Big Data Normalization for Massively Parallel Processing Databases

Nikolay Golov[1] and Lars Rönnbäck[2]

[1] Avito, Higher School of Economics, Moscow, Russia
ngolov@avito.ru, http://www.avito.ru
[2] Department of Computer Science, Stocholm University, Stockholm
lars.ronnback@anchormodeling.com, http://www.anchormodeling.com

Abstract. High performance querying and ad-hoc querying are commonly viewed as mutually exclusive goals in massively parallel processing databases. In the one extreme, a database can be set up to provide the results of a single known query so that the use of available of resources are maximized and response time minimized, but at the cost of all other queries being suboptimally executed. In the other extreme, when no query is known in advance, the database must provide the information without such optimization, normally resulting in inefficient execution of all queries. This paper introduces a novel technique, highly normalized Big Data using Anchor modeling, that provides a very efficient way to store information and utilize resources, thereby providing ad-hoc querying with high performance for the first time in massively parallel processing databases. A case study of how this approach is used for a Data Warehouse at Avito over two years time, with estimates for and results of real data experiments carried out in HP Vertica, an MPP RDBMS, are also presented.

Keywords: Big Data, MPP, database, normalization, analytics, ad-hoc, querying, modeling, performance

1 Background

Big Data analytics is rapidly becoming a commonplace task for many companies. For example, banks, telecommunication companies, and big web companies, such as Google, Facebook, and Twitter produce large amounts of data. Nowadays business users also know how to monetize such data. For example, various predictive marketing techniques can transform data about customer behavior into great monetary worth. The main issue, however, remains to be implementations and platforms fast enough to execute ad-hoc analytical queries over Big Data [2]. Until now, Hadoop has been considered a universal solution, but it has its own drawbacks, especially in its ability to process difficult queries, such as analyzing and combining heterogeneous data, and performing fast ad-hoc analysis [5].

This paper introduces a new processing approach, using Anchor modeling in massively parallel processing (MPP) databases. The approach significantly

increases the volume of data that can be analyzed within a given time frame. It has been implemented in HP Vertica [6], a column-oriented MPP RDBMS, and is used on a daily basis for fast ad-hoc query processing at Avito, a popular Russian web site for classified ads [10]. The approach gives their data scientists an ability to execute complex queries that process terabytes of data in minutes. The approach is generic and should apply equally well to other MPP databases, such as Pivotal Greenplum, Actian Matrix, and Amazon Redshift. The paper starts by describing the case for normalized Big Data at Avito, with subsections on Anchor modeling, benefits, theoretical estimates, and practical verification of the approach. The paper ends with the drawn conclusions.

2 The Avito Case for Normalized Big Data

Big Data is commonly defined using the "3Vs": *Volume* (large amounts of data), *Variety* (various forms and evolving structure), and *Velocity* (rapid generation, capturing, and consumption) [2]. Log files are common sources of structured Big Data. Web servers record logs of detailed user actions, click rates, visits, and other property records of web users. The sequence of pages visited by within a particular website is known as the *clickstream* of the user. Clickstreams are analyzed to understand traffic, the number of unique visitors, sessions, and page views. Clickstreams of groups of users often follow distinct patterns, the knowledge of which may help in providing customized content [1]. They may, however, also be generated by a non-human activity. Fake identities and Sybil accounts are responsible for a growing number of threats, including fake product reviews, malware, and spam on social networks. Similar clickstreams can be grouped into behavioral clusters to detect and eliminate non-human accounts [9]. Identification and elimination of a non-human activity is important in all analytical tasks, such as proper traffic estimation and pattern detection. It may also have significant reputational, ethical, and even legal effects if left unattended.

Clickstream analysis was one of the main defined objectives for the Data Warehouse at Avito. Based in Moscow, Avito is Russia's fastest growing e-commerce site and portal, "Russia's Craiglist". It grows $\approx 50\%$ a year and now second only to Craiglist and Chinese site 58 in the rating of classified sites [10]. In terms of 3Vs, Avito clickstream data have over 600 million user actions a day (Volume), a business model that is constantly evolving, where new features are constantly added (Variety), and users perform up to 1 million actions per minute. User profiles, which help to reject non-humans and generate personalized content, have to be recalculated in near real-time (Velocity).

