill suited for analysis. Whereas an appropriate format would be a multidimensional list. Which is why I created a script gprep.py that would convert the data to an appropriate form for plotting as well as filter the data so that only the points corresponding to a given criteria such as minimum number of requests or a set of countries remain. See appendix A.2.1 for full code of this script

3.5 Phase 3

In the phase 3 I used the data prepared in phase 1 and 2 to plot and analyze the data. Gnuplot was used for plotting the data. Having never used gnuplot before I had to learn it on the fly. Here I will cover several gnuplot commands which I found very important and descriptions of what they do. Originally I used version 4 of gnuplot for Debian “Etch” distribution, but later I discovered that some of functions I needed were not available there so I switched to version 4.2 and upgraded the debian system I used to “Lenny” distribution.

The gnuplot commands I use are either “set” commands that alter various plot parameters or “plot” commands that output the data to a selected plot format.

3.5.1 Set commands

The most important set commands alter the way data is displayed. For example “set style data histograms” shows the data as a bar chart, while replacing histograms with points would just plot a dot for each datapoint. The style can be further altered by e.g. using “set style histogram rowstacked”. Each option produces a completely different way to visualize the data choosing could be crucial for pointing out data properties. Each option also has specific requirements for the format of input files.

While with the defaults it is usually possible to plot the basic things just using the “set range [start:end]” command sometimes it is necessary to format the data differently such is in the case of timeseries. Following 3 commands deal with dataformat on an axis.

```
set xdata time
set timefmt '%Y-%m-%d-%H'
set format x '%d %h
```

The 1st command specifies that the data shall be viewed as time, 2nd sets the format for input data and the 3rd sets the format for the output.

3.5.2 plot commands

There are 2 types of plot commands “plot” for 2 dimensional plots and “splot” for 3 dimensional plots. I have found 2 dimensional plots to be best suited for the discrete data I had. Plot commands specify the file or function to plot, the columns in the file to use, the calculations to do on them and optionally the style parameters for each of the subplots. For example following command plots the file 'cph2.gp' using column 7 as lines with points with linetype 1 linecolor 1 and linewidth 2. It also plots the column 3 as vertical lines (impulses) and column 13 minus column 2 multiplied with column 3 using defaults.

```
plot 'cph2.gp' u 7 w linespoints'' \
lt 1  lc 1  lw 2, u 3 with impulses,'' u ($13-$2*$3)
```

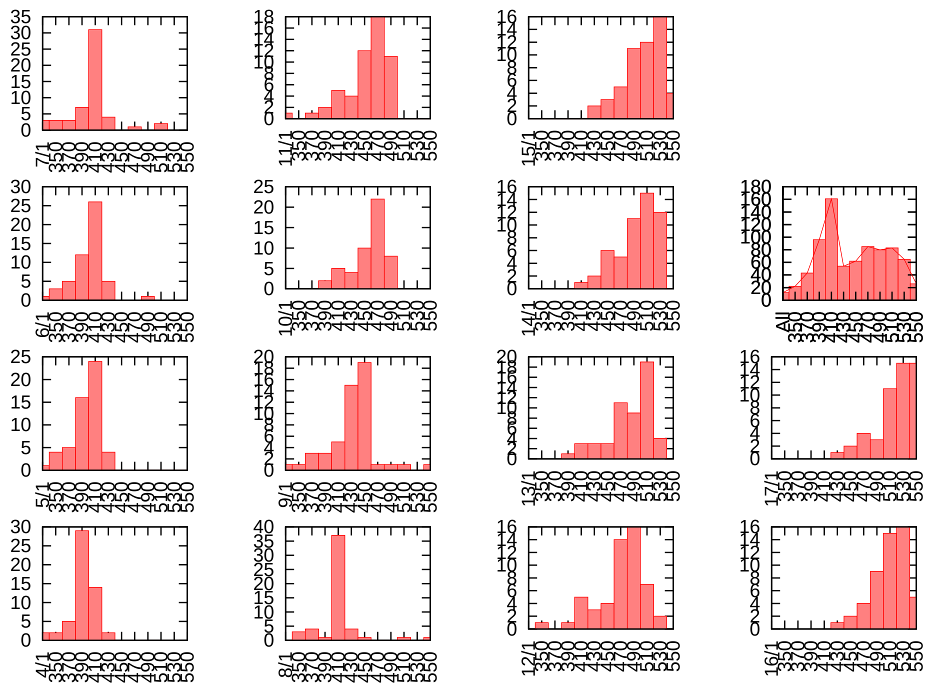


Figure 3.2: Daily distributions of requests per transcoder

3.6 Model

To be able to analyze how the requests are spread between transcoders I decided to see if country of origin had any effect on distribution of requests between transcoders. The problem with that is that graphing all of the data I had for the totals for each specific country would multiply the amount of data shown on the plot by a factor of 200 which would make it completely unreadable. Which is why I needed to create a model that would describe the data using just a couple of variables. That would be easy to plot for a large number of countries and still show the important trends.

To find such variables I decided to first plot the number of requests per transcoder each day as a statistical distribution. That is to divide transcoders into several groups according to the number of requests they processed and count the number of transcoders in each group. Then I could plot the results as a histogram for each day.

As calculating distribution data by hand did not seem reasonable I had to either find a tool that would do this or program it myself. The solution ended up being something in between. I used NumPy/SciPy package for python that contained multidimensional arrays and statistical functions to be performed on them and wrote a script (distrib.py, see appendix A.3) that would adapt my data to the proper format, run an appropriate function and then output the results to textfiles and gnuplot.

The numpy function I used ended up being numpy.histogram which takes data and number/specifications of ”bins” (groups) and returns the number of datapoints that fit each of the bins. Knowing the approximate totals of number of requests per transcoder I decided to specify the bins manually as range(330000,570000,20000), that is starting with 330000 and then divided into 12 even parts of 20000 each up to 570000. All the transcoders with 0 to 330000 requests would then end up in the first bin, 330000 to 350000 in the second 370000 to 390000 in the third and so on up until the last bin that would contain the ones with 550000 and above.

The end result of my script was a plot for each of the 14 days and one for a sum of results from all days for each bin such as shown on figure 3.2
Since the plots were not very complex they could be scaled down quite well so I decided to use multiplot so that all of the plots appear side by side in a 4 by 4 matrix as can be seen on figure X. X-axis shows the bins(marked by their lower edge) and y-axis the number of transcoders in each bins or in other words the number of transcoders with that many requests. The date of each subplot is stamped in its bottom left corner. Due to the way plotting script interacts with gnuplot first subplot (4/1/08) is situated at the bottom left corner of the matrix and the next is one step up in the same column, 5th sublot(8/1/08) is at the bottom of the second column and so on. The script(distrib.py) is included in the appendix.

Since the height of the bars tells us how many transcoders have processed an almost equal number of requests and the total number of requests is not constant the height at a fixed x value would not be a descriptive variable. However the the maximum height is still interesting to look at because it tells us something about how effective the loadbalancing algorithm was when compared to the total number of transcoders. The higher the column the more evenly are the requests spread out. The ideal case would be one bar with the height of 56. This however doesn’t seem to be the case most of the time. On the example plot the height seems to be decreasing over time.

However the height does not tell us everything because the bins are fixed. So several transcoders that have approximately even number of requests could end up in different bins if the are very close to the edge of a bin. Which is why another variable was needed. I decided to use width of the distribution as such a variable measured as the x position of the highest non-zero bin minus x position of the lowest non-zero bin divided by the bin width. The result roughly corresponds to scatter and describes the spread of the distribution on the x axis. Coupled with the first variable, maximum height this can describe a distribution in a more controlled fashion. For example consider 2 cases: one with all the transcoders divided evenly between 2 bins and the other with half of them being in one bin and the other half being spread over 5 bins. The maximum height would be the same in both cases even tho they are quite different. The width would then show that in case 2 the distribution is far more spread and thus the load balancing algorithm did not do as good a job.

The width and the height are not independent of eachother. At first glance we can see that the height limits the width in the sense that width can not be bigger than total number of transcoders minus height or $w < 56 - h$. Similarly we can show that the width limits the height or $56 - w < h$ in much the same way. This makes sense if there are no "holes" in the distribution, it is continuous or in other words or in other words there are no bins between the lowest and the highest with zero transcoders in them. However in reality there seems to be a number of holes in the distributions caused by various cluster issues. Which would mean that the equation changes so that the only limitation is that $w > 1 <=> h < 56$.

Calculating width and height for a number of contries required modifying the model to adjust to the varying number of requests. The bins had to be specified differently for each country and yet be comparable with each other. The solution I chose was to increase the number of bins to 20 and divide the results equally between 0 and the maximum for that country. the script that processes the raw SQLite output files and does the width and height calculation is also included in the appendix A.2.2 (distrib_gp.py)

Chapter 4

Results

This chapter presents the results produced in this thesis. There are 2 parts, first showing where the traffic processed by the cluster originates which is usefull for making decision with regards to global load balancing such as placement of future clusters and assigning the clients to clusters. Second part shows how the requests are distributed between the transcoders in the cluster and attempts to make a connection to the origin country data. This can help make decisions regarding internal load balancing within the cluster and general cluster maintenance.

4.1 Country of Origin

The first research goal suggested by Opera Software was to find out the origins of the incoming requests. With the help of the scripts outlined in previous chapter I have measured the totals for each of the available days. The results presented as a histogram are shown on Figure 4.1. To keep the plot readable only the countrys with over 100 thousand requests per day are shown. For each country on the x axis the plot shows 14 bars, one for each day sorted chronologically so that the development during the 2 weeks can be seen. As the differences in number of requests per country are quite large I chose to use logarithmic scale on y axis for this plot. The plot speaks for itself, but I will outline the most important trends. Russian Federation is the biggest source of traffic with 8 to 12 million requests per day. Next in line are India and Indonesia with 2 to 2,5 million, then Ukraine, South Africa and USA with around 1 million.

