УДК 004.04, 004.6

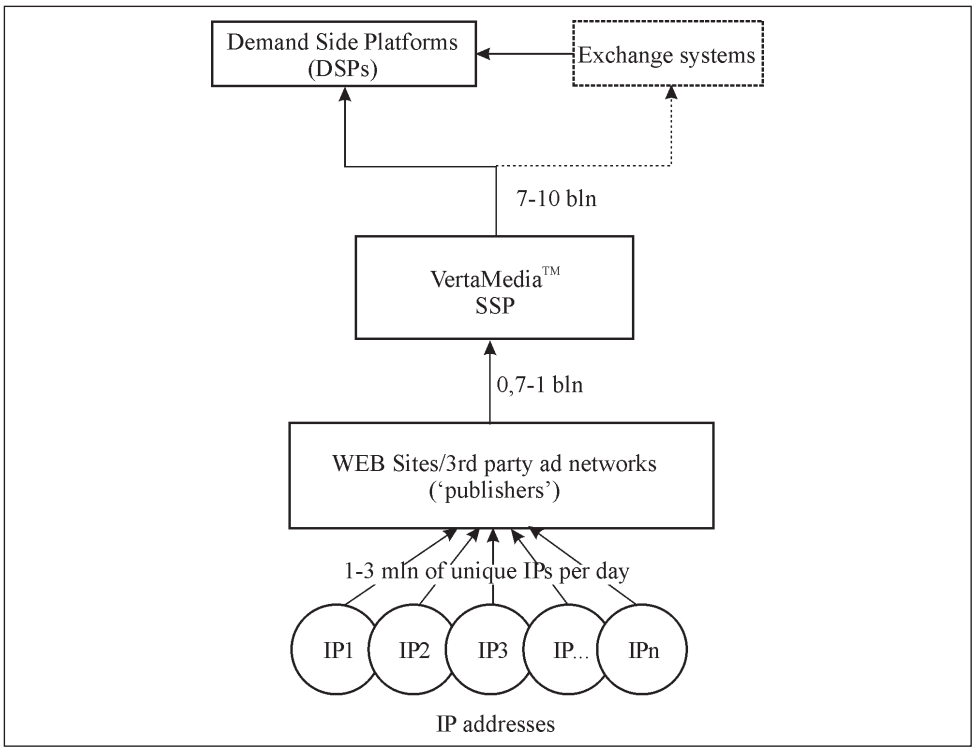
**A.V. Valialkin**
VertaMedia Company
(224 West 35th St., Suite 1102-5, New York, NY 10001, USA,
a.valialkin@vertamedia.com),
**O.I. Konashevych**, post-graduate
Pukhov Institute for Modelling in Energy Engineering
(15, General Naumov St., Kyiv, 03164, Ukraine,
a.konashevich@gmail.com)

# Real-time Method of Accurate Unique IPs Counting Across High Number of Distinct Dimensions and distinct Time Frames for Big Data Systems

The article describes a method which allows counting unique IP addresses within 10 bln of system events per day across high number of distinct dimensions (tuples). Log-based and probability-based methods showed unsatisfactory results. The proposed method allows avoiding excessive resource usage (RAM, CPU and persistent storage) as it appeared in a raw logs method and a probability method of counting. The method also avoids high statistic error for low cardinality as it appeared in a probability method. The main idea is to count unique IP addresses in distinct tuples in real time using RAM for short data interval processing, then flushing it to persistent storage, using merge algorithms to process and store unique IP counts in ordinary database from 5 minute, hourly, daily, weekly and monthly interval files.

Описан метод, позволяющий подсчитать число уникальных IP адресов из большого количества различных наборов данных (кортежей). Методы, основанные на сканировании логов и вероятностном подсчете, привели к неудовлетворительным результатам. Предложенный метод позволяет избежать чрезмерного использования ресурсов (процессора, оперативной и постоянной памяти), как при использовании метода сканирования необработанных логов и вероятностного метода подсчета, а также избежать большой статистической погрешности, как при использовании вероятностного метода на малых количествах уникальных значений. Основная идея метода состоит в том, что подсчет уникальных IP адресов в различных кортежах в реальном времени проводится в оперативной памяти. Обработка данных выполняется на коротких интервалах и затем они передаются в постоянную память с помощью алгоритма слияния. Обработанные счетчики IP адресов поступают в обычную базу данных из файлов с пятиминутным, часовым, суточным, недельным или месячным интервалом.