The BI team at Avito was challenged to develop a scalable Data Warehouse, that could grow in volume and complexity together with their business model, while being able to support analytical workloads, such as clustering analysis, correlation analysis, A/B testing (two-sample hypothesis testing), and Data Mining Algorithms. Hadoop and other NoSQL approaches were rejected in the process, and instead an MPP relational database, HP Vertica [6], and highly normalized data model, Anchor Modeling [7], were selected.

2.1 Anchor Modeling

Anchor modeling [7] is a database modeling technique resulting in implementations where tables are in 6NF, the sixth normal form. Entities and relationships in Anchor modeling are highly decomposed. In 6NF tables have no non-trivial join dependencies [3], making tables *narrow* with few columns in comparison to, for example, the *wide* tables of 3NF. The traditional concept of an entity is thereby spread out over many tables, referred to as an *ensemble* [4]. Massively parallel processing databases generally have shared-nothing scale-out architectures, such that each node holds some subset of the database and enjoy a high degree of autonomy with respect to executing parallelized parts of queries. In order to maximize utilization, each node should perform as much of its assigned work as possible without the involvement of other nodes. The following four constructs are used in Anchor modeling, all having a predefined distribution.

Anchor, table holding surrogate identifiers for instances of an ensemble. Each instance in the modeled domain has its own unique surrogate identifier and they are stored in anchors. Surrogate identifiers are immutable and assumed as the only part of an instance that cannot change over time. Anchors are distributed across the nodes by a modulo operation on a hash of the surrogate identifier, such that no duplication exists.

Attribute, table holding named property values for an ensemble, that cannot be described as ensembles in their own right. An attribute table holds the surrogate identifier of the instance and the property value, with an optional history of changes to those values. Attributes share the same distribution scheme as anchors, which keeps an instance of an ensemble with its history together on the same node.

Tie, table holding a relationship between ensembles, distinguished by the roles those ensembles play. Tie tables have one column for each involved role, holding a reference to a surrogate identifier. Ties are distributed across the nodes for each role, duplicating subsets of the tie such that all relationships that an instance takes part in can be resolved without the involvement of other nodes.

Knot, table holding a set of enumerated values. If the possible values of an attribute fall within a, usually small, finite set of values, or a tie represents a relationship which has or may change categories, such values are best represented through knots. Knots hold surrogate identifiers for every value and the value itself, where values should be unique, mutually exclusive and exhaustive. Knots are fully duplicated on every node.

Attributes and ties may be *static* or *historized* depending on if they keep a record of changes over time. Historized tables contain an additional column indicating since when a value or relationship is in effect. Attributes and ties may also be *knotted* in which case they contain a reference to a value in a knot table, rather than an actual value. All tables may also contain technical columns, such as a reference to custom metadata.

2.2 The Evolution of the Avito Data Warehouse

The first version of the Data Warehouse (DW) at Avito was built in 2013 using Anchor modeling, contained 10TB of data, and ran on an HP Vertica cluster of 3 nodes. It loaded data from two data sources; the back office system at Avito and clickstream web logs. Since then, the DW has grown, and the current size of the Avito data warehouse has been limited to 51Tb for licensing reasons. It now contains years of consistent historical data from 14 data sources (back office, Google DFP/AdSense, MDM system, CRM system, RTB systems, among others), and a rolling half year of detailed clickstream data. The cluster has been increased from 3 to 12 nodes in order to scale up performance.

Clickstream data are loaded every 15 minutes. At the beginning of 2014 each such batch contained 5 million rows (≈1.5GB) and 15 million (≈5Gb) one year later. Avito has evolved their data model over the years. The clickstream records originally had less than 30 attributes, while now containing more than 70. Clickstream data has grown many times, both in terms of velocity (number of rows per minute), volume (size), and variety (number of attributes). The growth was successfully handled through scaling up the cluster by the addition of nodes.

ETL processes are implemented using Python. Data sources are loaded using different approaches: clickstream is loaded using FluentD with MongoDB as intermediate cash, back office data are loaded using intermediate CSV files, and data from Google DFP and CRM system are loaded through web services. The current version of the Avito DW contains ≈ 200 anchors, ≈ 600 attributes, and ≈ 300 ties, loaded from the data sources. Some ties and attributes are historized, some are not. There are two distinctive modes of ETL processes:

- Increment from operational database. Characterized by a small number of rows from a large number of source tables and source columns, with most ties and attributes historized. Data is loaded every 4 hours, taking 30 minutes, from 79 source tables to 45 anchors, 83 ties, and 238 attributes. The largest source delta contains ≈1 million rows.
- Increment from clickstream. Characterized by a large number of rows from a small number of source tables and source columns, with most ties and attributes static. Data is loaded every 15 minutes, taking 15 minutes, from 16 source tables to 16 anchors, 39 ties, and 43 attributes. The largest source delta contains ≈10-15 million rows.