The measured totals for traffic differ each day. In fact the total traffic seems to be growing. To check if the shares of traffic stay the same for the countries I created table 4.1. Of regard to space only the top 9 countries are included here and country codes are used as titles. Each country's share of requests in the total for each day is shown together with arithmetic mean value and standard deviation in the bottom. The requests coming from countries below the top 9 are summed up in the "others" column. The shares seem mostly stable. Russia has the largest standard deviation of 1.36%. Looking at the data in the table the

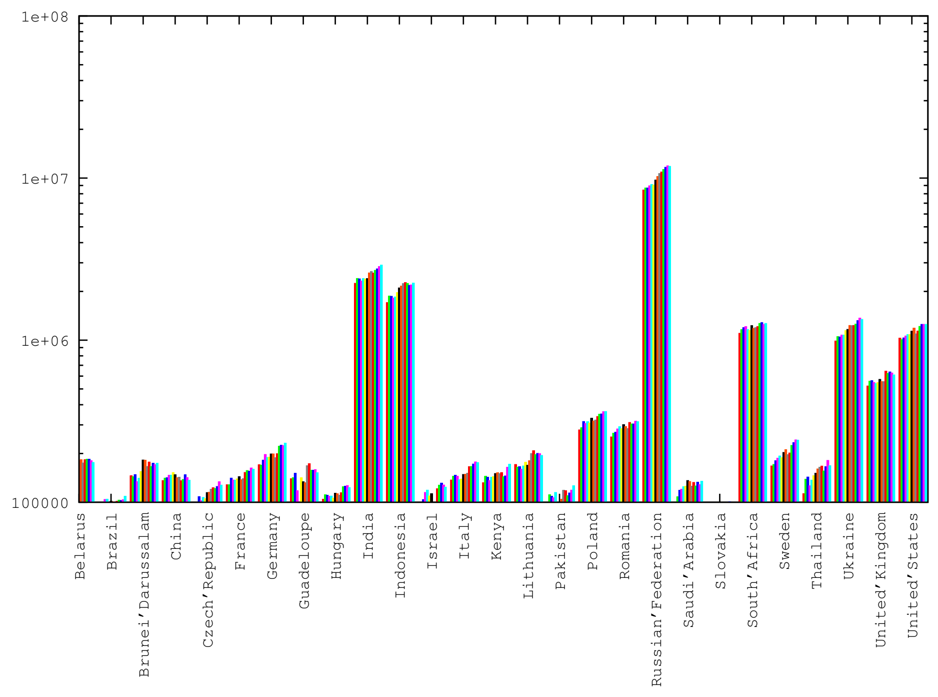


Figure 4.1: Totals for number of requests.

share of requests from Russia seems to be growing until the 17th. Figure 4.2 helps visualize this data showing percentages for the countries as a histogram and the scatter in the measurements as errorbars.

Normalization of results

The results for distribution of requests between countries of origin seem somewhat surprising. In an effort to explain them I compared them with statistics for population count from Wikipedia number of mobile phone subscribers obtained from Computer Industry Almanac [35], eurostat[36] and others [37, 38]. Note that Chinese users have a dedicated cluster in China. Unfortunately the latest available data is from 2005-2006 and thus 1 to 3 years old which a lot considering the rates of growth indicated by [36, 37, 38]. This data is summarized in table 4.2

I have then used this data to normalize number of requests, dividing request total by population and by number of subscriptions. The results are ploted on figure 4.3. As we can see the differences in population and mobile phone penetration do not change the picture that much with regard to relatively high number of requests from Russia and Ukraine. For India and Indonesia we can see that while the mobile penetration in those countries in relatively small the use of Opera mini amongst mobile users is quite high. The opposite is true for USA. An interesting thing to note on this plot is the difference between the 2 bars for each country as higher difference implies higher potential for growth in number of requests as the mobile phone penetration increases.

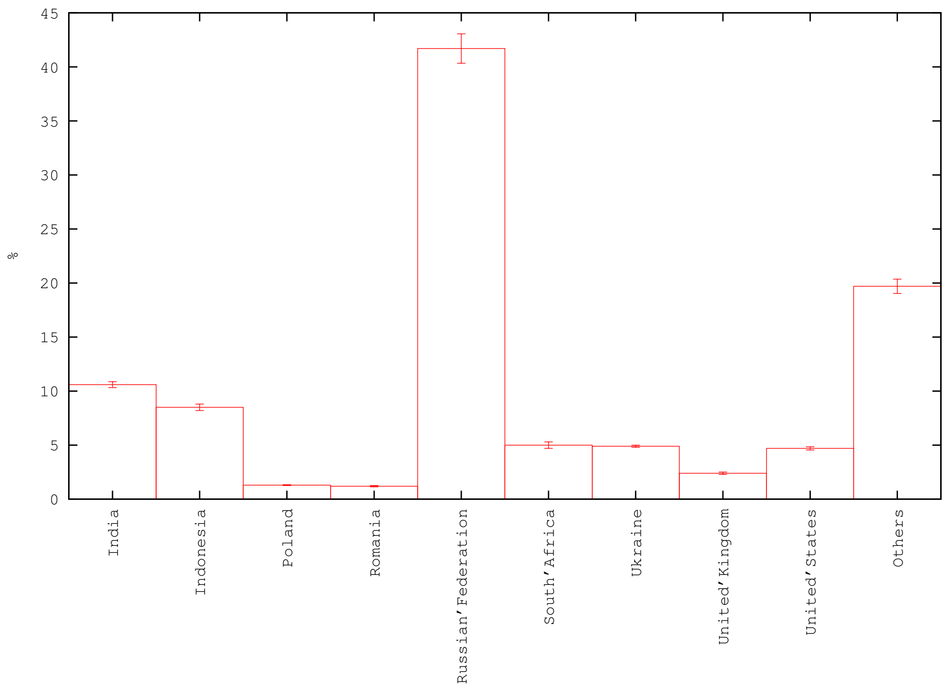


Figure 4.2: Mean percentage of requests for the top 9 countries and the rest with scatter as errorbars.

Date	IN	ID	PL	RO	RU	SA	UA	UK	US	Others
04/01	10.8%	8.2%	1.3%	1.2%	40.5%	5.3%	4.8%	2.5%	5.0%	20.4%
05/01	11.1%	8.6%	1.3%	1.2%	40.1%	5.3%	4.8%	2.6%	4.6%	20.2%
06/01	10.9%	8.5%	1.4%	1.2%	39.7%	5.5%	4.8%	2.6%	4.7%	20.7%
07/01	10.5%	8.2%	1.4%	1.3%	40.6%	5.5%	4.9%	2.5%	4.8%	20.3%
08/01	10.7%	8.3%	1.4%	1.3%	40.9%	5.2%	4.8%	2.4%	4.8%	20.1%
09/01	10.6%	8.7%	1.4%	1.3%	40.3%	5.1%	5.1%	2.4%	4.8%	20.3%
10/01	10.1%	8.9%	1.4%	1.3%	41.0%	5.2%	4.9%	2.4%	4.8%	20.0%
11/01	10.6%	8.8%	1.3%	1.2%	41.6%	4.8%	5.0%	2.3%	4.8%	19.6%
12/01	10.6%	8.9%	1.3%	1.1%	42.6%	4.8%	4.9%	2.2%	4.4%	19.3%
13/01	10.2%	8.9%	1.3%	1.2%	42.8%	4.7%	4.8%	2.5%	4.5%	19.1%
14/01	10.3%	8.5%	1.3%	1.2%	43.1%	4.8%	4.8%	2.4%	4.6%	18.9%
15/01	10.3%	8.1%	1.3%	1.1%	43.5%	4.8%	4.9%	2.4%	4.7%	18.9%
16/01	10.4%	8.0%	1.3%	1.2%	43.7%	4.6%	5.0%	2.3%	4.6%	18.9%
17/01	10.7%	8.3%	1.3%	1.2%	43.4%	4.7%	4.9%	2.2%	4.6%	18.8%
Mean	10.6%	8.5%	1.3%	1.2%	41.7%	5.0%	4.9%	2.4%	4.7%	19.7%
Stddv	0.27%	0.30%	0.04%	0.06%	1.36%	0.30%	0.10%	0.11%	0.15%	0.66%

Table 4.1: Distribution of requests between the top 9 countries and the rest with mean and standard deviation.

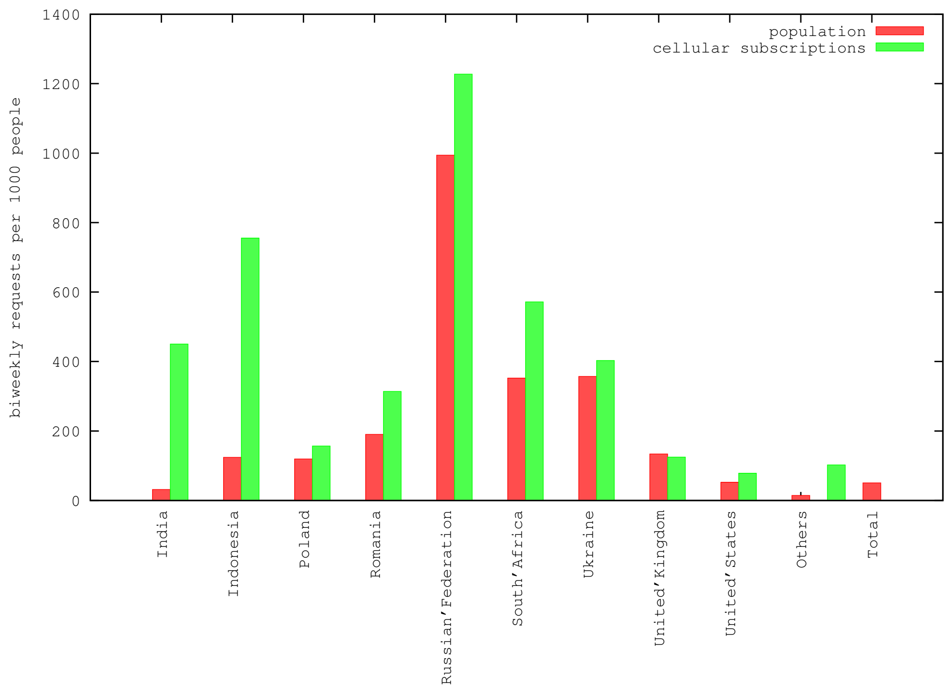


Figure 4.3: Number of requests per country for 2 weeks normalized by population and number of cellular subscribtions.

Country	IN	ID	PL	RO	RU	SA	UA	UK	US	Others	Total
Rq.tot	35.6	28.7	4.6	4.1	141.1	16.9	16.5	8.1	15.8	66.2	337.6
Population	1132.8	231.6	38.1	21.4	142.0	47.9	46.3	60.6	304.0	4646.4	6671.2
Subscr.	79	38	29	13	115	29.5	41	65	202		3300
Penetration	0.07	0.16	0.76	0.61	0.81	0.62	0.88	1.07	0.66		0.49

Table 4.2: Number of requests over 2 weeks,population, number of subscriptions (millions) and mobile penetration as number of subscriptions divided by population.

