ISBN 978-83-60810-51-4

Content Delivery Network Monitoring

Krzysztof Kaczmarski
Marcin Pilarski
Faculty of Mathematics and Information Science, Warsaw University of Technology
ul. Koszykowa 75, 00-662 Warszawa, Poland
Email: k.kaczmarski@mini.pw.edu.pl
Email: marcin.pilarski@mini.pw.edu.pl

Abstract—**This document describes the architecture of a distributed Content Delivery Network (CDN) monitoring system and its deployment in a research environment in one of the biggest telecommunication company in Poland and involves about fifty nodes distributed in the country and database system located on dedicated storage cluster working in RD center in Warsaw.**

I. Introduction

Nowadays big media companies are highly interested in publishing in the internet which has became one of the most influencing medium in the world. Tens millions of users may be easily gathered by an event or well designed marketing strategy. However, no media publisher can stand network traffic of this volume. This is how telecommunication companies and Content Delivery Networks are involved by playing crucial role in assuring high level of services especially high quality streaming data. Its main idea, content migration and distribution from the central publishing server (often called an origin server) to as many nodes as possible, shortens the path to clients, reduces network traffic and decreases bottlenecks in the infrastructure. It is also much more economical to place CDN nodes in already existing telco backbone than to build a new backbone from scratch only for one content provider. Telecommunication companies begun selling CDN as a service to many content providers which further reduces running costs. In this win-win solution each party is satisfied: media can achieve better quality of service for end users reducing internal financial engagement and the users may get high quality of streaming media on demand. However, this scenario requires tight cooperation between a content provider and a network owner. It often includes charges for certain data volume or achieved speed of data transfer and therefore requires detailed analysis of data flow, activities of users and service quality. The task of high level data delivery monitoring remains an open problem.

This paper describes an architecture of a CDN system monitoring service and experiences of its deployment in one of the biggest telecommunication companies in Poland. The rest of the paper is organized as follows. The remaining part of this section describes the functional and non-functional requirements for the CDN monitoring system. Section II provides system architecture and presents experiences and results. Section III presents results and section IV concludes.

A. Problem Description

CDN Monitoring requires detailed informations, statistical as well as sensor measurements to be available real-time and cover any given period of time not loosing any detail. For example, if a malfunction of one is detected one may need to track its behaviour up to the unlimited point in the past in order to find possible coincidences with other events. Therefore all relevant data concerning CDN operation must be stored in exact shape. The general functional requirements include:

1) all data and statistics must be available real-time
2) the system must be able to answer any query for any given period of time in the past
3) all measured data must be safe and cannot be lost in case of hardware failures
4) the system must scale well in case of substantial CDN growth (more nodes to be covered)
5) the system must scale well in case of substantial number users increase (more events per second)
6) the system must be available all the time even in case of hardware failures
7) the system must cover the following types of reports:
   - content delivery (speed of content downloading, bytes sent to clients, number of requests, etc.)
   - content popularity (number of requests per content)
   - users behaviour (number of unique users, number of different files requested by one user, etc.)
   - nodes behaviour (alive nodes counters, reachable nodes, etc.)
8) all information if possible should be supplied with geographical information to allow geo-spatial analysis
9) the system must present aggregated global reports as well as reports concerning peering groups or single nodes
10) the system must be able to separate reports for different content providers, origin servers or geographical locations

Data aggregation cannot reduce information resolution in order to be able to query all available data.

Additionally there are significant non-functional requirements concerning mostly number of data to be processed in given time. The most basic analysis can be based on CDN logs data fetching. We performed platform load experiments covering 10Gbps per single node, which generated around

