Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 801–805

Content Delivery Network Monitoring with Limited Resources

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

Bogdan Banasiak, Christophe Kabut
Orange Labs Poland
Telekomunikacja Polska S.A.
ul. Obrzeżna 7, 02-691 Warszawa, Poland
Email: bogdan.banasiak@orange.com
Email: christophe.kabut@orange.com

***Abstract*—This article presents results of designing a Content Delivery Network monitoring system for resource limited applications. CDN monitoring is important both for content providers (media companies) and administrators (Internet Service Providers). It is a challenging task since network traffic may generate huge volume of data which must be parsed and analysed in real-time. This paper describes the design of a prototype system that uses a small resource footprint, scalable Big Data solution, which is motivated by real world use cases.**

I. Introduction

DISTRIBUTED internet systems monitoring is already an important branch of industrial computer systems. Network data transfer, hardware state and many other aspects of distributed systems need to be constantly tracked in order to detect anomalies, prevent failures and measure hardware and software load. In this field Content Delivery Network (CDN) monitoring becomes a fast developing branch of telecommunication. Network caching is used in a variety of different contexts to provide cost savings due to decreased bandwidth consumption, but also to reduce network latency. Due to the techniques described in [1] that makes RAM/HDD resources ratio important especially for use cases in developing regions of the world.

CDN Monitoring requires detailed information, both statistical and sensor-reported to be available real-time and cover any given period of time while not loosing any detail. For example, if a malfunction is detected one may need to track it back behaviour up to the unlimited time point in order to find possible coincidences with other events. Therefore all relevant data concerning CDN operation must be stored in exact shape.

There are several highly efficient systems able to collect and store raw logs. Flume [2] for example is able to collect log lines, send them to a HDFS storage and generate reports on them. Hive [3] can evaluate SQL-like queries over large data volumes collected in Hadoop HDFS [4]. All these classical Big Data technologies require large hardware configurations while storing data inefficiently from the time series point of view. Much better approach is applied in OpenTSDB [5], a dedicated time series database running on Hbase [6]. Its data model can be effectively tuned to achieve both very fast querying and limited hardware. This paper describes optimizations for a CDN monitoring system based on experiences of its deployment in one of the biggest telecommunication companies in Poland which operates commercial CDN services.

A. Limitations of Network Traffic Monitoring

The CDN monitoring becomes especially challenging in developing regions where storage footprint for monitoring is very limited. The situation at those regions is especially unfortunate since the transmission of data logs into public clouds or third party services cannot be performed for the reasons like cost, security and privacy.

This type of monitoring system may be useful in environments where the link bandwidth is a constrained resource e.g. the CDN nodes deployed at libraries, schools, remote communities, etc., basically all locatlizations where uplink bandwidth is limitted.

Also, the security and privacy constraints are general in this scope e.g., most of ISPs are reluctant to deploy subset of CDN systemts called Transparent Caching solutions primarily because of logging concerns in third party services.

The goal of this research was to develop CDN monitoring tool for usage in developing regions with limited resources. We present the system which provides the same level of flexibility of traditional heavy weight monitoring tools.

II. Time Series Database

From a theoretical point of view the system needs to store time series understood as a collection of observations made sequentially in time [7]. These discrete observations T are represented by pairs of a *timestamp* and a *numerical value* (t_i, v_i) with the following assumptions:

- number of data points (timestamps and their values) in one time series is not limited,
- each time series is identified by a name which is often called a *metric name*,
- each time series can be additionally marked with a set of *tags* describing measurement details,
- observations may not be done in constant time intervals,
- storage should not limit time series to be piecewise constant or linear (see Fig.1).

978-1-4673-4471-5/$25.00

A. Architecture

The general architecture of the system is composed of three components: data collection, data storage and querying engine.

As in many other systems data are inserted by distributed *data collectors*, which may work directly on data sources and push data into the data storage. Usually data collection is a result of the ongoing measurement or monitoring processes (the operating system, databases and web servers or network devices: see OpenTSDB monitoring tools [8]). The system does not limit possible data which can be inserted and analysed. The only requirement is that it must have a form of *time series*.

According to the existing taxonomies (like in FAME system [9]) measured values are of two types [10]:

- level values stay the same from one period to the next in the absence of activity. For example, inventory is a level value, because inventory stays the same if you neither buy nor sell.
- flow values are zero in the absence of activity. For example, energy expenses go to zero if there is no consumption.