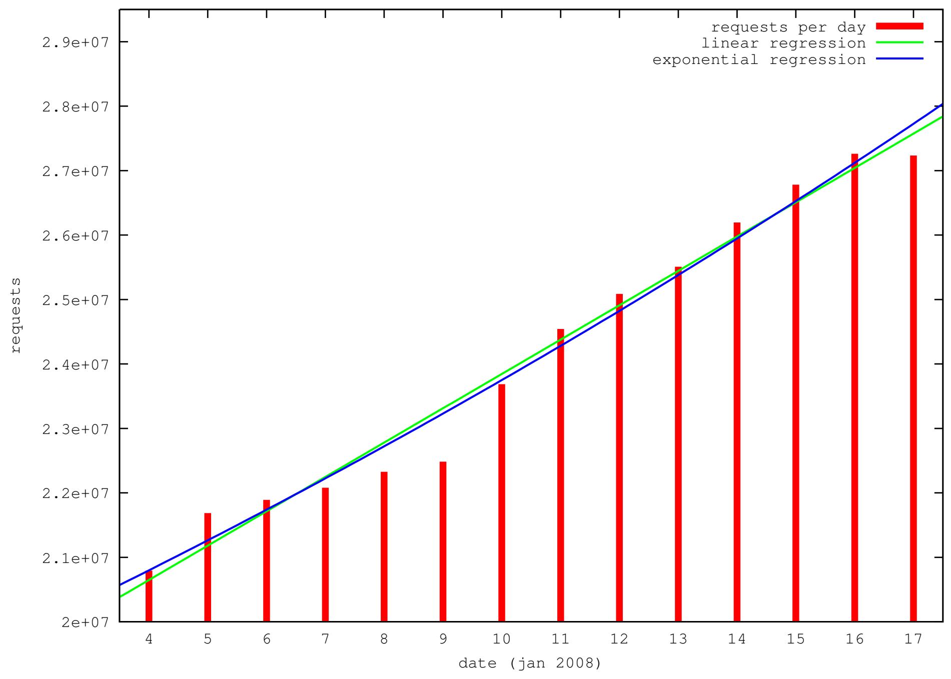


Figure 4.4: Totals for number of requests with regression.

4.1.1 Development of requests totals

While looking at numbers for daily totals I have noticed that the total number of requests was growing. Here I will take another look at that tendency. Figure 4.4 gives an overview of the traffic development. The days, from 4th to 17th of January are on the x axis and the y axis shows the number of requests. Here we can see that after an initial hop of 1 million requests or 5% from 4th to 5th the traffic is growing slowly until the 9th then the rate of growth increases and the traffic keeps growing until the 17th when the growth suddenly stops. The total growth over those 2 weeks is over 7 million requests per day or 25%

In order to be able to predict the future rate of growth I have applied linear and exponential regression to the results using the formulas described in chapter 2. The results were as follows:

- Linear: intercept: a=18517474.62 , slope: b=532785.27
- Exponential: intercept: log a= 16.76 , a=19036838.35 ,slope: b=0.02

The correlation coefficients are:

- Linear: $r = 0.9878$ $r^2 = 0.9758$
- Exponential: $r = 0.9883$ $r^2 = 0.9768$

The exponential coefficient is slightly higher so the exponential regression is a better fit. In other words when trying to extrapolate future growth the exponential function should be closer. However the difference is not significant

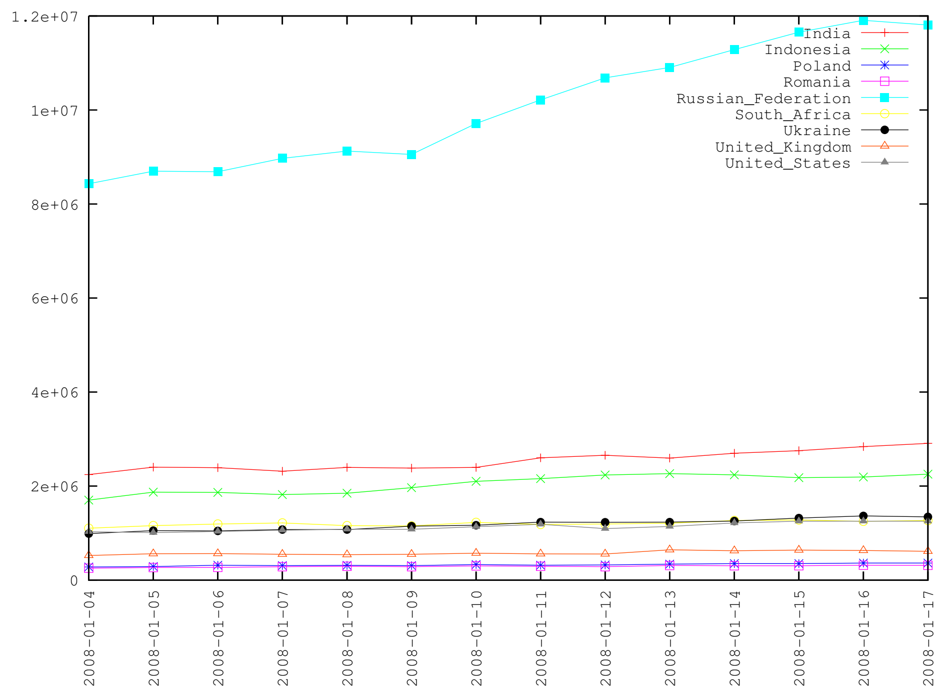


Figure 4.5: Development of number of requests per country.

unless one attempts to make predictions very far into the future which is ill advised based on just 2 weeks of data.

The next step is to look at the same data for the countries with most requests. That is what figure 4.5 shows. To be able to see the development for each country on the same plot I used lines with points instead of histogram representation here otherwise this plot is similar to the previous one. The cyan line represents the traffic from Russia which is the dominant source of traffic with approximately 40% of all requests. In fact the shape of this graph strongly resembles previous figure. Specifically the sudden flattening out of the totals graph corresponds to a decrease in traffic from Russia on the 17th. Most countries show a stable increase in traffic over the whole duration. As can be seen from the table 1 The growth is not proportional to the country's share of the totals as for example in the case of Russia and Ukraine their share is increasing from 40.6% and 4.8% on the 4th to 43.7% and 5.0% on the 16th respectively and drops over the next 2 days. So far I have been unable to find any particular reason such as a flash-crowd event that this development can be attributed to.

4.1.2 Hourly variations

Counting all the requests processed by the cluster each hour in the duration of 2 weeks produced the results shown in Figure 4.6. On this graph we can see an oscillating pattern. The traffic is at its lowest late at night (3-5 a.m.) and at it's highest in the evening (6-8 p.m.). This is somewhat surprising considering

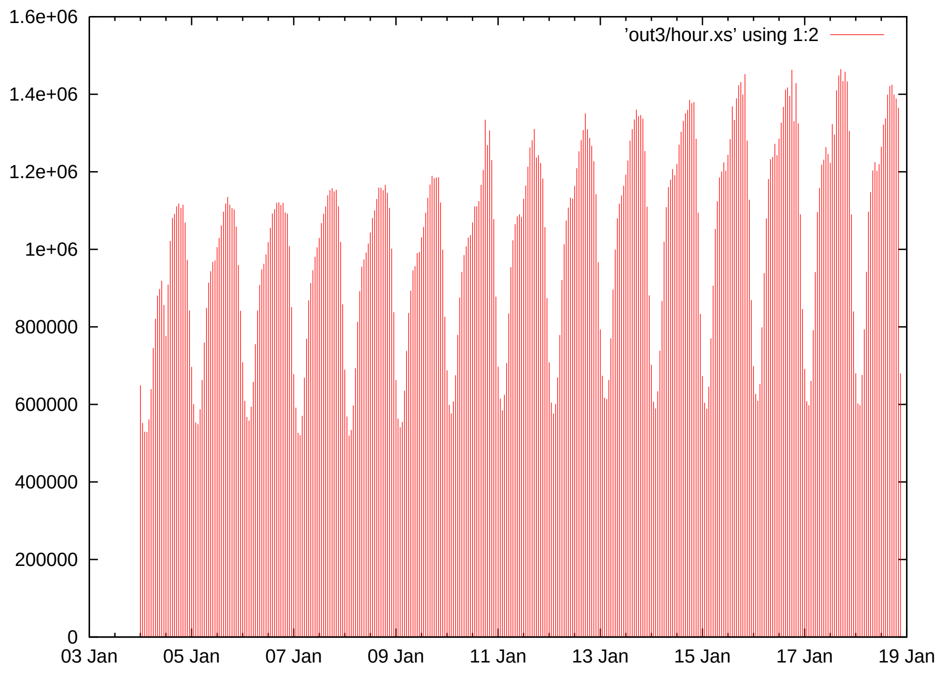


Figure 4.6: Number of requests per hour for the whole cluster

that the cluster serves users in all timezones. One reason for this could be that some timezones are overrepresented.

Again we can see that the maximum number of requests per hour is growing in the observed timeframe. The growth is not uniform. Particularly the number of requests seems to be growing faster during peak hours than during the rest of the day. This phenomenon is very interesting form the global load balancing point of view since it's the peak that can eventually be more than the servers/link can manage and the need to use the capacity at another location arises. On the other hand countries that peak at approximately the same time or at least have significant amount of traffic at the peak time for other countries could be separated to different clusters if it is geographically feasible.