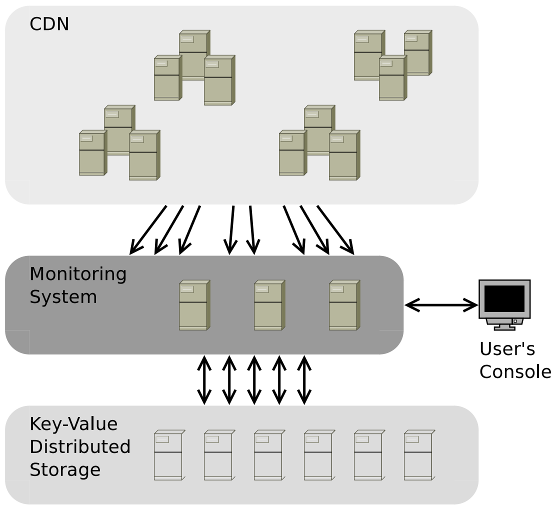


Fig. 1. General architecture of the CDN monitoring system

400.000 log entries every five minutes. This means that with equal 30Gbps (maximum predicted) load of a system build of 50 nodes we may need to process only 80 entries per second on single node.

II. CDN Monitoring Database

The presented requirements lead us to the conclusions that first the system must be distributed in order to achieve high scalability and safety of data and second must be based on one of a key-value storages which can quickly perform map-reduce tasks in order to answer queries concerning any data and time period. Data collecting should be also distributed to minimize network traffic and perform as much of data analysis in parallel as possible. The general architecture of the system in presented in figure 1.

There are three layers of our solution. CDN data collectors which are placed on the CDN nodes. They perform the first step of data analysis, aggregation and annotation. Then this preprocessed data is sent to the database system which is also responsible for user queries processing. The last layer is constituted by data storage which is actually hidden from users. However its map-reduce capabilities are required for real time query answering.

A. CDN Logs

This section briefly describes Coblitz CDN content delivery logging mechanism. It follows a general idea of events logging and may be perceived as an example of events storing. We believe that any other CDN system may use similar logs or any other reporting solution which can be used as an entry point of our data analysis.

Each log entry is composed of the fields presented in table I. We can treat it as a raw database table which will be an input for data analysis performed by collectors in CDN nodes. Any other data published on nodes may be treated similarly. Our approach does not loose generality focusing only on Coblitz Logs. What is more, we are also prepared for any Coblitz log mechanism changes. We only need to update log parser or log collector. The rest of the system may remain the same.

B. Metrics Collectors

The crucial part of the system is devoted to collecting data from CDN nodes. In order to describe appropriate solution we must first understand what kind of data we need to gather.

The simplest type of data is based on counters like number of MB sent from a node, speed, or number of requests. This can be collected independently on each node. In the global image this kind of metrics are additive, sum of MB sent from each node gives us total number of MB sent from all CDN. This also holds if we slice data for given content provider, file types or origin servers.

However there are also metrics which are not additive when moving from local to global scale. For example number of unique client IPs. This value collected for a single node cannot be added to a value of another node. Sets of IPs and other not additive metrics must be processed globally in order to get correct results.

This leads to hierarchical set up of metrics collectors. The first layer works on the nodes, very close to local log files. They are responsible for initial data analysis and additive metrics. This layer sends data in two directions. Additive metrics are directly sent to a store via a database interface. Preprocessed data for non-additive metrics are sent to the second layer of collectors to perform reduction. This phase of data processing may be multi-step to get good efficiency as it is done in other distributed reduction solutions. The architecture of collectors in presented in figure 2.

C. Local Data Aggregation

Log files containing entries for each event may grow up to several hundreds thousands of records. The first level of aggregation done locally in a node must reduce this very detailed information by aggregating events which apply to the same data category. For example, if we calculate number of

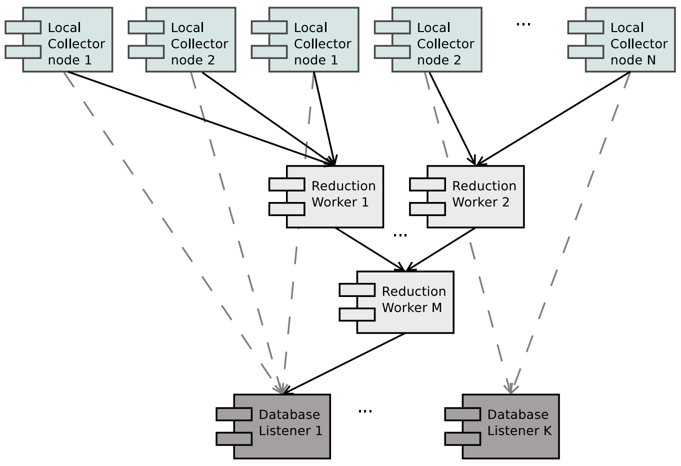


Fig. 2. Data flow among the collectors. Solid lines denote global values reduction procedure. Dashed lines indicate direct database storing actions.