This distinction turns out to be important when interpolating missing values and for time scale conversion. Our system is open for both solutions by inserting zeros when necessary during data collection time.

As we described in [11] our CDN monitoring system is built on OpenTSDB which uses HBase and HDFS as a data storage. OpenTSDB is responsible for storing data in two HBase tables: the first one contains main data compacted into one hour blocks, the second one keeps time series and tag IDs. OpenTSDB is also evaluating queries by getting proper data blocks from HBase and aggregating them into a time series according to the query semantics. Performance of all the system is influenced by all the three components: HDFS, HBase and OpenTSDB.

B. CDN Metrics and Metrics Querying

For the purposes of CDN monitoring the most important metrics from an operational point of view are:

- bandwidth – average number of bits per second transferred from the platform within a given time period [kbps, Mbps, Gbps]
- traffic – sum of bytes transferred in a given time period [kB, MB, GB]
- sessions – number of active sessions in a given time period

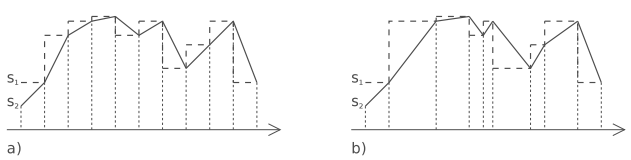

Fig. 1. Time series with constant (a) and variable (b) sampling, piecewise constant (S_1) and piecewise linear (S_2).

- unique clients – number of unique IP addresses
- url hits – number of content download events content from given URL addresses
- byte cache hit ratio – percent of data volume sent from the cache [%]

Additionally, all metrics must be further divided into the following dimensions:

- node name – CDN node name or IP which sends data to a client
- cluster group name – group of nodes logically grouped together
- http response code – HTTP code sent in response to a client request
- country ISO code – client's geographical location
- AS code – Autonomous System code of a client's Internet Service Provider
- CDN instance name – a logical grouping of nodes working in one or multiple CDNs
- provider name – name of a content provider
- origin server name – name of an origin server which contains the original data cached by a CDN
- url – url which is tracked by the system

We use the following metric naming schema:
`[infrastructure].[measurement].[aggregation]`
where:
`infrastructure` indicates a system which is measured (in our case CDN instance, but could also be CPU),
`measurement` indicates a name and type of the measurement being done (for example Mbps, MB, requests, etc.),
`aggregation` describes the type of aggregation done during the data collection which results in one data point for a given time interval.

For example, one of the clients' requests in Poland downloading the content from a hypothetical disco-tv could be stored as (a metric name, timestamp and value followed by a list of tags):

```
cdn.mbps.avg5min 2013-02-20-21:04:20 2.50
node=node01.waw.cdn-lab.pl group=waw02 httpcode=200
country=pl as=AS5617 cdn=lab provider=disco-tv
origin=share.disco.pl url=clip463421-doda
```

which means that within 5 minutes period the collector calculated average value of Mbps to be 2.5 for server named **`node01.waw.cdn-lab.pl`** being in cluster group **`waw02`** for a client located in Poland in **`AS5617`** downloading content from a provider named **`disco-tv`** originating from a server **`share.disco.pl`** with the use of CDN instance named **`lab`** and referencing url **`clip463421-doda`**.

This approach, characterized by defining all possible dimensions for each metric, enables very flexible querying of collected data. For example, one could ask for an average Mbps for given provider in Germany in a given time period or number of error codes returned for a given provider's network (identified by AS number) when accessing certain CDN node by given URL. Please note, that all tag values combination defines one time series. Total number of time series stored for

tag name	min values	real-life
node	10	100
group	5	10
httpcode	5	12
country	1	90
as	1	30
cdn	1	5
provider	1	5
origin	1	20
url	10	50
total time series	2500	$810 \cdot 10^9$

TABLE I
APPROXIMATION OF POSSIBLE NUMBER OF TIME SERIES FOR ONE METRIC IN MINIMAL AND REAL-LIFE SCENARIOS.

one metric is then given by $v_1 \cdot v_2 \cdot \ldots \cdot v_i,\ i = 1 \ldots l$ where l stands for the number of tags attached for a metric and v_i is maximal number of distinct values possible for tag i. Let us analyse how many time series can be generated by an average metric and tags set. An approximation of minimal and real-life scenarios are given in Table I.