To answer such questions as how to divide countries between transcoder clusters, which countries would interfere the least with each other and are least likely to have peak at the same time it is usefull to know how the traffic is distributed between countries on the same hourly basis. Figure 4.7 shows the hourly variations of traffic for the 11 countries which are origins of at least 10000 requests per hour. Total number of requests is also shown (top green line) Russia (cyan line) averages about half of the total traffic so it's not surprising that the shapes of the blue line and the red line are quite similar. However there is a difference The graph for Russia is much steeper and the maximum comes later. In other words there is a country or rather several countries whose graphs have a similar shape to the one for Russia, but are placed more to the

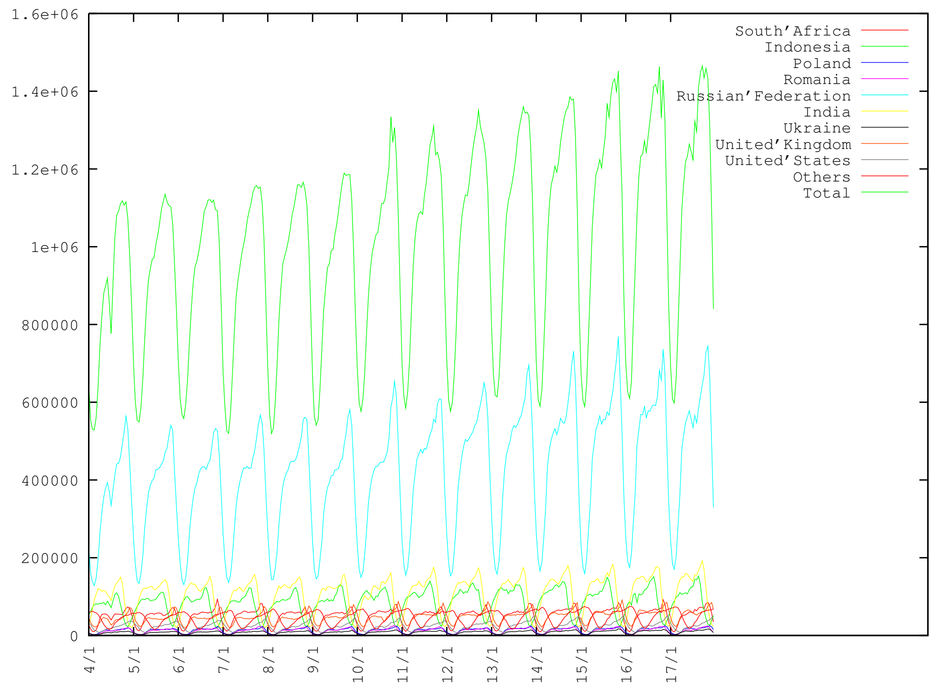


Figure 4.7: Number of requests per hour per country.

left on the x-axis. India and Indonesia seem to be showing a pattern that is remarkably close to that of Russia, albeit more even and smooth so they could be the ones affecting the totals graph. It seems that among the countries shown here only USA has a maximum at a time when the totals graph is at a minimum. That means that the traffic from USA could easily share a cluster with any of the other countries here.

In an effort to look further into this I created a stacked version of this plot as seen in figure 4.8. This plot is a lot like figure 4.7, but the difference is that the number of request from each previous country is added to the next so that they appear to be standing on top of eachother. The totals come last allowing me to see what part of total each country's contribution represents as well as the contribution from the countries not specifically mentioned as the difference between the top line of total and the top line of the second to last graph(United States). What we see here is that most of the countries here seem to have a considerable number of requests situated near the peak for the total and the combination of requests from India, Indonesia and South Africa smoothens the data from Russia and move the total to an earlier hour. However a large share of their traffic still comes during the peak time for Russia, so moving them to a different cluster would ease the total peak load.

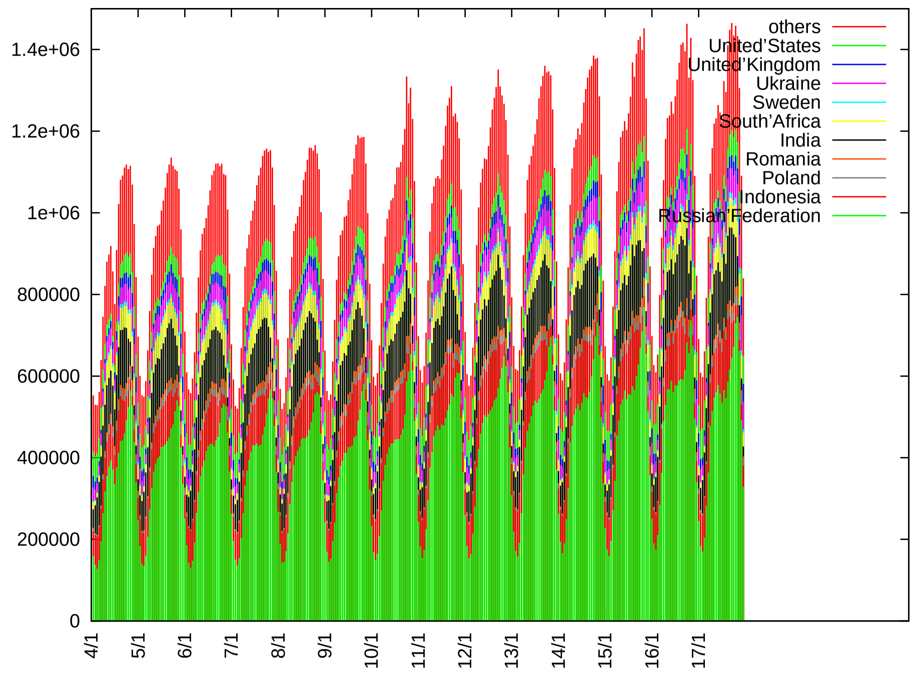


Figure 4.8: Number of requests per hour per country, stacked.

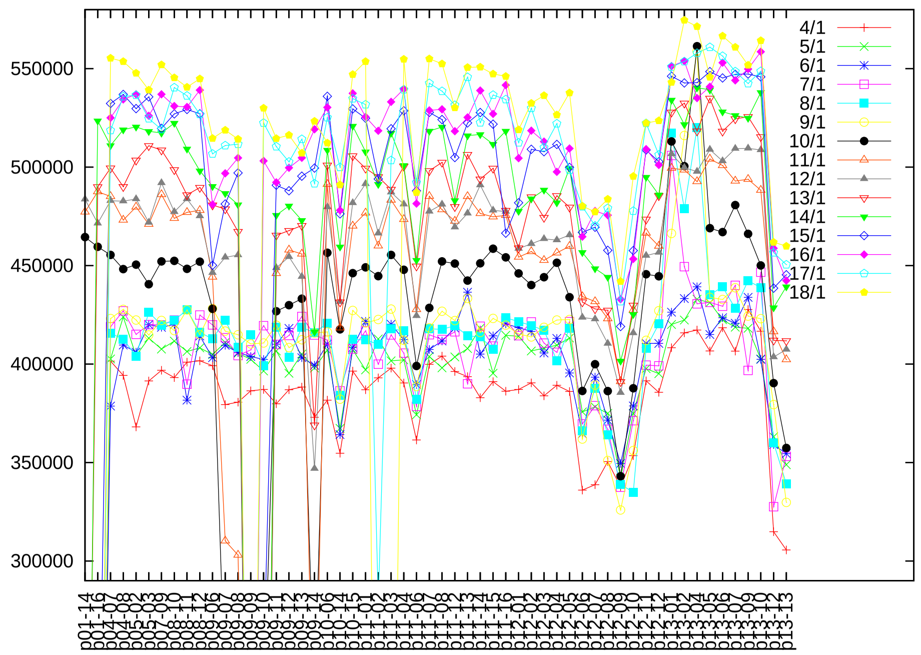


Figure 4.9: Number of requests per transcoder from 4/1/08 till 17/1/08

4.2 Transcoders

Figure 4.9 shows the number of requests each transcoder has processed during each of the days in the available data. The data was obtained from the main table in the database using a ”group by date, transcoder id”. The x axis shows transcoders. (The labels are transcoder names.) For each of the 56 transcoders 1 point for each day is plotted for a total of 14 points. The y coordinate of those points represents number of requests processed that day. This way each graph on the plot shows how the requests were distributed between transcoders that day.

There are several things this plot tells us. Some of the graphs dip below visible area for several days(particularly p1 and p9 series). That is generally a result of some transcoders having processed very few or none requests at all that day. As far as I'm informed the reason for that is downtime/maintenance on single servers in the cluster.

Considering the load balancing done for the requests I expected the number of requests to be more or less even amongst the transcoders, however this plot shows something different. Most of the transcoders are within 10% (50000 requests) of the median for each day and some show even greater deviations, particularly p12-6 - p12-10 range being 20% below median most of the time and p13-2 - p13-5 are often above median by the same amount. Since there is no difference in hardware, links or setup of the transcoders the only thing affecting distribution of requests between the transcoders is the load balancing algorithm. The memory effect could account for these deviations, but the number of recurring requests from the same clients would have to be significant.

The limitation of this plot is that we do not know if the algorithm chose to distribute the requests this way due to the memory effect or some other factor which could for instance be the total data transfered or the time/CPU-time required to process the requests. However the plot of data transfered showed the same pattern so that is probably not the reason.

There is another clear trend on this plot, the graph for each of the days is for the most part above the graph for all the previous days. The growth in number of requests shown earlier is affecting the plot.. Particularly noticeable is that a 3rd of total increase happened in a jump of about 50000 requests between 9/1 and 11/1 while several transcoders show even higher increase. The jump coincides with 2 transcoders being down, which would suggest an increase of load on the other transcoders, tho only 2 of them being down should result in a much smaller increase.

Figure 4.10 shows same data as Figure 1 arranged in a stacked histogram. This way is much easier to see the total number of requests per transcoder

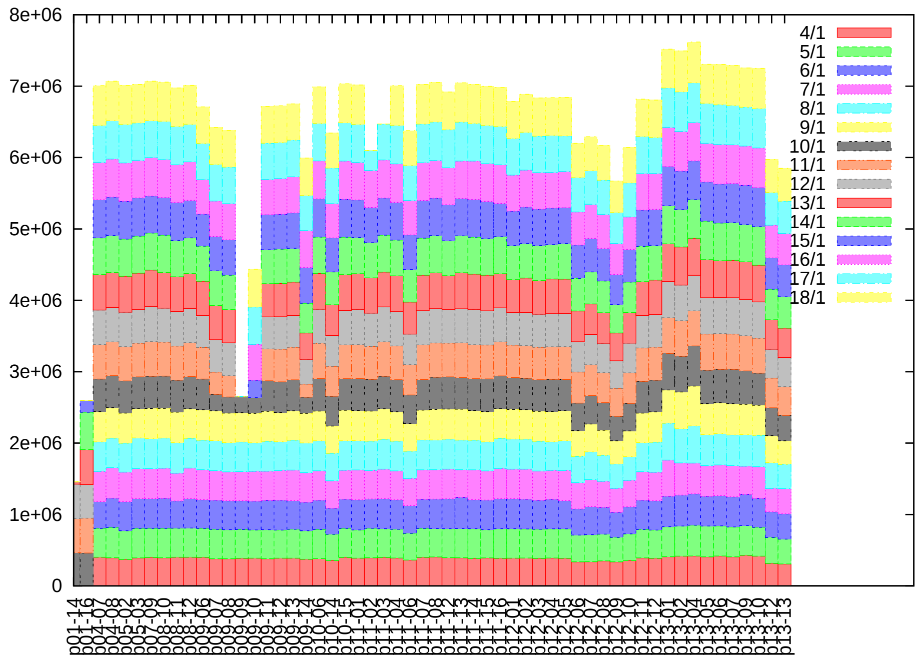


Figure 4.10: Number of requests per hour per country, stacked.

processed during the whole observed period as well as relative number of requests processed per day. We can see that for the total number of requests processed during this week some transcoders show a deviation of over 1 million requests compared to the mean(Only considering the transcoders that were up for the entire observed period). That is almost twice the number of requests processed by a single transcoder per day which shows the potential for improvement. We can also see that the bars for the “underperforming transcoders” are smaller on the 9th and 10th than on the previous days which can further explain the drastic jump in load on the other transcoders noted above.