2.3 Beneficial Effects of Normalization

An important effect of Anchor modeling is the ease of which new attributes, ties, and anchors can be added to the model, only resulting in new tables. The creation of such are close to instantaneous and populating them with data causes no locks for existing operations and ETL processes. Applications remain unaffected, since the existing schema is non-destructively extended [7], and can be dealt with to incorporate new features when time permits.

When data is sparse, arising though the addition of new attributes or when an attribute does not apply to all instances, normalization provides another

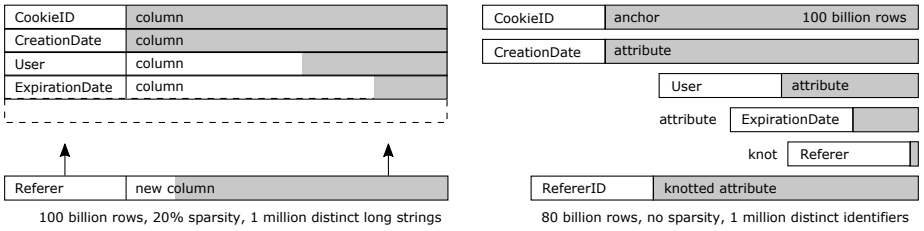

Fig. 1. Extending a less normalized model (left side) and a corresponding Anchor model (right side).

benefit. Only actual values are stored and "nulls" are represented by the absence of rows. For example, when less than half of the cookies have a known user, as in Fig.1, the attribute contains fewer rows. Furthermore, when the number of distinct values is relatively low compared to the total number of values, knotted attributes can be used. Rather than repeating the relatively few long strings representing referers (URLs), these are stored as unique values of a knot. The knotted attribute instead contains identifier references, much smaller than the strings they represent. A query with a condition on the referer, such as containing a particular substring (UTM mark detection), can then be computed much more efficiently. The licensing cost of Vertica depends on "raw data size", the size of comma-separated lists of values in tables. In the less normalized model, referer strings are repeated 80 billion times, but only 1 million times in the Anchor model. By using Anchor modeling, Avito were able to store substantially more data without affecting its licensing cost.

2.4 Reporting and Ad-hoc Analysis

Using Anchor modeling with a Vertica cluster has proven beneficial at Avito for data modeling, data loading, and data maintenance. Though, from their experience, it has required some additional efforts to implement reporting and ad-hoc analysis. Reporting is considered as frequently spreading information, based on the same queries, to many recipients. Execution time of those queries cannot be longer than a few seconds. Ad-hoc querying is to test hypotheses, train models, or explore data to find suitable reports, using new queries. While shorter execution times are preferable, it is acceptable that these queries run for minutes or even hours in some cases.

Reporting can be implemented by creating dedicated denormalized data marts and using specialized BI software. Such data marts are implemented as regular views or materialized views. For reporting, the experience of Avito is that it is impossible to avoid data marts. The Anchor model stores fully historized versions of instances, which business users are not used to. There may also be the need to impose business logic or aggregate data in the marts. The development of efficient such views, as well as ad-hoc analysis, is based on the creation of high performance queries. While the optimizer in Vertica generally does a good

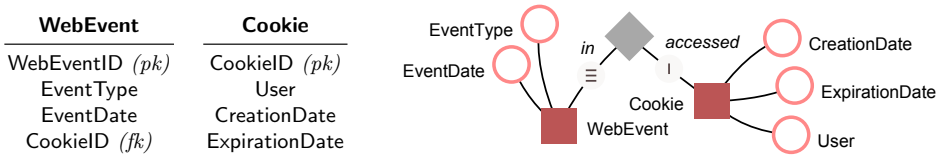

Fig. 2. An example 3NF model (left tables) and corresponding Anchor model (right diagram) of click stream data.

job, when data is big, starting from ≈100 million rows, it may produce poor execution plans.

The BI team at Avito performs ad-hoc queries accessing hundreds of billions of rows in a Anchor model on a daily basis. Because of this, a query modification approach was designed. The approach aims to optimize the query execution plan according to the highly normalized data model. The following sections will describe a sample task illustrating the approach. Its benefits in comparison to the execution plan in a denormalized data model are presented.