*K e y w o r d s: probability method, statistics, information technologies, queueing theory, big data, statistical process control.*




Scheme of requests statistics

**The Problem.** Ten billion network events across distinct dimensions per day caused VertaMedia™ company to search for suitable methods of statistics operating. As a glossary there are some words about professional sphere where the discussed methods were applied.

VertaMedia™ is a company which provides services for Internet video advertising with its own originally developed software as a supply side platform. Clients of the company are multiple ‘Publishers’. Demand Side Platforms (DSP) with Advertisers (these names are professional language) are partners, who use Publishers’ resources to promote advertisement. ‘Publishers’ are owners of advertising spaces on sites (‘spaces’ are usually called as ‘Inventory’); they can be owners of sites or professional ad operators.

Advertisers and Demand Side Platforms are companies which promote online video advertisement and are represented in VertaMedia™ system as a set of advertisement campaigns (‘campaign_id’). Advertisers represent the interests of initial clients, who are owners of promoted goods and services. Advertisers and Publishers deal with intermediaries who sell and buy Advertisement/Invento-

ries. 'Users' are sites visitors. They are those whom advertisements are shown, while they are navigating web pages.

VertaMedia faced with issues of arranging arrays of data to get accurate statistics. Statistical data structure consists of unique IP addresses traffic coming to Publishers' sites and related with such IPs requests. The scheme of requests and scope of structure array is shown on Figure.

Figure shows that up to 3 mln unique IP addresses generate daily up to 1 bln requests to VertaMedia$^{TM}$ ad servers and cause these servers to operate up to 10 bln inquiries and replies to/from DSP.

Collecting and operating the traffic statistics of 10 bln events per day appears as a nontrivial task as it required notable resources of CPUs, RAM and persistent storage. Statistics collecting was of business interest of the company since it allowed its managers to make more accurate decisions and to control the quality of services.

The number of unique IP addresses each publisher sends to each DSP is an important metric used by traffic analysts in VertaMedia$^{TM}$. This allows optimizing traffic flows for maximizing company's and customers' profits.

**Body Section.** Company's researchers formulated the initial task: to create a system algorithm that can quickly and accurately with minimum resources expenditure respond to the question: "How much unique IP addresses caused the events in the ad network with a specific period of time (the last 5 minutes, hour, day, week and month) for given tuples: (country, publisher_id[1], source_id[2], campaign_id[3]), (country, publisher_id, source_id), (country, publisher_id, campaign_id), (country, publisher_id), (country, campaign_id); within the number of events of 10 billion per day".

Speed processing requirements were identified in the range of a couple of seconds. At the first stage of developing statistics results were processed by managers, therefore time of processing was not critical. However, company considers to add automation. The developed methods show good processing time and could be applied for further autoprocessing.

The first research activities showed that the obvious solution was less acceptable. The idea was to store logs of values (time, publisher_id, source_id, campaign_id, ip) and count the required data 'on the fly'. This approach required too much recourses of disk space to store — about 1 Tb per day.

Another tested method — probabilistic counting [1] — did not fit mostly because of lack of accuracy and high requirements to RAM. The probabilistic

---

[1] Publisher's ID.

[2] Publisher's sub-ID which belongs to some web site under publisher's control.

[3] Advertiser's ID.

method is one of fundamentals of Queueing theory. Company's researchers considered that some of methods could give acceptable results. M. Harchor-Balter emphasizes that queueing theory is built on a much broader area of mathematics called stochastic modeling and analysis [2, 3]. Markovian assumptions [4], such as assuming exponentially distributed service demands or a Poisson arrival process [3], greatly simplify the analysis; hence much of the existing queueing literature relies on such Markovian assumptions [2, 3]. However, in some cases Markovian assumptions are very far from reality [2, 3]. After theoretical analysis it was decided to make express experiments with HyperLogLog which is most close to initial tasks and the program implemented in open libraries in GitHub [5].