4.2.1 Transcoder request distributions

Figure 4.11 shows the results of plotting both height and width of totals distribution. Green bars show height measured in transcoders and red bars show width measured in bins. If we divide the plot into 3 equal parts we can notice certain trends. In the first part of the plot the height is quite high, between 30 and 40 transcoders which signifies well-balanced distribution, however the width is growing during this period meaning that destabilization is in process as at least some of the transcoders are deviating even though most are in line. In the second part we can see that the width stays almost the same, but the height is decreasing meaning more and more transcoders get uneven load. Third part looks a lot like second part, but is far more stable which lead me to conclusion that the cluster is converging towards a somewhat uneven but stable pattern.

4.2.2 Distribution Height an Width per Country

The next step is to go deeper and look at similar graphs for the traffic generated in each country. This is shown on figure X and figure Y. The plot for the top 16 countries had to be split in 2 different ones because the graphs had a tendency to overlap. With the limited number of colors available this should make the plots more readable. The x axis shows days. The y axis shows the height of distribution on the top part of the plot and the width on the bottom for each of the 14 days. The number of bins in this calculation was 20 which also limits the maximum width possible for this data to 20. The maximum possible height is still limited by the total number of transcoders, which is 56.

Looking at the height we can see that it is very stable for some countries where the graph is practically flat and somewhat more varying for others, with deviations of 5-10 transcoders per bin. The most stable ones are also often the highest ones such as Lithuania, Sweden, Belarus and Kazakhstan. Those are also the ones with lower total number of requests. In a way this makes sense as it is easier for the algorithm to find capacity to handle the requests on more transcoders when the demand for the capacity is not to high. The stability in height may also imply stability in total number of requests and distribution of recurring and new requests. For some of the countries, particularly Russia, India, South Africa, Sweden and Romania the height seems to start relatively high, between 25 and 35 and decline somewhat over time to around 20-25 which is similar to what we have seen in the plot for the totals. Considering that the first 3 are also the ones with top number of requests they could be the ones causing the similar pattern on that graph.

There are several interesting things about width as well. One is that many of the graphs are quite stable so that the width is also the same on the 18 as it was on the 4th. Several graphs however break this pattern exhibiting a number of large (up to 10 bins wide) hops. Germany and Kazakhstan are particularly drastic. It's worth to note that all of the countries with this "jumpy" pattern are the countries on the bottom of this list with regard to total number of requests. This means that some transcoders may be very close to the threshhold for number of requests processed set for this plot. This way the width would drop as some transcoders near the bottom disappear and grow as they go up in number of requests. However checking with the height graph speaks against this explanation as all of the countries with this pattern seem to have high values of height throughout the plot. A different explanation may be that this happens in connection with small changes in the cluster layout such as several transcoders going down and up in the middle of a day, as observed earlier. This would cause the distribution to have an outlieing bar with one or more transcoders with relatively low number of requests while the transcoder is up. That would make the model assign a high number to the width parameter, even though most of transcoders are actually evenly loaded. Then as the transcoder goes down again the outlieing bar disappears and the width drops dramatically. Something that speaks for this explanation is the drop in width on the 10th that most countries exhibit. As we have seen earlier several changes in the cluster happened on the 10th with transcoders being down which has obviously affected the width.

A notable phenomenon on the width graph is the correlation between width and number of requests in the sense that all the top countries in the number of requests, Russia, India and Indonesia are also topping the width plot having maximum or very high number of bins. This can imply several things. As we can see later most of the requests from these countries come at the same time, during the peak for the cluster so that saturation can affect the evenness of distribution. In other words if the algorithm runs out of room on several transcoders it will have to use other transcoders which can then skew the per country distribution while not affecting totals all that much. Another explanation might be that these countries have a high number of recurring clients so the memory effect would be very influential, which in combination with high load would also cause higher width. While all the new requests would be evenly spread out and have low width, the recurring requests would stick to the transcoder they were first served by. So if the first of the recurring requests happened during peak they are more likely to be unevenly distributed and the unevenness can then increase each peak. Russia is a special case in this respect due to a factor unrelated to the load balancing algorithm. I have found out that in Russia there is a community of Opera mini users at [39] that utilize a modified version of Opera mini that lets them pick a particular transcoder and use it regardless of its load or memory. It is impossible to estimate the size of this community and how much of an effect it has from the data I have available, but it is definitely something to keep in mind.

Height and width plotted side by side show the development of both over time quite well. However it is not as easy to see how those to values can affect each other. To consider the connections between width and height other than those imposed by the model I decided to plot them against each other similar to a regression plot. Figure X shows the result. For each value of height (x axis) it's width is plotted (y axis). The resulting dots are colored according to country of origin. It is obvious that the height and width are not linearly dependent. The usefulness of this plot lies in being able to identify 4 distinct groups in the resulting points. Group 1 consists solely of Romania which is a unique case where height is quite low and width is medium low. This means that distribution is not even, but not too spread without oulieing cases. This leads me too believe that this is just unevenness caused by the memory effect. All dots are grouped together so the situation is stable.

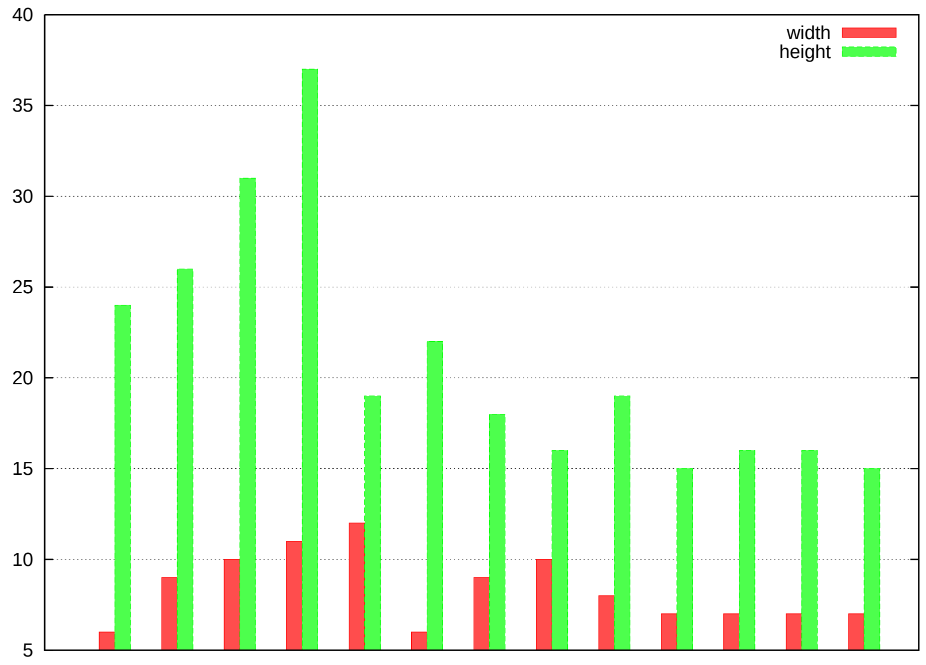


Figure 4.11: Height and width of the request distribution.

The next group is bigger. It contains most of Ukraine, USA, UK, Poland and South Africa. In this group the height is low and the width is high, but not maxed out. This means that the requests are spread out very unevenly. The next group is situated on a horizontal line at 20. 20 is the maximum number of bins which means that countries in group three have extremely high width. The countries in question are Russia, India and Indonesia which are also the ones with highest total number of requests. This implies that there is a connection between number of requests and width. The height in this group varies from medium to very high in case of Russia, which leads me to believe that the extremely high width might be cause by outlier cases due to transcoders going up and down. The last group contains Lithuania, Belarus Germany and Kazakhstan. In this group width is medium to very low and the height is quite high. This means that requests from this group are most evenly distributed around the cluster. This group is also on the lower part of the top 15 list which again leads to conclusion that it is easier to achieve even distribution when the number of requests is lower.

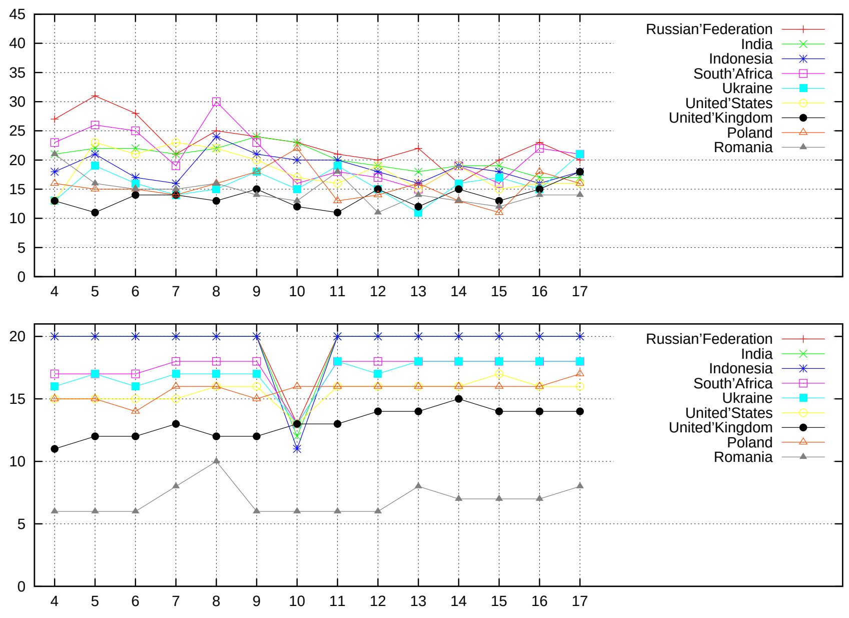


Figure 4.12: Height and width of the request distribution per country. Top 9 countries sorted by total number of requests.