TABLE I
SAMPLE LOG ENTRY FIELDS.

No	Field name	Field type	Field meaning
1	log_version	Number	Log entry version (currently only 1)
2	account_name	Text	Name of the content provider
3	request_id	number	Unique request number
4	start_time	Epoch [ms]	Start of downloading time (unix format)
5	elapsed_time	seconds.msecons	Downloading time
6	client_IP	Decimal.decimal.decimal.decimal	IP
7	HTTP_method	Text	GET or PUT
8	URL	Text	Resource url
9	HTTP_version	Text	Version oh http protocol
10	response_code	Number	http response code
11	total_object_size	Number [bytes]	size of a file
12	bytes_sent_to_client	Number [bytes]	size of transmitted data
13	cache_hit_bytes	Number [bytes]	size of data taken from cache
14	range_from	Number [bytes]	0 in case of beginning of a file
15	range_to	Number [bytes]	-1 in case of end of a file
16	content_type	enumeration	Mime type of the content
17	redirection_location	enumeration {0, 1, 2}	{none, to origin server, to a peer node}
18	status	enumeration {complete, progress}	status of a data chunk

requests for certain URL all records referring to that URL may participate but will produce only one value as an output.

Local data aggregation script processes log file as a raw table database and produces a time limited MOLAP cube. The cube may contain arbitrary number of dimensions and metrics depending on client needs. Any value derivable from log file and interesting as a statistical result may be a metric. Any value which is a category for a user who wants to distinguish values may be a dimension. Figure 3 presents a sample cube in form of a tree. Each dimension forms a separate level. Any possible value of a dimension generates children in the next level. Leafs contain metrics. In the example the cube has three dimensions (state, provider and origin server) and two metrics (requests and Mbps).

Our data cube is inherently connected to time passing. Since log files change dynamically, our cube is valid only for certain period of time. One cube intersection changes every given period of time. In a longer time period one intersection forms a time series. Different dimensional values define different intersections in a cube which means also different time series.

In our solution cube creation is done by a simple Python script using nested `dict` built-in type. Is is a classical key-value structure. At each tree level children are defined by dictionary entries with keys for each available dimension value. A dictionary value contains a nested dictionary for a sub-tree. The cube is created by a recursive procedure which takes a single log entry, extracts all dimensional values and then searches for appropriate child in the tree. Although Python is not the fastest possible solution one local collector script calculating around dozen metrics may process about 1000 log entries per second. Being executed every 300 seconds (time of local log rotation) it could handle correctly logs containing up to 300.000 entries. If a better efficiency is necessary Python interpreter may be easily exchanged with something faster.

D. Example Metrics

This section lists sample metrics which may be calculated by local collectors only upon simple CDN log files and other local system data. All values, as it was explained earlier, change every five minutes and are stored in database as time series. Therefore to calculate number of request per hour we need to sum 12 subsequent values in a selected series.

Each metric is calculated for one CDN node. However, most of them may be easily aggregated among nodes to get global metric value. For example adding number of request from all nodes will give correct number of requests for all CDN. For each metric we describe which aggregation function (sum, min, max, avg) produces sensible results in a global scope.

Content Metrics:

cdn.vod.requests
Number of Video-on-Demand requests.
Globally – min, max, avg, sum.

cdn.Mbits_per_second
Number of megabits per second sent to end users.
Globally – min, max, avg, sum.

cdn.Mbytes
Number of megabytes sent to end users.
Globally – min, max, avg, sum.

cdn.BHit
Percentage of cache hits.
Globally – min, max, avg.