HyperLogLog is an algorithm for the count-distinct problem, approximating the number of distinct elements in a multiset [6]. Calculating the exact cardinality of a multiset requires an amount of memory proportional to the cardinality, which is impractical for large data sets with high cardinality. Probabilistic cardinality estimators, such as the HyperLogLog algorithm, use significantly less memory than this, at the cost of obtaining only an approximation of the cardinality. The HyperLogLog algorithm is able to estimate cardinalities of >10^9 with a typical error rate of 2 %, using 1.5 kB of memory [6].

*Table 1*

```
stat=> select count(*) from
requests_unique_ip_day_campaign_id_publisher_id_source_id_country where time >
now() - interval '2 days'; count
_______
587254
(1 row)
```

*Table 2*

```
stat=> select count(*) from
requests_unique_ip_day_campaign_id_publisher_id_source_id_country where ips_count >
1000 and time > now() - interval '2 days';
count
_______
51720
(1 row)
```

*Table 3*

```
stat=> select count(*) from
requests_unique_ip_day_campaign_id_publisher_id_source_id_country where ips_count =
1          and time > now() - interval '2 days';
count
_______
96180
(1 row)
```

As a result of research it was found that only less than 10 % (51,720 of 587,254) of tuples had more than 1000 of unique IP addresses, which is shown in two SQL requests examples below (Table 1 and Table 2).

It was also found that the number of tuples with one IP address exceeds 20 % (96,180 of 587,254) (Table 3).

Experiments showed that tuples with numbers of IPs less than 1000 gave high statistical error which was by an order of magnitude higher than that of an accurate calculation. Another reason of failure with this approach was that this method required too much on-line memory.

In order to estimate the number of unique IPs a separate memory object should be kept for each tuple (time_interval, publisher_id, source_id, campaign_id). The number of tuples exceeds one million and may increase by several orders at any time

*Table 4*

```
stat=> select publisher_id, count(*) from requests_day_campaign_id_publisher_ id_source_ id_
country where time > now() - interval '2 days' group by 1 order by 2 desc limit 10;
 publisher_id  | count
--------+---------
13346 | 207406
13280 |  67484
12540 |  59002
11752 |  45738
12776 |  30929
12046 |  26520
12139 |  21420
13540 |  13992
12333 |  13030
12070 |  11980
(10 rows)
```

*Table 5*

```
stat=> select publisher_id, source_id, count(*) from requests_day_campaign_id_publisher_id
_source_id_country where time > now() - interval '2 days' group by 1,2 order by 3 desc limit 10;
 publisher_id  | source_id  | count
-----------+-------------+--------
12540     |   1918     |   950
12540     |   1919     |   918
12540     |   1967     |   899
12540     |   1926     |   899
12540     |   1940     |   871
12540     |   1966     |   870
12540     |   1809     |   856
12540     |   1917     |   855
12540     |   1910     |   853
12540     |   1911     |   841
(10 rows)
```

when ‘source_id’ cardinality increases. The size of each object starts from 1 Kb. However the more it is, the higher is the resulting accuracy. But this solution does not scale well for large amounts of memory for tuples.

Consequently research efforts were concentrated on searching of our own original decision. The hypothesis to keep a list of unique addresses for each tuple and to store it in operating memory did not work because a large amount of memory with fast random access was required.

Another hypothesis was to keep a list of unique tuples for each IP. This solution was similar to the first hypothesis regarding the memory consumption. But it was easier to count the number of unique IPs for incomplete tuples, for example, (publisher_id, source_id), (publisher_id, campaign_id) etc.

Finally, it was decided to store the structure of data in a file sorted by IP, publisher_id, source_id, campaign_id. Sorting is necessary for the subsequent merging of a new data with the existing data in the file in streaming mode, without use of additional memory and without random I/O.