2.5 Theoretical Estimates and Practical Verification

The largest data set available at Avito for testing purposes is the clickstream data set. It will be compared using an Anchor model and a 3NF model, seen in Fig.2, having exactly the same information content. In the models WebEventID is the surrogate identifier of a clickstream record, EventDate is the date and time of an event, EventType is its type, such as 'search', 'message', or 'purchase', CookieID is a surrogate identifier of a cookie, User is the user bound to the cookie, CreationDate is when it was created, and ExpirationDate is when it expires. These models are simple, and only represent a small subset of the actual model used at Avito, but sufficient in order to verify the expected benefits.

Experiments were made in the column-oriented massively parallel processing relational database HP Vertica v.7.1.0.3, with cluster sizes of 5, 10, and 12 nodes, where each node is an HP ProLiant DL380p Gen8 server with double Intel Xeon E5-2680 v2 CPUs, 256GB RAM, 25 * 300GB SAS 15k SFF 2.5" HDDs connected through a RAID card with 2GB cache, and an HP 530SFP+ Ethernet 10Gbit 2-port LAN-adapter for communication. Since the experiments were carried out in a production environment, daily operations, such as ETL and reporting, may have affected the results. In order to minimize such effects, experiments were repeated several times and averages calculated. The two models in Fig.2 were populated with data, consisting of approximately 50 billion actual web events accessing 3.4 billion cookies.

2.6 Scenario, Query Optimization, and Results

The chosen ad-hoc query is to calculate the number of unique users who triggered page view events (condition on EventType) during February 2014 (condition on

```
select count(distinct c.User)
from Cookie c
join WebEvent we
on we.CookieId = c.CookieId
where date_trunc('month',
      c.CreationDate) = '2014−02−01'
and c.CreationDate = we.EventDate
and we.EventType in (42, 43)
```

```
select count(distinct u.User)
from Cookie_WebEvent tie join User u
on u.CookieID = tie.CookieID
join CreationDate cd on cd.CookieID = tie.CookieID
join EventDate ed on ed.WebEventID = tie.WebEventID
join EventType et on et.WebEventID = tie.WebEventID
where date_trunc('month', cd.CreationDate) = '2014−02−01'
and cd.CreationDate = ed.EventDate
and et.EventType in (42, 43)
```

Fig. 3. Corresponding queries in the 3NF model (left) and the Anchor model (right).

EventDate), and have new cookies (CreationDate of the cookie is the same date as the EventDate). The query was selected for estimation, because it is typical, it is simple for reading, and it requires great amount of system resources to operate. The SQL code for the queries, as they would look in the 3NF model and the Anchor model, can be seen in Fig.3.

The execution plan in the 3NF model is the one selected by the Vertica query optimizer. Many modern query optimizers can make use of column statistics, holding information about the distribution of values, to determine the optimal join order. The most selective conditions in a query are applied first, yielding successive intermediate result sets between joins that have as few rows as possible. Since a non-optimal plan was chosen by the optimizer, the execution plan in the Anchor model was forced by producing intermediate steps by hand. The hand made plan demonstrates that efficient query execution plans for complex analytic queries in an Anchor model exists. Based on such, the BI team at Avito implemented a framework for semi-automatic generation of efficient plans for Big Data. Future generations of Vertica optimizers can be complemented with similar logic. The statistics of the sample set is as follows.

- R, row count of Cookie anchor table ≈ 3.4 billion.
- S, attribute value selectivity expressed as selection of "one out of S".
  Condition on CreationDate reduces data to $1/S_1 \approx 1/3.01$.
  Condition on EventType reduces data to $1/S_2 \approx 1/3.20$.
  Condition on EventDate = CreationDate reduces data to $1/S_3 \approx 1/3.17$.
- M, average number of events per cookie. $M \approx 50/3.4 \approx 15$.
- A, average size of each column = 8 bytes.
- P, size of disk page = 4000 bytes.

For **3NF**, according to [8], the optimizer will choose an early materialization strategy for the smaller, inner side, of the join. The experiments confirmed this, with the following execution plan.

1. Read CookieId, User, CreationDate columns from disk to RAM, according to EM-pipeline strategy [8]. Data is filtered by the condition on CreationDate.
2. Read disk pages to match remaining CookieId keys in the WebEvent table. A hash join is used, as well as a resegmentation of the key between nodes.
3. Load EventDate and EventType for the matched WebEvent rows. Data is filtered by the condtition on EventType.