The real-life approximation was done using production data from one of the working average CDN systems. It assumes it is possible that accessing all URLs will be tracked with possibly all http response codes and hitting all possible nodes from each country. Since there might be some dependencies between the dimensions the real observed values may be smaller, however, the database must be prepared for the worst scenario which may appear for example during malfunction. Actually, anomalies are the most important from the analytical point of view. Therefore the system should be able to present data including all possible dimensions, and tag values ranges.

C. Data Volumes

Another aspect of CDN monitoring is the estimation of data volumes which must be processed by the system in real-time. CDN traffic monitoring may be based on at least three information sources: CDN proprietary logs, Apache http logs and Syslog events. Due to the latest IETF standarization efforts [12] and [13] we may expect fourth information source of the logs from CDN interconected systems. One request for content generates one line in the log entry (about 0.2 kB). One Smooth Streaming video generates up to 300 entries per 5 minutes (about 60 kB) 10k users watching 90 minutes smooth streaming video generates about 10GB of log data. In some cases log coalescing funciton may be used, however that techniques may reduce size of log by a factor of 10 in average. Another example is 100k users downloading a 1GB game with 5Mbps connection may generate up to 1GB of log data. These data need to be downloaded and analysed and cannot wait for batch processing at night or weekend days. CDN systems offering content worldwide may be equally loaded all the time in which case a monitoring system must collect and process data in real-time.

III. OPTIMIZATIONS

It is a typical HBase performance design pattern to build clusters of at least 11 nodes. In more demanding environments 50 nodes is an average number. This approach would lead to building a monitoring infrastructure more expensive than the monitored CDN itself. Therefore the time series database must be deployed on a single node cluster with a pseudo-distributed configuration on one hand allowing for possible quick extensions in the future and generating sensible running costs on the other hand.

A. Single Node Configuration Performance

Let us now analyse the performance of a single node configuration. We executed two types of queries:

- Aggregating all available time series for a given metric into a one time series in a given time window. For example[1]: `select sum:cdn.bandwidth.15min from 01.01.2012 to 31.12.2013`
- Performing a selection of time series using tags before the aggregation and within a given time window. For example: `select sum:cdn.bandwidth.15min from 01.01.2012 to 31.12.2013 where provider=disco-tv and country=de`

Due to the CDN logs behaviour we used an equal sampling with 5 minutes time period therefore each day is described by $24 \times 12 = 288$ time points. Please note that the time series may not be continuous and therefore their number may vary throughout a queried time window. In Table. II we can see that an average time series processing speed for a query which is scanning all time series in given a time window is about 660k points per second. Since we scan all data, the number of points and time series is constantly increasing. The processing of 4 days data takes almost 40 seconds. Queries for the periods longer than 4 days failed due to an out of memory error. Similar situation appears for the filtered time series querying presented in Table III but we may observe that processing speed is much worse due to more complex data selection from the database. However, the number of processed points is much smaller allowing for better response times. Although it was possible to run queries covering even 16 days, the response time took 108 seconds, which is totally unacceptable from a user's points of view.

B. Reduction of data complexity

One of the conclusions from the previous section is a need to reduce the number of time points and time series in the database. This can be achieved in two ways:

- by aggregating the points in time with downsampling, which can be called *horizontal compaction*
- by aggregating time series with grouping the tags for one metric, which can be called *vertical compaction*

Both solutions lead to a reduction of available information but may be acceptable if consulted with the user queries. For example, in a most typical example a user wants to get brief information about the system and would like to drill down if needed. Therefore both the detailed and coarse information

[1]In this paper we use an abstract query language with obvious semantics to express platform independent queries.

TABLE II
QUERYING AGGREGATING ALL TIME SERIES, DAYS: 1...4.

queried days	proc. time	data points	periods	series	pts/sec.
1	2.9	2,225,293	288	7,726	767,342
2	13.2	9,497,624	576	16,488	719,516
3	28.5	17,137,873	864	19,835	601,328
4	38.1	25,099,032	1,152	21,787	658,767

TABLE III
QUERYING FILTERED TIME SERIES, DAYS: 1...16.

queried days	proc. time	data points	periods	series	pts/sec.
1	6	793,907	288	2,756	132,317
2	8.4	1,295,810	576	2,249	154,263
3	14.1	2,284,509	864	2,644	162,021
4	20.6	3,250,740	1,152	2,821	157,802
16	108.9	12,479,745	4,608	2,708	114,598

should be kept in the system. In case of short time periods a user is interested in a detailed information, for the longer ones the details would not be visible anyway, therefore the coarse plots are just fine.