This approach made it possible to reduce RAM consumption from a couple of dozens of gigabytes to hundreds of megabytes during merging moments and moments of counters reset of unique IPs through all tuples. In this way researchers have obtained the method which can form tuples shown in request examples below.

The number of unique IP counters for each (source_id, campaign_id, country) tuple per Publisher (‘publisher_id’) for the current day in Table 4.

The number of unique IP counters for each (campaign_id, country) tuple per site (‘source_id’) in the context of a Publisher (‘publisher_id’) for the current day in Table 5.

*Table 6*

```
stat=> select publisher_id, source_id, campaign_id, count(*) from requests_day_campaign_
id_publisher_id_source_id_country where time > now() - interval ‘2 days’ group by 1,2,3 order
by 4 desc limit 10;
    publisher_id  | source_id  | campaign_id | count
    ----------+----------+-----------+----
    12540   | 1918  |   1024 | 181
    12540   | 1918  |    968 | 181
    12540   | 1918  |   1073 | 180
    12540   | 1926  |    968 | 180
    12540   | 1919  |    968 | 179
    12540   | 1918  |   1015 | 178
    12540   | 1970  |   1015 | 177
    12540   | 1967  |   1024 | 175
    12540   | 1919  |   1073 | 175
    12540   | 1967  |   1073 | 174
    (10 rows)
```

The number of unique IP counters for each country per Advertiser's Campaign ('campaign_id') in the context of some Publisher's site (respectively - 'publisher_id' and 'source_id') for the last day in Table 6.

The number of distinct (country, publisher_id) tuples per month in Table 7.

The number of distinct (country, publisher_id, source_id) tuples per month in Table 8.

The number of distinct (country, publisher_id, source_id, campaign_id) tuples per month in Table 9.

The number of distinct (country, campaign_id) tuples per month in Table 10.

Top unique IP counters ('ips_count') for the tuple (campaign_id, publisher_id, source_id, country) for a current day in Table 11.

A sample of the minimum number of unique IPs ('ips_count') for the tuple (campaign_id, publisher_id, source_id, country) for a current day in Table 12.

Other combinations are possible, as it works as an ordinary SQL database.

*Table 7*

```
stat=> select count(distinct country || ',' || publisher_id) from
requests_unique_ip_month_campaign_id_publisher_id_source_id_country where time >
now() - interval '60 days';
  count
---------
  1568
```

*Table 8*

```
stat=> select count(distinct country || ',' || publisher_id || ',' || source_id) from
requests_ unique_ip_month_campaign_id_publisher_id_source_id_country where time >
now() - interval '60 days';
  count
--------
129516
(1 row)
```

*Table 9*

```
stat=> select count(distinct country || ',' || publisher_id || ',' || source_id || ',' || campaign_id)
from requests_unique_ip_month_campaign_id_publisher_id_source_id_country where time >
now() - interval '60 days';
  count
---------
2684197
(1 row)
```

The mentioned tuples are stored in RAM for the period of 5 minutes and then merged with the file containing unique IP data for the current hour. ‘Hour’ file is merged with ‘Day’ file every hour, while ‘Day’ file is merged into ‘Week’ and ‘Month’ file every day.

*Table 10*

```
stat=> select count(distinct country || ',' || campaign_id) from
requests_unique_ip_month_campaign_id_publisher_id_source_id_country where time >
now() - interval '60 days';
        count
        -------
        3115
        (1 row)
```

*Table 11*

```
stat=> select * from
requests_unique_ip_day_campaign_id_publisher_id_source_id_country where time >
now() - interval '2 days' order by ips_count desc limit 10;
                time               |campaign_id| publisher_id| source_id | country |
ips_count
        ----------------------+----------+---------+---------+-------------+--------------
        2016-04-05 00:00:00+00 |  272 | 12392 | 11278 |   US  | 502561
        2016-04-05 00:00:00+00 |  217 | 12392 | 11278 |   US  | 497580
        2016-04-05 00:00:00+00 | 1209 | 12392 | 11278 |   US  | 494679
        2016-04-05 00:00:00+00 |  552 | 12392 | 11278 |   US  | 491190
        2016-04-05 00:00:00+00 |  992 | 12392 | 11278 |   US  | 485995
        2016-04-05 00:00:00+00 | 1177 | 12392 | 11278 |   US  | 485951
        2016-04-05 00:00:00+00 | 1026 | 12392 | 11278 |   US  | 478815
        2016-04-05 00:00:00+00 |  272 | 12392 | 10604 |   US  | 456033
        2016-04-05 00:00:00+00 |  217 | 12392 | 10604 |   US  | 450713
        2016-04-05 00:00:00+00 | 1026 | 12392 | 10604 |   US  | 448131
        (10 rows)
```