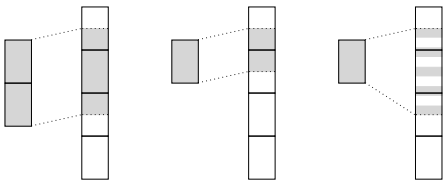

Fig. 4. When a part of a join is reduced, from the two disk pages (left) to one page (center, right), the number of pages needed to be read in order to produce the join depend on whether the joined keys are concentrated (center) or spread out (right). For an arbitrary join, the number of pages to read is normally significantly reduced.

4. Filter loaded data by the condition on EventDate and CreationDate, calculating the number of unique User values.

RAM usage can be estimated as $\frac{3*R*A}{S_1} + \frac{M*R*A}{S_1} + \frac{R*A*M}{S_1*S_2} \approx 190Gb$. While servers may have more than 190GB of RAM, this is but one of many possible ad-hoc queries. A Big Data installation is likely to support multiple concurrent processes (ETL, maintenance, reporting), all competing for resources. It is therefore impossible to guarantee that each type of ad-hoc query will obtain enough RAM at the moment of execution. The plan described above is then no longer possible, forcing the query optimizer to spill the join operation onto disk. A *join spill* (the term may differ between vendors) means that some part of an execution plan requires too much RAM, and data have to be separated into N chunks[3], small enough to fit into available RAM, and that can be processed sequentially with their results assembled afterwards.

Join spill reduces maximum RAM utilization, but increases disk I/O, a slower resource. Fig.4 illustrates, why a condition reducing a table on one side of the join may not reduce the number of disk operations for the table on the other side of the join. Therefore, disk I/O operations can be estimated according to *optimistic* (concentrated keys) or *pessimistic* (spread out keys) scenarios. In the optimistic one, the number of operations is the same as in a RAM join, whereas in the pessimistic one the whole table may need to be scanned for each chunk.

- Disk page ops, optimistic: $\frac{R*A*(4*M+3)}{S_1*P}$
- Disk page ops, pessimistic, RAM join: $\frac{R*A*(4*M*S_1+3)}{S_1*P}$
- Disk page ops, pessimistic, join spill: $\frac{R*A*(4*N*M*S_1+3)}{S_1*P}$
- Logical ops: $\frac{R*M}{S_1} + \frac{R*M}{S_1*S_2} + \frac{R*M}{S_1*S_2*S_3}(log(\frac{R*M}{N*S_1*S_2*S_3}) + 1)$

For an **Anchor model** the hand-made execution plan aims to maximize merge join utilization. Hash join can be almost as fast as merge, but it requires a lot of RAM. If limited, the join must again be spilled during execution. The following execution plan can either be implemented by hints or by a set of subsequent temporary tables.

[3] $N = \langle$pessimistic RAM estimation$\rangle / \langle$available RAM$\rangle$, rounded up.

1. Read CookieId, CreationDate columns from disk to RAM, according to EM-pipeline strategy [8]. Data is filtered by the condition on CreationDate.
2. Load WebEventID from WebEvent–Cookie tie table via merge join.
3. Streaming sort, resegmentation and storing of WebEventId into temp table.
4. Filtered loading of EventType attribute table, joined with the temp table from step 3, reducing the count of WebEventID keys.
5. Load EventDate and CookieId from EventDate attribute table and triple merge join the WebEvent–Cookie tie and temp table from step 4 inside RAM.
6. Merge join of temp tables from step 1 and step 5, streaming filtering according to the condition CreationDate = EventDate.
7. Loading of User values from User attribute table inside RAM via merge join with temp table from step 6, calculating the number of unique User values.

The CPU in each server is expected to perform 10^9 FLOPS and the I/O able to push 150MB/s, both considered to be conservative estimates. Using these metrics the estimated execution time was determined together with the presented formulas for 3NF and Anchor modeling, with the results seen in Table 1.

- Disk page read ops, optimistic: $\frac{R*A*(2*S_2*S_3+6*S_2*S_3*M+6*S_3*M+2*S_3+2*M)}{S_1*S_2*S_3*P}$
- Disk page read ops, pessimistic: $\frac{R*A*(4*S_2+2*M*S_2+8*S_1*S_2*M+2*S_1*S_2+2*M)}{S_1*S_2*P}$
- Disk page write ops: $\frac{2*R*A*(1+M)}{S_1*P}$, RAM usage (max): $\frac{R*A*M}{S_1*S_2}$
- Logical ops: $\frac{R}{S_1}*log(\frac{R}{S_1})+\frac{M*R}{S_1}*log(\frac{M*R}{S_1})+\frac{M*R}{S_1*S_2}*log(\frac{M*R}{S_1*S_2})+\frac{M*R}{S_1*S_2*S_3}*log(\frac{M*R}{S_1*S_2*S_3})$

Table 1. Theoretical estimates of execution times and actual results from experiments.