CDN Life Metrics:

cdn.node.hits
Number of processed log entries.
Globally – min, max, avg, sum.

cdn.node.unique_ips
: Number of connections from unique IP numbers. This metric calculated for given node can be only understood locally. It is not additive among different nodes since sets of unique IPs for many nodes may contain non empty intersection. At a local level it can be only a measure of a local system. Utilization at a global level requires global IP sets to be calculated globally.
: Globally – max – also gives information which node has the biggest number of unique clients.

cdn.node.max_same_ip_connections
: Maximum number of requests from the same IP. This metric again calculated for single node is correct only for this node. For global metric this values should be calculated upon data from all available nodes.
: Globally – max – also gives information which node has the biggest number of connections from the same IP. This may help to detect potential attackers.

The last two metrics are not globally additive and require reduction procedure (done by reduction workers) described in the next section.

Collectors Self State Metrics:

cdn.collector.processing_time
: Time in seconds of single log processing on a node.
: Globally – max, min.

cdn.collector.heartbeat
: This is a very interesting metric containing information about local data collector state. It is a simple counter initialized when a collector is started. Each log processing activity increases a counter by one. If a collector runs correctly this value should increase by one every 5 minutes. Globally sum of this metric for all nodes should increase by number of running collectors every 5 minutes. If slope of this metric changes it means a collector malfunction.
: Globally – sum – increases by number of running collectors – min, max – shows which collector is running the longest/shortest time – avg – shows an average time of collector run. The time is understood as a 5 minute periods counter.

cdn.collector.errors
: Number of errors encountered in local node processing. This is a simple information metric which may point administrator to a certain collector log file in order to analyse local problems.
: Globally – min, max, sum, avg.

Hardware State Metrics: The following metrics are not collected from CDN log files but taken directly from Linux `proc` system. We find them informative about the current state of the hardware. Ability of joining these statistics with other CDN data also lets us to infer many interesting informations about correlations of CDN content delivery and hardware utilization.

proc.net.bytes
: Bytes in/out (rate)
: Globally – min, max, sum, avg.

proc.net.packets
: Packets in/out (rate)
: Globally – min, max, sum, avg.

proc.net.errs
: Packet errors in/out (rate)
: Globally – min, max, sum, avg.

proc.net.dropped
: Dropped packets in/out (rate)
: Globally – min, max, sum, avg.

df.1kblocks.used
: 1K blocks used
: Globally – min, max, avg.

df.1kblocks.free
: 1K blocks available
: Globally – min, max, avg.

proc.loadavg.15min
: last 15 minutes processor load average
: Globally – min, max, avg.

proc.meminfo.memfree
: Current system free memory
: Globally – min, max, avg.

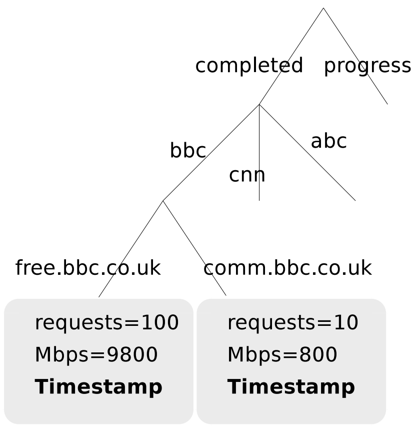


Fig. 3. A sample cube for two metrics (requests, Mbps) and three dimensions (status, provider, origin_server). Only one branch visualized for clarity.

E. Reduction Workers

Reduction workers are designed to perform efficient global metrics calculation. They are organized in a tree with data flow from bottom to the top (see fig. 1). Each worker is specialized for one simple reduction task. Each task is defined by number of inputs and algorithm to be done. For example unique IP connections require input IP sets union and sending it to the next reduction level, which again performs union on inputs and sends results to the next level. The last reduction worker gets global data and may finally calculate global metric value. The result is sent to a database system.

F. Time Series Database Component

Our prototype utilizes OpenTSDB [1], a popular time series database system. It is widely used in data centres to monitor nodes state. It is able to store values in time with bandwidth of more than 10.000 points per second. Each value may be tagged with arbitrary number of tags, string labels. Tags may be used in queries to combine or split different time series. For example if we consider processor load it may be tagged with name of a node. The system is able to perform aggregation of given nodes processors load and draw sum, min, max or average or show load for single nodes. Proper time series tagging lets to group time series and get proper queries results.