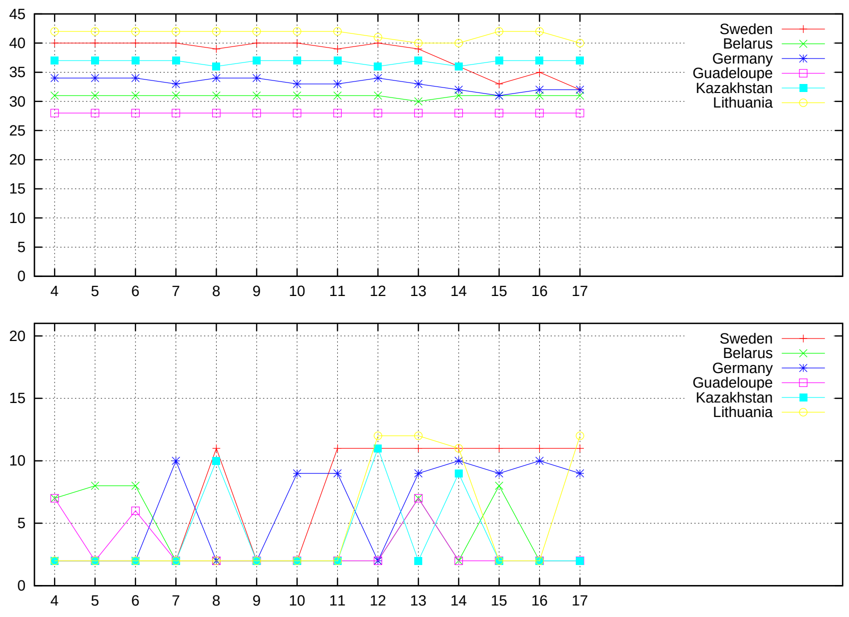


Figure 4.13: Height and width of the request distribution per country, part2.

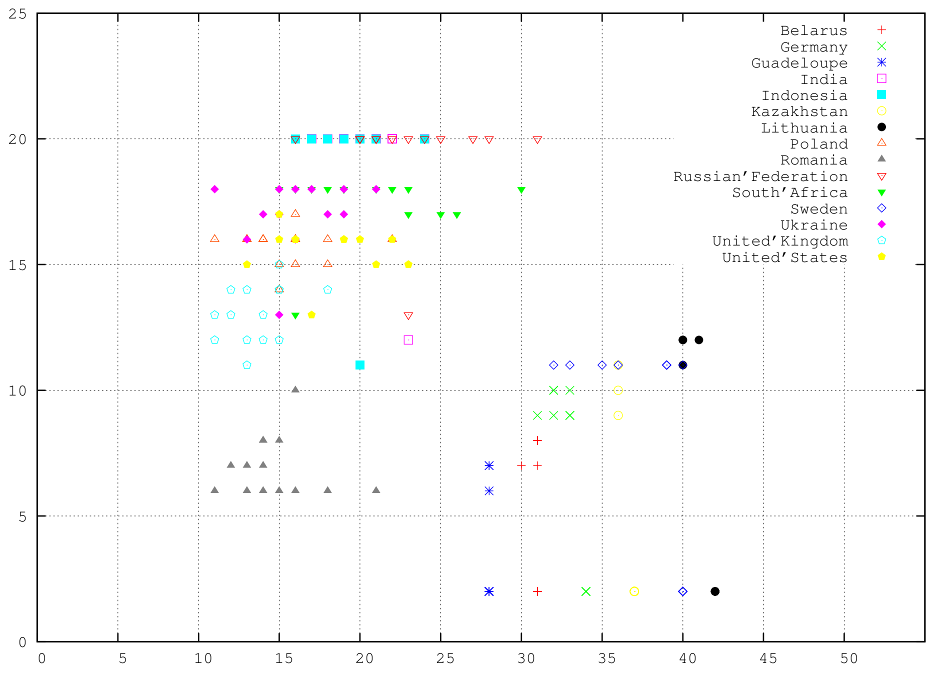


Figure 4.14: Height vs width of the request distribution per country.

Chapter 5

Conclusions and Discussion

5.1 Discussion of results

In this thesis I was going to analyze the data available in the Opera mini log files and see if the results of analysis could have any significance for decisions regarding such cluster management issues as finding appropriate location for new clusters and configuring load balancing. After exploring several directions of the analysis my results contain several finding which could indeed be relevant for such decisions. In this section I will recap these results and discuss their significance. In the first part of my work I have looked into graphing countries where the traffic to Opera mini originates. This has direct relation to the cluster placement decisions. It seems reasonable that the clusters should be geographically close to the country they are serving most of the time. According to my results over 40% of traffic comes from Russian Federation with and another 20% from it's closest neighbors such as Ukraine and Belarus. This makes Russia a good location for a cluster. Something to note is that Russia is reasonably close to the current cluster located in Norway so it could prove satisfactory for handling that traffic provided the capacity is scaled to meet the demand or that some other clusters are created to relieve this one of the traffic from some other locations. Of course there are other factors that affect location decision such as routing, agreements with ISP and telecom companies and financial considerations that could be just as important. These however are out of scope of this thesis, but could be something to look at in further work.

One of the findings that I consider most important is the development in the total number of requests. Granted the fact of growth of traffic in itself was not surprising seeing that observation of this growth in terms of link utilization was what suggested the need for this work. However I have quantified this growth and looked into into it's sources. Assuming this rate of growth continues and given knowledge about maximum capacity available it is possible to suggest appropriate time for upgrading the cluster with a faster link and more servers. Knowing the source of growth it is possible to make assumptions regarding how creation of new clusters would affect the current one and whether upgrades are needed. For example the data suggests that a new cluster handling requests from Russia would not only relieve the current cluster of 40% of the traffic but also reduce the rate of growth considerably since the traffic from Russia is the fastest growing one. One the other hand a cluster to handle requests from USA is not as beneficial at the first glance as the traffic from USA is does not comprise that large a part of the totals and it's peak is at a low traffic time for the current cluster. However if the same cluster could handle traffic from India and Indonesia which is reasonable, considering that with world routing[40] the path there would still be shorter than from the current cluster. This would take 20% of the load from current cluster. This would also separate the countries that share peak times which would make the effect more significant.

The last part of my results concerned the distribution of requests between transcoders. The first results showed that the internal load balancing algorithm wasn't perfect and the distribution was often uneven which was attributed to a memory effect. After looking further I concluded that there could be a connection between the spread of distributions and the load on the cluster in the sense that lessening the load could help spreading the load more evenly. The current imbalance however calls for some adjustments of the algorithm such as a controll mechanism that would ensure more equal load.

5.2 Limitations

There are several factors that are important to note when considering my results. The log files that I used cover 2 weeks. This period was chosen as the maximum that could be handled considering available disk space and processing time. I believe this is enough to see general trends in the data, but looking at the complete data for the years Opera mini has been available would obviously give more accurate data. I had a choice of selecting a different time span. For example only using 1 day per week would have allowed me to sample 3-4 months of data. This would have allowed me to look at the trends for a longer period of time, particularly I could have checked whether the growth in number of requests continues at the same rate for a longer time. However this would have made the results more uncertain by eliminating redundancy of 7 days per week. Also it would have made impossible to look at the weekly variations. However this turned out to be insignificant as I was unable to find any specific trends there anyway. prevent this work from being able to give a full picture of the situation. Choosing to capture live traffic could also have had a similar effect in the sense that the volume of data would have been reduced adn I would have had more controll over periods of time to include.

Of regard to time constrains I had to focus on 4 variables: time/date, transcoder, country of origin and number of requests However there are a number of other variables that I haven't been able to analyze that could affect the cluster management decisions. For instance country of destination and the volume of traffic measured in data transfered could be checked for correlation with the data

I analyzed and are in themselves quite important. This brings me to the next section which describes further work that could be done on this field.

5.3 Further work

Log analysis is limited only by the number of available logs and the number of variables. In other words further work could concentrate on analyzing the variables not covered in this and the parallel projects and checking whether the results could further illuminate the current situation and assist the management decisions. Some such variables could be destination(url) location, versions of the Opera mini browser and its markup language, language settings in the browser and so on. It is possible that a way could be found to parse the logs more efficiently thus creating opportunities for considering larger time spans. A different approach could be to check for correlations between the data available in the logs and some external variables such as resource utilization on the transcoders. Perhaps that way some other factors affecting the work of the load balancing algorithm could be uncovered. Finally this approach could be applied to analyzing logs from various other applications such as web server farms or dns. Depending on the log format the scripts I wrote could easily be adapted to collect any type of data and adapt it for graphical presentation.

Bibliography

[1] Miniwatts Marketing Group. Internet usage statistics,the internet big picture,world internet users and population stats. Technical report, 2008.

[2] Reuters. Global cellphone penetration reaches 50 pct. 2008. Online: http://investing.reuters.co.uk/news/articleinvesting.aspx?type=Media &storyID=nL29172095&pageNumber=0&imageid=&cap=&sz=13 &WTModLoc=InvArt-C1-ArticlePage2.

[3] Internet Systems Consortium. Internet domain survey, jan 2008. Technical report, 2008.

[4] Opera Software ASA. Opera mini features. Online: http://www.operamini.com/features/, 2008.

[5] Opera Software ASA. Opera mini faq. http://www.operamini.com/help/faq/, 2008.

[6] NetApplications. Browser version market share. Technical report, 2008.

[7] M. Zorz. Interview with christen krogh, opera software's vp of engineering. 2007. Online: http://www.net-security.org/article.php?id=1052.

[8] K. Kopper. *The Linux Enterprise Cluster Build a Highly Available Cluster with Commodity Hardware and Free Software*. No Starch Press, 2005.

[9] B Gueye et al. Towards ip geolocation using delay and topology measurements. In *IEEE/ACM Transactions on Networking*, 2006.

[10] E. Katz-Bassett et al. Towards ip geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006.

[11] L. Subramanian V. N. Padmanabhan. An investigation of geographic mapping techniques for internet hosts. In *Proceedings of the 2001 SIGCOMM conference*, 2001.

[12] L. Daigle. Rfc 3912 whois protocol specification. 2004.

[13] C. Davis. Rfc 1876 a means for expressing location information in the domain name system. 1996.

[14] A. Ma. Netgeo - the internet geographic database. Onine:http://www.caida.org/tools/utilities/netgeo/, 2007.