	(optimistic–pessimistic), actual execution time		
Plan type	5 Nodes	10 Nodes	12 Nodes
3NF RAM Join	(671–2075), 3133s[4]	(335–1038), 662s	(280–865), 491s
3NF Spilled Join	(687–8399), 3017s	(344–4119), 1703s	(287–3423), 1172s
Anchor modeling	(1185–4515), 2643s	(849–2514), 1174s	(719–2074), 959s

Anchor modeling is not able to reach the speed of the RAM joined 3NF query, but it does comparatively well and is faster than the spilled join, which is the more likely of the two. Considering that no work have to put into creating dedicated indexes in Anchor modeling, whereas the problem grows exponentially with the number of columns in a 3NF table, it proves itself very suitable for ad-hoc querying. It also scales similarly to the others when the number of nodes is raised. In 3NF, where non-trivial join dependencies exist, data may also need to be fetched from other nodes. Tables less normalized than 6NF are therefore, with respect to the autonomy of the nodes, suboptimal for MPP.

[4] This join required > 38Gb of RAM on each node and it also spilled to disk.

3 Conclusions

While much have been said about Big Data with respect to information that have little structure, such as media in different forms, there has been little research into structured Big Data outside of NoSQL solutions. Systems that produce logs, sensors that give information, and other high transaction environments, such as banking, stock trading, and retail to name a few, all yield large volumes of structured data and should all benefit from the approach described in the paper. The experiments carried out showed that the approach works well in the simplified case of two linked entities. Real-world business cases sometimes require three, four or even a dozen linked entities in a single ad-hoc query. Such cases multiply the risk of join spill occurring in some step, and amplify its negative effects. When the number of joins, tables, and filtering conditions increase, the accuracy of the estimated RAM requirement decreases. The query optimizer may significantly overestimate RAM requirements and cause unnecessary join spills. Until rectified, forcing plans is necessary in order to achieve maximum performance. The given approach has been in use at Avito for over a year, for ad-hoc and regular reporting, and even for near-real time KPIs. It has demonstrated stability in terms of execution time and resource consumption, flexibility with respect to schema evolution, and low maintenance with respect to total cost of ownership.

References

1. Arindam Banerjee, Joydeep Ghosh: Clickstream Clustering using Weighted Longest Common Subsequences. Proceedings of the Web Mining Workshop at the 1st SIAM Conference on Data Mining
2. Chen, M., Mao, S., Liu, Y.: Big Data: A Survey, Mobile Networks and Applications Volume 19, Issue 2, pp 171-209, April (2014)
3. Date, C. E., Darwen, H., Lorentzos, N. A.: *Temporal Data and the Relational Model*, Elsevier Science (2003)
4. Hultgren, H.: Modeling the Agile Data Warehouse with Data Vault, Vol. 1, Brighton Hamilton (2012)
5. Kalavri, V., Vlassov, V.: MapReduce: Limitations, Optimizations and Open Issues. TrustCom/ISPA/IUCC, IEEE, pp. 1031-1038 (2013)
6. Lamb, A., Fuller, M., et al.: The vertica analytic database: C-store 7 years later. Proc. VLDB Endow. 5, 12, pp. 1790-1801, August (2012)
7. Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P., & Wohed, P.: Anchor modeling — Agile information modeling in evolving data environments, *Data & Knowledge Engineering*, Vol. 69, Issue 12, 1229-1253, Elsevier, Dec. (2010)
8. Shrinivas, L., Bodagala, S., et al.: Materialization strategies in the Vertica analytic database: Lessons learned. Christian S. Jensen; Christopher M. Jermaine & Xiaofang Zhou, ed., 'ICDE' , IEEE Computer Society, pp. 1196-1207 (2013)
9. Wang, G., Konolige, T., et al.: You Are How You Click: Clickstream Analysis for Sybil Detection. USENIX Security, pp. 241-256, August (2013)
10. Russias Avito Becomes Worlds 3rd Biggest Classifieds Site After $570M Deal With Naspers http://techcrunch.com/2013/03/11/russias-avito-becomes-worlds-3rd-biggest-classifieds-site-after-naspers-deal/