For example figure 4 presents three plots of number of CDN log entries. Each value stored in a database is tagged with labels containing provider, node and geographical location. Time series can be therefore grouped for providers, locations and nodes or drilled down to see metrics for single nodes or single resources.

OpenTSDB runs several distributed demons (Time Series Demons – TSD). They are independent and equal without any state sharing. All of them communicate with common data store which is build on HBase [2]. Since HBase is distributed itself and provides data replication there is no single point of failure in the system. HBase follow BigTable distributed data model (sorted distributed hash-map) and is able to scale very well and answer queries very quickly.

A big advantage of OpenTSDB is that users do not need to communicate with the data store directly. A user may just send a query to an arbitrarily selected TSD which produces an answer in form of a data plot or a list of data points which may be further analysed in other statistical systems. The system is very simple, stable and reliable, yet very powerful.

III. Results

A. *Other Clusters Monitoring Systems*

While distributed cloud services became more and more popular many services providers discovered problems with hardware layer state monitoring. There appeared many different solutions which are able to track events and parameters of distributed nodes and present them in a feasible administration console. These systems must process millions or billions of records and must be extremely efficient. In many cases advanced data mining algorithms are necessary to get appropriate information from this kind of huge data sets. Often companies collect all possible data for future analysis even not being sure what kind of data extraction is possible and what kind of analysis is necessary.

Ntop is a network traffic measurement system that shows the network interface usage [3]. It can show many different views of network utilization and display statistics limited to the last 24 hours. It is quite low level tool and cannot collect CDN content flow information. Our solution may both perform ntop like tasks and also provide high level data analysis.

Our solution is a bit similar to RMON monitoring standard [4] in the aspect of monitoring agents. In RMON agents are located in many places and track internet packets going through that place. In our solution agents are located in all CDN nodes.

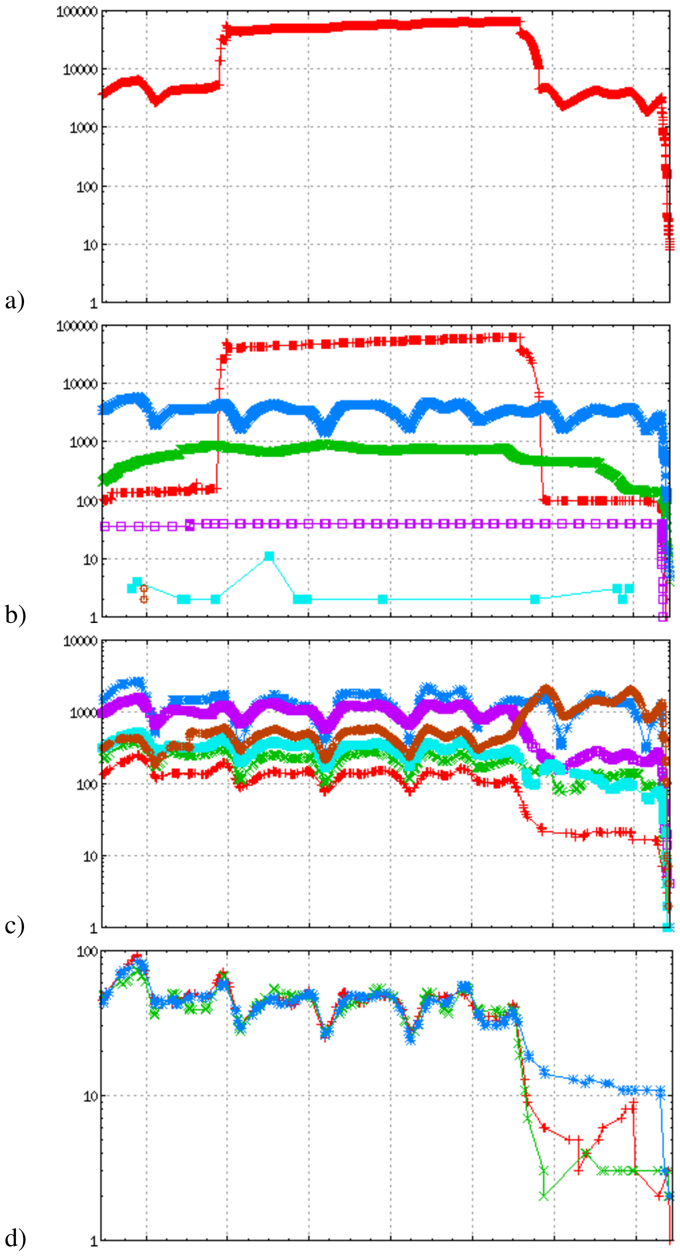
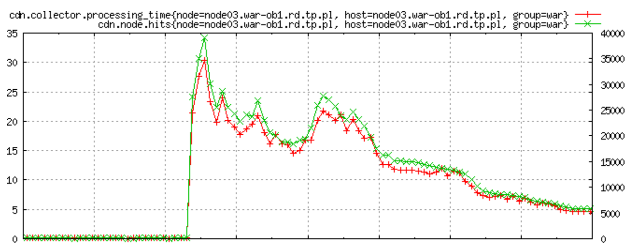