1) Downsampling Collectors: There are two possible ways to calculate the downsampled time series. The first one, the easiest, is to perform downsampling right during the data collection process. However, this would require to store aggregates for a long time according to the downsampling period (even one week or more). Keeping a collector's state for a longer time may be dangerous in case of a server malfunction. In industrial prototype we prefer solutions which are stateless, run frequently and returning results as quickly as possible. Therefore, we propose another way by implementing additional collectors which process data already stored in the database and send it back into another metrics after aggregation. The basic architecture of this solution is visible in Fig. 2 where the new collector is in gray colour. Obviously there should be one downsampling collector running for each downsampling time period. It is not working continuously but rather started every given time interval. For flexible querying it produces several metrics containing four different aggregations if sensible: *minimum*, *maximum*, *sum* and *average* of values for the sampled time period. Additionally to enable further average calculation it also counts a number of processed points and put the result into another metric called *count*. For some metrics (like Mbps) the summation downsampling does not make any sense and should be omitted.

2) Tags Grouping During Data Collection: During the prototype evaluation phase, we have observed that although a client could run queries containing any combination of tag values he is usually interested only in a subset of two or three tags. The queries concerned: geographical location; traffic per AS, http response code, origin server; and download speed. All time series need to be further divided per content providers. Therefore, instead of one metric with 9 tags, as it was described in II-B, we propose five metrics with six tags. In case of bandwidth metric (cdn.mbps) it could be:

- **`cdn.mbps-country`** with tags:
  `node, group, cdn, provider, origin, country`

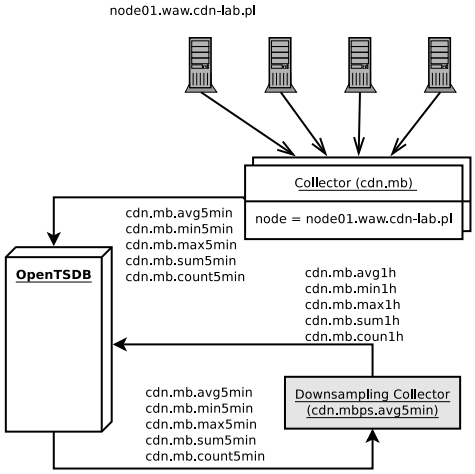

Fig. 2. The basic architecture of a downsampling collector.

- **`cdn.mbps-group`** with tags:
  `node, group, cdn, provider, origin, group`
- **`cdn.mbps-network`** with tags:
  `node, group, cdn, provider, origin, network`
- **`cdn.mbps-speed`** with tags:
  `node, group, cdn, provider, origin, speed-group`
- **`cdn.mbps-httpcode`** with tags:
  `node, group, cdn, provider, origin, httpcode`

Obviously, a user cannot display for example http codes for given AS or country with the above metrics and tags. However, these detailed queries can be processed with the original unoptimized metrics since selecting many tags greatly reduces the number of queried time series. Furthermore the query gets sensible number of data points. The optimized metrics should be used for general queries aggregating many time series. This optimization is a kind of a pre-aggregation done for certain query types.

This vertical aggregation can be run as a post-processor alike in downsampling or during the data collection process. Due to the architecture presented in our previous publication [11] it is much simpler to build it as a MOLAP cube with a reduced number of dimensions.

IV. RESULTS

Adding the vertical compaction optimization by aggregating time series during the data collection time has increased the log processing time for one CDN log covering data transfer for 5 minutes period from a single node by a factor of 3 from about 10 to 30 seconds. The log rotates every 5 minutes (300 seconds) so the time left can be used for processing log files even 10 times larger which will not be the case of a small or medium system.

The horizontal time compaction by the downsampling of the existing data and putting it back into the database as a new time series does not introduce any additional cost in an on-line log processing and does not need to be further studied here.

TABLE IV
SPEED-UP FOR QUERIES ON OPTIMIZED TIME SERIES.

days	1	2	3	4	16
speed-up total	3.2	11	17.8	16.5	
speed-up filtered	12	12	11.75	15.8	13.4