[15] MaxMind inc. Geoip. Onine:http://www.maxmind.com/app/ip-location, 2008.

[16] Wikipedia the free encyclopedia. Load balancing. Online: http://en.wikipedia.org/wiki/Load_balancing_(computing), 2008.

[17] V. Viswanathan. Load balancing web applications. *O'Reilly ONJava*, 2001. Online: http://www.onjava.com/pub/a/onjava/2001/09/26/load.html.

[18] B. Leong MIT B. Dean, J. Kogel. Lecture:load balancing. Online: mit.ocw.universia.net/18.996/s02/lecture-notes/lect7.pdf, 2002.

[19] B. Awerbuch. Online client-server load balancing without global information. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, 2005.

[20] J. Megarity S. Haigh. Measuring web site usage: Log file analysis. 1998.

[21] K. Bharat J. Pitkow. Webviz: A tool for world-wide web access log analysis. In *Proceedings of the First International WWW Conference*, 1994.

[22] R. Fielding. *wwwstat, HTTPd Logfile Analysis Software*. 2001. Online: http://ftp.ics.uci.edu/pub/websoft/wwwstat/.

[23] D. Eager A. Mahanti, C. Williamson. Traffic analysis of a web proxy caching hierarchy. *IEEE Xplore,Network, Volume 14, Issue 3*, 2000.

[24] A.O. Sykes. An introduction to regression analysis. 1998.

[25] Wikipedia the free encyclopedia. Regression analysis. Online: http://en.wikipedia.org/wiki/Regression_Analysis, 2008.

[26] G.E. Dallal. Introduction to simple linear regression. 2000. Online: www.tufts.edu/ gdallal/slr.htm.

[27] Wikipedia the free encyclopedia. Coefficient of determination. Online: http://en.wikipedia.org/wiki/R-squared, 2008.

[28] E.W. Weisstein. Frequency distribution. Online: http://mathworld.wolfram.com/FrequencyDistribution.html, 2008.

[29] M. Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.

[30] G. Iannaccone et al. Monitoring very high speed links. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[31] L. Deri. Passively monitoring networks at gigabit speeds using commodity hardware and open source software. In *Proceedings Passive & Active Measurement Workshop (PAM)*, 2003.

[32] H.P. Langtangen. *Python Scripting for Computational Science, Second Edition*. Springer, 2005.

[33] M. Owens. *The Definitive Guide to SQLite*. Apress, 2006.

[34] Internet Assigned Numbers Authority. *List of top-level domains*. 2008. Online: http://data.iana.org/TLD/tlds-alpha-by-domain.txt.

[35] Computer Industry Almanac Inc. China tops cellular subscriber top 15 ranking. 2005. Online: http://www.c-i-a.com/pr0905.htm.

[36] eurostat. Number of mobile phone subscriptions 2005. Technical report, 2006. Online: http://epp.eurostat.ec.europa.eu/portal/page?_pageid=1073,46870091&_dad=portal&_schema=PORTAL&p_product_code=ECB10000.

[37] iKS Consulting. iks- rating: Cellular communications in ukraine. Technical report, 2006. Online: www.iks-consulting.ru/eng/engpdf/4.pdf.

[38] MTN group. Country profile south africa. Technical report, 2006. Online: http://www.mtn.com/mtn.group.web/investor/profile/southafrica.asp.

[39] Wikipedia the free encyclopedia. Opera mini, modifications. Online: http://ru.wikipedia.org/wiki/Opera_mini, http://operamini.ru/index.html.

[40] TeleGeography. Global internet map 2006. Technical report, 2006. ISBN 1-886142-64-5.

All online documents are current per 16.05.2008

Appendix A

Scripts

A.1 Data Processing

A.1.1 ipstrip.py

```
#ipstrip.py extracts IP addresses and URLs from an Opera mini log file.
# @author Valeri Cheremetiev


import os,re,time


#The script expects to find a file called filelist with full or relative path
# for the logs.
#IP addresses are printed to STDOUT with counters for total number
# of IPs processed and either case b,
# forwarded IP or case c origin IP and counter for the current case.
files=open( "filelist", ’r’)
ipdbf= open( "iplist", ’w’)
urldbf= open( "urllist", ’w’)
fc=0

#domain names to geolocate
dom=["com","net","org","info","biz","tv","gov","mil","edu","as"]
for lfile in files:
fc+=1
ifile = open( "lfile[:-1], ’r’)

#following regular expression is used to filter correct URLs
r=r"(?:http[s]?://|www.|[WAPwap].)(.*?)(?:/|\s|[A-Z]|:|$)"
cnt=0
cnt1=0
cnt3=0
for line in ifile:
  cnt1+=1
m=line.split(";")
if len(m)>34:
#URL is tagged as Unknown if it can’t be found in field 35.
 url="Unknown"
 s=re.search(r, m[35])
 if s:
    ur= s.group(1).split(".")

#only URLs with following 1st level domain names will be output
   if len(ur)>1 and ur[len(ur)-1]in dom:
cnt+=1
#Up to 3 levels of domain name will be extracted
if len(ur)>2 and len(ur[-2])<4 and str(ur[-3]).lower()!="www":
url=".".join(ur[-3:])+"\n"
elif len(ur)>1:
url=".".join(ur[-2:])+"\n"
urldbf.write(url)

#Use a forwarded IP (field 36) if found, origin IP (field 34) otherwise
 if len(m)>36:
if m[36].strip()!="" and m[36].strip()!="XFF": and m[36].strip()[:1]!=’#’
cnt1+=1
ip=str(m[36])[:-1].split(", ")[0]
ipdbf.write(ip+"\n")
print "b",cnt1,ip
  else:
cnt3+=1
ip=m[34]
ipdbf.write(ip+"\n")
print "c",cnt3,ip,fc
ifile.close()
```

A.1.2 dist_line.py

```
#This script removes duplicate lines from a file,
#writes the results as a new file with same name and a ".d" extesion
#and prints the time and cpu time taken.
#You can specify a "database file" such as a previously processed file
#to exclude more IPs from the results
#@author Valeri Cheremetiev
```

```
import time,sys
try:
ifile = open( sys.argv[1], 'r')
except:
print "usage: sys.argv[0] [input file] <database file>"
ufile = open(sys.argv[1]+".d", 'w')
db={}
results={}
r2={}


t0 = time.time();  c0 = time.clock()
cnt=0
for l in ifile:
l=l[:-1]
cnt+=1
if not(results.has_key(l)):
print cnt,l
results[l]=1
print cnt

if sys.argv[2]:
 ipdb = open( sys.argv[2], 'r')
 for line in ipdb:
l=line.split(": ")
db[l[0]]=1

cnt=0
for l in results.keys():
cnt+=1
if not(db.has_key(l)):
print cnt,l
r2[l]=1

for l in r2.keys():
ufile.write(l+"\n")
ufile.close()
print len(results.keys()), len(r2.keys())

print 'elapsed=%g, CPU=%g' %  ( time.time()-t0, time.clock()-c0)
```

A.1.3 countryTH.pl

```
#This script uses GeoIP database to geolocate all the IPs
#or domain names in the input file.
#Results are written to standrt output,
#output redirection can be used to write them to file.
```

```
#@author Valeri Cheremetiev

die "Usage: $0 [infilename] <number of threads>" if $#ARGV < 1;
use Geo::IP;
use threads;
use Thread::Queue;
my $Q = Thread::Queue->new;
my $Q1 = Thread::Queue->new;




$infile  = $ARGV[0];
$th=10;
$th = $ARGV[1] if $#ARGV > 1;
open(INFILE,  "<$infile") or die "unsuccessful opening of input file; $!\n";

@lines = <INFILE>;

$ff=0;
$kk=@lines;


#set up threads
while (@lines>0){
$Q->enqueue(substr ((shift @lines),0,-1));
}

for (1..$th){
$Q->enqueue(undef);
push @th, threads->new(\&sub2);
}

#start the thread and wait for completion, after which print results to STDOUT
for (0..$th-1) {
$thx=pop @th;
$thx->join();

$Q->enqueue(undef);
while ( defined( $ln = $Q1->dequeue ) ) {
print $ln;
}
}



#subroutine to get the ip/url from Q, geolocate it and put the results in Q1
```

```
# progress is printed to STDERROR
    sub sub2 {
my $gi = Geo::IP->new(GEOIP_STANDARD);
my $k=0;
my $id=threads->self->tid();
my $xid="";
for (1..$id){
$xid="$xid----";}
$xid="$id$xid";
      while ( defined( $ln = $Q->dequeue ) ) {
$k++;
my $c=$gi->country_name_by_name($ln);
if ($c eq "") {
$c="NA";}

print STDERR "$xid $k\n";
my $o= "$ln: $c \n";
$Q1->enqueue($o);
 }
print STDERR "$id: done: $k\n";
}
```