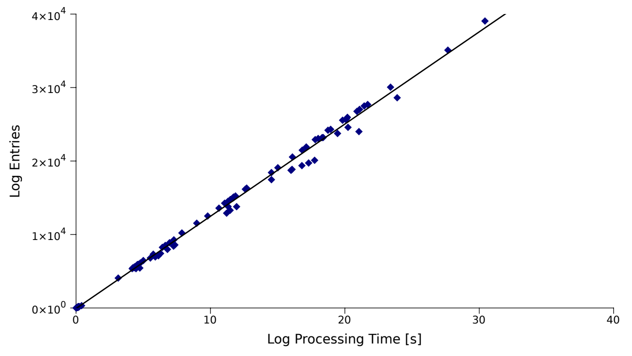
Fig. 4. Analysis of number of CDN log entries for one week. a) summary for all nodes. b) summary for different content providers. c) summary for one provider and different geographical locations. d) separated entries for nodes for one location for one provider.

Different approach is presented by Chukwa which is devoted to arbitrary logs collection and analysis [5]. Data is stored in not aggregated form and may require huge space. Reports are calculated by custom functions which must process all data. In the contrary our system stores only numerical data in appropriate resolution.

Another system which is designed to scale to large number of nodes is Scribe [6]. Again it only keeps messages in a form for key-value informations as categories and strings sent to subscribers of particular types of messages. It is rather a notification systems than a database which can be queried



a)



b)

Fig. 5. Analysis of log processing time in single CDN node. a) plot of processing time (red line scale to the left) and number of log entries (green line scale to the right) during the heavy load experiment. There is a strong linear correlation between the two variables presented in the bottom figure (b).

afterwards.

In the contrary our solution not focuses on data volume reduction and stores only tagged numerical values which are subject to arbitrary queries. It may also legitimate with high scalability and great robustness. We managed to store thousands of data points per second and retrieve data in queries not lasting more than a few seconds even when drilling though data collected from one month of cluster operation.

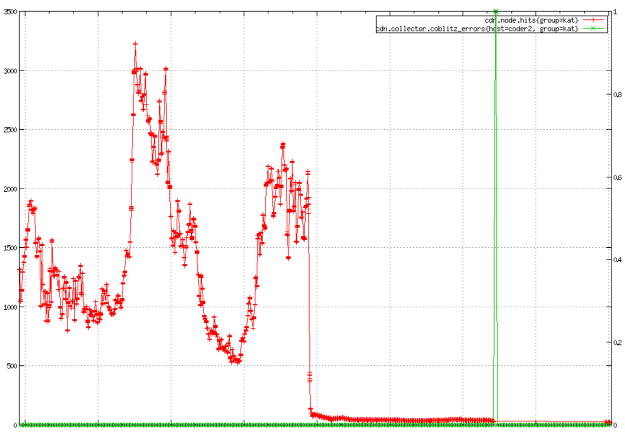
B. The CDN Monitoring System Capabilities

This section describes our CDN monitoring system capabilities.