*Table 12*

```
stat=> select * from
requests_unique_ip_day_campaign_id_publisher_id_source_id_country where time >
now() - interval '2 days' order by ips_count limit 10;
        time               | campaign_id| publisher_id | source_id |  country | ips_count
        ----------------------+---------+-----------+-----------+-------------+-----------
        2016-04-05 00:00:00+00 |  21  |  11752  |  37718  |  AU  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  30649  |  AU  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  30653  |  AU  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  35207  |  CA  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  19011  |  US  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  22799  |  CA  |     1
        2016-04-05 00:00:00+00 |  21  |  11400  | 222831  |  CA  |     1
        2016-04-05 00:00:00+00 |  21  |  11400  |  89986  |  CA  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  44925  |  CA  |     1
        2016-04-05 00:00:00+00 |  21  |  11752  |  38779  |  CA  |     1
        (10 rows)
```

**Processing Algorithms.** The basic data structure is a map keyed by IPs with values containing a set of distinct (publisher_id, source_id, campaign_id) tuples. Each incoming event updates the map unless the map contains data (publisher_id, source_id, campaign_id) for the corresponding IP. The map contains data for the last 5 minutes. Limiting in-memory map to a short period of time (5 minutes in this case) allows limiting RAM usage (to 200MB in our case). The map size is proportional to the number of distinct (ip, publisher_id, source_id, campaign_id) tuples for the given time interval. Map update speed is O(1), i.e. it does not depend on the map size.

Flushing in-memory map into a file. The output is sorted by IP, then by (publisher_id, source_id, campaign_id). This opens the possibility for the next step. Flushing requires only a small constant amount of RAM.

Merging two files (5min -> hourly, hourly -> daily, daily -> weekly, daily -> ->monthly). Merge is similar to the merge pass used in merge sort [7]. Merging step requires only a small constant amount of RAM.

Counting unique IPs per (country, publisher_id, source_id, campaign_id), (country, publisher_id), (country, publisher_id, source_id), (country, campaign_id) by linear scan of the files. The counting occurs every hour for hourly statistics, every day for daily statistics etc. Counting step requires RAM size proportional to the number of unique (publisher_id, source_id, campaign_id) tuples (O(dimensions count)).

***Scalability.*** Algorithms mentioned above have good scalability:

Per-IP in-memory maps may be populated concurrently by arbitrary number of worker processes (threads).

Files to be merged may be split into arbitrary number of IP ranges, so ranges may be merged concurrently.

Unique IPs' counting may run concurrently on distinct IP ranges.

For example, in-memory maps can be divided into a few independent parts and updated independent of each other in different threads (processes, machines). This also makes the algorithm suitable for MapReduce-like processing [8].

***Performance.***

Currently the implementation processes up to 200K events per second on a single CPU core.

RAM usage does not exceed 200MB during per-IP in-memory map updating. RAM usage does not exceed 1GB during unique IPs counting step.

Files are merged by a single thread (CPU core) at the speed of 40K unique IPs per second.

Given the perfect scalability of the algorithm, it should be able to process up to 200K*40=8M events per second on a single machine containing 40 CPU cores (which is typical datacenter hardware). This results in 8M*3600*24 = ~700G

events per day per a single machine. The processing will require only 200MB*40 = 8GB of RAM, since unique IPs counting step (which requires 1GB of RAM per CPU core) may be performed serially across CPU cores.