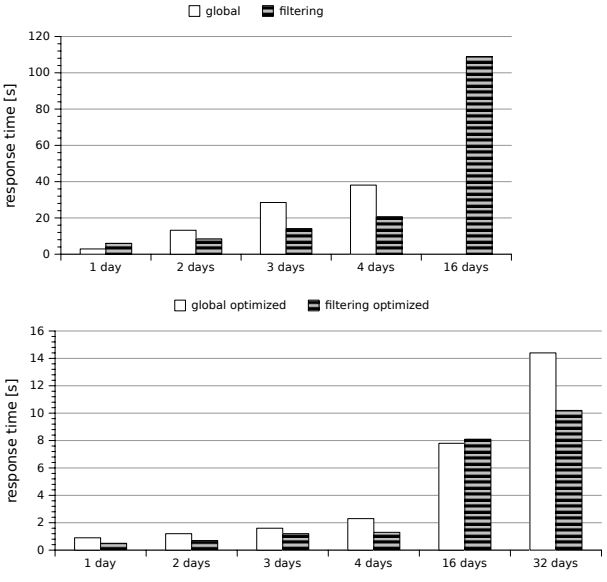

Fig. 3. Response times for initial unoptimized system (a) and final optimized version (b). The charts present two types of queries: global aggregating all time series and filtering performing selection before aggregation. For unoptimized system global query for more than 4 days could not be run due to out of memory error. The queries were run on time series without downsampling.

The reduction of time series and tags describing one metric resulted in an outstanding improvement of the database querying performance (see Table IV and Figure 3). This effect was possible first by the reduction of Hbase tables lookups. When processing queries, OpenTSDB first looks for IDs of the time series in a meta table. The more time series and tags, the more expensive this initial query can be. Then, the second great speed up is achieved by reducing the complexity of the time series needed to be grouped and interpolated in order to get the final time series which answers the query.

The response times for the same queries, but working on remodelled time series, are around 15 times faster for the longer time periods. This optimization decreased the amount of memory necessary to process the time series during the aggregation and therefore allowed for the processing of the times ranges longer than 4 days, which is obviously important for the real-life monitoring platforms.

V. CONCLUSIONS

We presented the problem of a small and medium scale CDN monitoring system in a case of limited resources which does not allow for real Big Data storage cluster. The initial state could not be accepted by the industrial requirements since the short term queries were processed too slowly and longer term queries could not be evaluated at all. Adding more resources by inserting computational nodes or increasing size of memory was not possible due to budget constraints. A significant improvement was achieved by introducing optimizations of the time series stored in the system. All initial properties of the system including the real-time processing and a fine grained data store allowed for detailed queries.

As the next step we plan to study the number of time series reduction allowing for arbitrary queries on an optimized series. This could be achieved by adding more metrics with reduced tag sets.

VI. ACKNOWLEDGMENTS

We appreciate many valuable comments from the anonymous reviewers of Federated Conference on Computer Science and Information Systems 2013. We thank Sara Oueslati, Paris, France for her great support with setting up the logging environment in the ISP CDN networks, and Marc Fiuczynski, New Jersey, USA for additional technical feedback.

REFERENCES

[1] A. Badam, K. Park, V. S. Pai, and L. L. Peterson, "Hashcache: Cache storage for the next billion.," in *NSDI*, pp. 123–136, USENIX Association, 2009.
[2] The Apache Software Foundation, "Apache Flume." http://flume.apache.org, 2013.
[3] The Apache Software Foundation, "Apache Hive." http://hive.apache.org, 2013.
[4] The Apache Software Foundation, "Apache Hadoop." http://hadoop.apache.org, 2013.
[5] B. Sigoure, "OpenTSDB scalable time series database (TSDB)." http://opentsdb.net, 2012. Stumble Upon.
[6] The Apache Software Foundation, "Apache HBase." http://hbase.apache.org, 2013.
[7] C. Chatfield, *The analysis of time series: an introduction*. Florida, US: CRC Press, 6th ed., 2004.
[8] OpenTSDB, "Whats opentsdb." http://opentsdb.net/, 2010-2012.
[9] "MARKETMAP ANALYTIC PLATFORM." http://www.sungard.com/fame, 2013.
[10] D. Shasha, "Time series in finance: the array database approach." http://cs.nyu.edu/shasha/papers/jagtalk.html.
[11] K. Kaczmarski and M. Pilarski, "Content delivery network monitoring," in *FedCSIS* (M. Ganzha, L. A. Maciaszek, and M. Paprzycki, eds.), pp. 633–639, 2012.
[12] L. Peterson, J. Hartman, and M. Pilarski, "A simple approach to cdn interconnection," *Internet-Draft*, no. draft-peterson-cdni-strawman-00, pp. 1–27, 2011.
[13] G. Bertrand, I. Oprescu, F. L. Faucheur, and R. Peterkofsky, "Cdni logging interface," *Internet-Draft*, no. draft-ietf-cdni-logging-04, pp. 1–41, 2013.