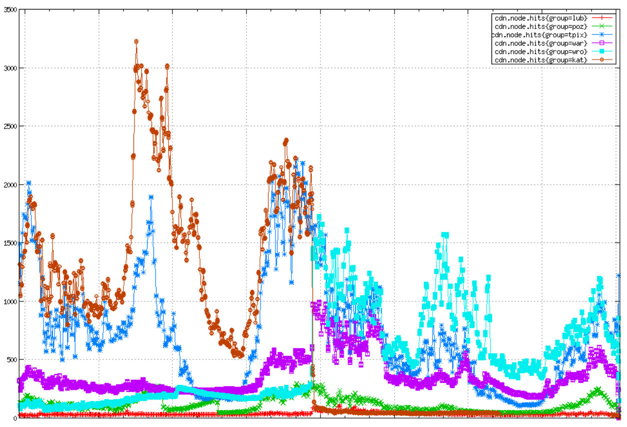
The system built on OpenTSDB and HBase has no single point of failure. It scales well for hundreds of nodes and may process queries very quickly using the map-reduce procedure on Hadoop store. We achieved all functional and non-functional features described in user expectations in section I-A.

Knowing the efficiency of the OpenTSDB and HBase we need to focus on our local collectors performance which may be the only potential bottleneck. We encountered that the overall efficiency of a single node data collector meets the requirements. Figure 5 shows that we are able to process approximately 1000 log lines per second. This means that with given 5 minutes time window to process all the log we can properly handle 300.000 log lines. This is more than sufficient result. What is more, as it was stated earlier, we can speed it up by exchanging Python language scripts with a faster solution.

The system perfectly diagnoses content streaming. For example we may track changes in content distribution load among geographical locations. Figure 6 shows this kind of



a)



b)

Fig. 6. Analysis of CDN log entries for one week. a) summary for all nodes. b) summary for different content providers.

analysis. Upper plot presents overall users of CDN cluster located in Katowice. We can observe that at certain point in time traffic dramatically decreases and then completely disappears. Combining this plot with a graphical representation of node pinging errors we can observe that there is coincidence of this two events. A node encountered network communication problems which resulted in traffic absence. Diagnosing of radical change in cluster load requires further investigation. Bottom plot of fig. 6 presents traffic of all CDN peering groups. We can see that in the same time when Katowice load decreased Warsaw and Wroclaw utilization increased. This means that the traffic was automatically moved to other clusters. In fact this was cased by an accidental error introduced by the system administrator. This proves usability of the solution in the field of content distribution monitoring.

IV. Conclusions

In this paper we presented architecture of a distributed CDN monitoring system. It is well scalable, robust and assures data consistency for any period of time regardless to the time window one would like to analyse. It can answer complicated queries and present any information required both for content providers and system administrators.

REFERENCES

[1] B. Sigoure, “OpenTSDB scalable time series database (TSDB).” http://opentsdb.net, 2012. Stumble Upon.

[2] The Apache Software Foundation, “Apache HBase.” http://hbase.apache.org, 2012.

[3] L. Deri, R. Carbone, and S. Suin, “Monitoring networks using ntop.,” in *Integrated Network Management* (G. Pavlou, N. Anerousis, and A. Liotta, eds.), pp. 199–212, IEEE, 2001.

[4] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu, “Introduction to the remote monitoring (RMON) family of MIB modules,” tech. rep., The Internet Society, Network Working Group, 2003.

[5] J. Boulon, A. Konwinski, R. Qi, A. Rabkin, E. Yang, and M. Yang, “Chukwa, a large-scale monitoring system,” in *Proceedings of CCA*, vol. 8, 2008.

[6] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, “Scribe: The design of a large-scale event notification infrastructure.,” in *Networked Group Communication* (J. Crowcroft and M. Hofmann, eds.), vol. 2233 of *Lecture Notes in Computer Science*, pp. 30–43, Springer, 2001.