**Conclusion.** As a summary it is noted that the method which was developed by VertaMedia$^{TM}$ researchers can be applied to similar tasks in different spheres, that it is interesting from the methodological point of view. In particular it could be applied for data processing in Statistical process control, which uses checklists to help to distinguish the "special" causes from the variations of the "usual" reasons to provide quality control [9].

Let us name key positions research outputs:

- neither logs nor probability could not satisfy the needs for accurate statistics of unique IPs in the range of 10 bln events daily because of high demands for resources and too remarkable statistical error;
- the main point of the method is to distinct wanted tuples and to operate them in RAM within 5 minutes and then to store the results in a persistent memory;
- processed data set is stored in files in selected periods of time (ex. 5 minutes, hourly, daily, weekly, monthly);
- DB statistics files (by periods) is updating according to the merging algorithm;
- flushing unique IP counters to db after linear scan of files.

Developed methods have shown high efficiency in daily 10 bln system events flow in resources usage (CPUs, RAM and persistent memory) compared to methods of logs counting and probability methods. Finally, developed methods are high scalable as could be used in distributed systems and process even more data in several independent threads with minimum resources usage.

Описано метод, який дозволяє підрахувати кількість унікальних IP адрес із великої кількості різних наборів даних (кортежів). Методи, базовані на скануванні логів та імовірнісному підрахунку привели до незадовільних результатів. Запропонований метод дозволяє уникнути надмірного використання ресурсів (процесора, оперативної та постійної пам'яті), як це відбувається при використанні метода сканування необроблених логів та імовірнісного методу підрахунку, а також уникнути великої статистичної похибки, як при використанні імовірнісного метода на малих кількостях унікальних значень. Основна ідея методу полягає в тому, що підрахунок унікальних IP адрес в різних кортежах в реальному часі проводиться в оперативній пам'яті. Обробка даних виконується на коротких інтервалах і потім вони передаються у постійну пам'ять згідно з алгоритмом злиття. Оброблені лічильники IP адрес надходять з файлів у звичайну базу даних з п'ятихвилинним, годинним, добовим, тижневим або місячним інтервалом.

REFERENCES

1. Erdös, P. (1959), available at: http://cms.math.ca/10.4153/CJM- 1959-003-9 (accessed April 4, 2016).
2. Harchol-Balter, M. (2013), Performance Modeling and Design of Computer Systems. Queueing Theory in Action, Cambridge University Press, New York, USA.

3. Cox, D.R. and Isham, V.I. (1980), Point Processes, Chapman & Hall, London, UK.
4. Durrett, R. (2010), Probability: Theory and Examples (4th ed.) Cambridge University Press, Camridge, USA.
5. Available at: https://github.com/clarkduvall/hyperloglog (accessed October 21, 2015).
6. Flajolet, P., Fusy, E., Gandouet, O. and Meunier, F. (2007), HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm, *Proceedings of the 2007 International Conference on the Analysis of Algorithm(AOFA '07)*, available at: http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf (accessed April 4, 2016).
7. Knuth, D. (1998), "Section 5.2.4: Sorting by Merging", The Art of Computer Programming 3 (2nd ed.), Addison Wesley, USA *Sorting and Searching.*
8. Dean, J. and Ghemawat, S. (2004), MapReduce: Simplified Data Processing on Large Clusters, available at: http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf (accessed April 4, 2016).
9. Shewhart, W. (1931), Economic Control of Quality of Manufactured Product, D.Van Nostrand Company, New York, USA. ISBN 0-87389-076-0.

Received 20.04.16;
After revision 28.04.16

*VALIALKIN Aliaksandr Valerievich, Backend developer, VertaMedia Company, USA. Belarussian State University of Informatics and Radioelectronics, Automated Control in Technical Systems, 2005. The field of research — systems design, systems performance optimization, high load systems.*

*KONASHEVYCH Oleksii Ihorovych is a post-graduate student of the Pukhov Institute for Modeling in Energy Engineering of NAS of Ukraine; graduated from the National Aviation University in 2005; in 2011 he graduated from Kyiv National Trade and Economic University, Advanced Training Institute. The field of research — blockchain technology.*