A.1.4 cfiltersql.py

```
'''This script parses logsfiles specified in the filelist
 for each request the ip and url are checked agains datafiles
 for geolocation after which following data is inserted into database:
 Date, transcoder ID,number of the request in file, timestamp,
 microedition locale,image quality setting, obml version, handset language,
 compressed data size, uncompressed data size, processing time,source ip address,
 ip country, destination url, url country'''
#Following files must be present: filelist, c_ip_db - ip to country resolution list,
#c_url_db url to country resolution list
#@author Valeri Cheremetiev

import os,re,time

from pysqlite2 import dbapi2 as sqlite
c0=0
utf=open("malf_urls", "w")
ldb={}

#open resolution files and compile them
ldbf=open("landdb", "r")
for line in ldbf:
l=line.split()
land=" ".join(l[1:])
```

```
lk=l[0].lower()
ldb[lk]=land
ipl={}
ipdb=open("c_ip_db", "r")
for line in ipdb:
l=line.split(": ")
land=l[1]
ipl[l[0]]=land[:-1]
urll={}
urldb=open("c_url_db", "r")
for line in urldb:
l=line.split(": ")
if len(l)>1:
land=l[1]
urll[l[0]]=land[:-1]

files=open( "filelist", 'r')

#create database
conn = sqlite.connect('rq.sqlite')
c = conn.cursor()
#c.execute('drop table rq')
c.execute('''create table rq(nr integer,date numeric, mc text,img integer,
ver integer,lang text,c text,csize integer,usize integer,time integer,
ip text,url text,cip text,curl text)''')
fc=0
for lfile in files:
print lfile
fc+=1
ifile = open( "/home/student/logs2/"+lfile[:-1], 'r')
#r=r"(?:http://(.*?)(?:/|\s|[A-Z]|:|$)|www.(.*?)(?:/|\s|[A-Z]|:|$))"
r=r"(?:http[s]?://|www.|[WAPwap].)(.*?)(?:/|\s|[A-Z]|:|$)"
p=re.compile(r)
cnt=0
cnt1=0
cnt3=0
for line in ifile:


#defaults
curl="NA"
cip="Mcip"
url="Malformed"
ip="mooipd"
  cnt1+=1
ur=""
m=line.split(";")
```

```
if len(m)>34:
 if m[36].strip() != "XFF":
  cnt+=1


#process url
  s=p.findall(m[35])
  print s
  if s:

   if len(s[0])>1:

if s[0]!="":
cnt+=0
ur=s[0].split(".")

url=ur[len(ur)-2]+"."+ur[len(ur)-1]
dom=["com","net","org","info","biz","tv","gov","mil","edu","as"]
if ur[len(ur)-1] in dom:
if urll.has_key(url):
curl=urll[url]

elif ldb.has_key(ur[len(ur)-1]):
curl=ldb[ur[len(ur)-1]]
else:
curl=ur[len(ur)-1]
  elif str(m[35])[:1]!="b":
c0+=1
url=str(c0)+": "+m[35]+"\n"
   utf.write(url)


#process IP
  if m[36].strip()!="" and m[36].strip()[:1]!='#':
cnt1+=1
ip=str(m[36])[:-1].split(", ")[0]
  else:
cnt3+=1
ip=m[34]
  if ipl.has_key(ip):
cip=ipl[ip]
  else:
cip="NA"


#input data into database
  print cnt,m[0],m[5],m[10],m[18],m[19],m[27],m[31],m[32],m[33],ip,url,cip,curl
  c.execute(''''insert into rq values (?,?,?,?,?,?,?,?,?,?,?,?,?,?)
''', (cnt,m[0],m[5],m[10],m[18],m[19],m[27],m[31],m[32],m[33],ip,url,cip,curl))
ifile.close()
conn.commit()
```

A.1.5 cfiltersql2.py

```
'''This script is a compact version of cfiltersql
with minimal data extracted. Requires filelist and a contemporary
 version of geolite database GeoIP.dat to be present ''

import os,re,time,GeoIP
from pysqlite2 import dbapi2 as sqlite
c0=0

gi = GeoIP.open("GeoIP.dat",GeoIP.GEOIP_MEMORY_CACHE)

files=open( "filelist", 'r')
conn = sqlite.connect('rqsmall.sqlite')
c = conn.cursor()
#c.execute('drop table rq')

fc=0
c.execute('''create table rq(trcid text,date text,hour numeric,minute numeric,
second numeric,ip text,cip text,ver integer,fov integer)''')
t0 = time.time();  c0 = time.clock()
for lfile in files:
        for z in range(29):
                print lfile
        lf=lfile.split(".")
        dato=lf[1]
        trcid=lf[2]
        fc+=1
        ifile = open( "/n/"+lfile[:-1], 'r')
cnt=0
cnt1=0
cnt3=0
for line in ifile:
cip="unknown"
ip="unknown"
  cnt1+=1
```

```
fov=2
ur=""
m=line.split(";")
if len(m)>34:
 if m[36].strip() != "XFF":
  cnt+=1

  if m[36].strip()!="" and m[36].strip()[:1]!='#':
cnt1+=1
ip=str(m[36])[:-1].split(", ")[0]
fov=1
  else:

cnt3+=1
fov=0
ip=m[34]
  cip=gi.country_name_by_addr(ip)
  tstamp=m[0]
  tstamp=tstamp[8:]
  t=tstamp[-6:]
  h=t[:2]
  mn=t[2:4]
          s=t[4:]
  print trcid,dato, h,mn,s,ip,cip,m[18],fov
  c.execute('''insert into rq values (?,?,?,?,?,?,?,?,?)
''', (trcid,dato, h,mn,s,ip,cip,m[18],fov))
ifile.close()
conn.commit()

print 'elapsed=%g, CPU=%g' %  ( time.time()-t0, time.clock()-c0)
```

A.2 Data preparation

A.2.1 gprep.py

```
#This script converts a sqlite3 output file grouped by 2 parameters to a 2dimensional
#@author Valeri Cheremetiev

if1=open('rpl','r')

iff=if1.readlines()

#a class for easy handling of 2 level dictionarys
class Ddict(dict):
    def __init__(self, default=None):
        self.default = default

    def __getitem__(self, key):
        if not self.has_key(key):
            self[key] = self.default()
        return dict.__getitem__(self, key)



#read the file
n=Ddict( dict )
for line in iff:
line=line[:-1]
a=line.split("|")
n[a[1]][a[0]]=a[2]

#Various filter settings are possible such as:

# a list of allowed/forbidden values in one of the fields

"""
if a[0] in '''India|Indonesia|Poland|Romania|Russian
Federation|South Africa|Ukraine|United Kingdom|United States'''.split("|"):
n[a[0]][a[1]]=a[2]
"""

#a range for values
"""
if int(a[2])>300000:
n[a[0]][a[1]]=a[2]
"""
# it is also possible to merge several output columns into one field

"""
if int(a[1])!=0 and int(a[1])!=6:
n[a[0]][a[1]]=str(a[2])+str(a[3])
"""

#create key lists
a1=[]
for b1 in n.keys():
    for c1 in n[b1].keys():
        if not c1 in a1:
            a1.append(c1)


#insert zeroes into missing positions to preserve formatting
```

```
for a2 in n.keys():
    for b2 in a1:
        try:
         ax=n[a2][b2]
        except:
         n[a2][b2]=0

a1.sort()

m={}
b4=n.keys()
b4.sort()
for e in a1:
for f in b4:
if not m.has_key(e):
m[e]=[]
m[e].append(n[f][e])


#print results to output using digit format with 7 leading zeroes
g=m.keys()
g.sort()
for a3 in g:
ol=a3
for cx in m[a3]:
ol+="  %07d" % int(cx)
print ol
```

A.2.2 distrib_gp.py

```
#This script converts input datafile into a 3-dimesional numpy array then goes
#thru this array calculating how the input data is distributed between 20 bins
# from 0 to local max and outputs the results to a file.
#Expected files: input cphx.dat, output hspread.gp
#@author Valeri Cheremetiev

import pprint
from numpy import *
if1=open('cphx.dat','r')
of1=open('hspread.gp','w')

iff=if1.readlines()

#a function to strip leading and trailing zeros from a list
def dzero(a):
  for x in range(0,len(a)-1,1):
if a[x]==0:
continue
else:
a=a[x:]
return a

#set up dictionaries to serve as keys fro the array
cz={}
cc=0
tz={}
tc=0
dz={}
dc=0

#create a 3d array with dimesions 50,56,14
az = arange(50*56*14).reshape(50,56,14)

#process the data and fill the array
for line in iff:
  line=line[:-1]
  a=line.split("|")
  if int(a[3])> 5000:
if not cz.has_key(a[0]):
cz[a[0]]=cc
cc+=1
if not tz.has_key(a[1]):
tz[a[1]]=tc
tc+=1
if not dz.has_key(a[2]):
dz[a[2]]=dc
dc+=1
az[cz[a[0]],tz[a[1]],dz[a[2]]]=int(a[3])


#sort the keys for ordered output
czk=cz.keys()
czk.sort()
dzk=dz.keys()
dzk.sort()
cstr= "  ,-,".join(czk)
of1.write(cstr+"\n")
print cstr

#calculate and output distributions
for date in dzk:
xm=date+":,"
```

```
for country in czk:
p=az[cz[country],:,dz[date]]
p=dzero(p)

s,f=histogram(p,20, range=(0,p.max()), normed=False)
#print s,f
mm=s.max()
ar=s.tolist()
ar=dzero(ar)
if ar:
ar.reverse()
ar=dzero(ar)
else:
ar=[]
xm+="%02d,%02d," % (mm,len(ar))
print xm
of1.write(xm+"\n")
```

A.3 Plotting

A.3.1 distrib.py

```
#This script calculates how the input data is distributed between given bins
# and then plots the results using gnuplot then calculates the height
#and width of the distributions and outputs thos to STDOUT
#@author Valeri Cheremetiev

import pprint,Gnuplot

from numpy import *

if1=open('rptr.f.p','r')

#set up the gnuplot options
g = Gnuplot.Gnuplot(debug=1)
#g('set term post enh col')
#g("set output '/home/student/out3/mpt1.eps'")
#g('set term x11 size 1200,900')
g('set multiplot')
g('set data style boxes')
g('set size 0.25,0.25')
g('set xtics rotate')
g('set style fill solid 0.5')
```

```
# set the range for the bins the data and the ticlabels
rg=range(330000,570000,20000)
rg1=range(0,26,1)
days=range(4,18,1)
mf=['"'+str(x/1000)+'" '+str(rg1.pop(0)) for x in rg]
mf=mf[1:]
ticks="("+','.join(mf)+")"
g('set xtics %s'% ticks)

#process the data
ij=[]
trc=[]
for line in if1:
lx=line.split()
l=[int(x) for x in lx[2:]]
ij.append(l)
trc.append(l[1])


bbb=array(ij)
bbt=transpose(bbb)


sf=[]
b=array(rg)


#calculate histograms
for i in range (0,14,1):
i+=1
h=bbt[i]
print h
s,f=histogram(h, b, range=None, normed=False)
sf.append(s)

#set the range for multiplot
lp=[]
for i1 in range(0,4,1):
for j1 in range(0,4,1):
l,p=(float(i1)/4,float(j1)/4)
lp.append([l,p])


#plot the data
for meh in sf:
```

```
ff=lp.pop(0)
orf="set origin %g, %g" % (ff[0],ff[1])
g(orf)
tx='"%d/1"' % days.pop(0)
ticks='(%s 0,' % tx +','.join(mf)+')'
g('set xtics %s'% ticks)
g.plot(meh)

g('unset multiplot')



# a function to remove leading zeros in array
def dzero(a):
  for x in range(0,len(a)-1,1):
if a[x]<=1:
continue
else:
a=a[x:]
return a

#calculate height and width of the distributions
ls=[]
xm=[]
for m in sf:
mm=m.max()
xm.append(mm)
ar=m.tolist()
ar=dzero(ar)
ar.reverse()
ar=dzero(ar)
print ar
ls.append(len(ar))
print ls
```