Computer Standards & Interfaces 54 (2017) 86–93

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Big Data normalization for massively parallel processing databases

Nikolay Golov[a,*], Lars Rönnbäck[b]

[a] National Research University Higher School of Economics, Moscow, Russia
[b] Department of Computer Science, Stockholm University, Stockholm, Sweden

ARTICLE INFO

Keywords:
Big Data
MPP
Database
Normalization
Analytics
Ad-hoc
Querying
Modeling
Performance
Data Lake

ABSTRACT

High performance querying and ad-hoc querying are commonly viewed as mutually exclusive goals in massively parallel processing databases. Furthermore, there is a contradiction between ease of extending the data model and ease of analysis. The modern 'Data Lake' approach, promises extreme ease of adding new data to a data model, however it is prone to eventually becoming a Data Swamp – unstructured, ungoverned, and out of control Data Lake where due to a lack of process, standards and governance, data is hard to find, hard to use and is consumed out of context. 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. This technique is almost as convenient for expanding data model as a Data Lake, while it is internally protected from transforming to Data Swamp. A case study of how this approach is used for a Data Warehouse at Avito over a three-year period, with estimates for and results of real data experiments carried out in HP Vertica, an MPP RDBMS, is also presented. This paper is an extension of theses from The 34th International Conference on Conceptual Modeling (ER 2015) (Golov and Rönnbäck 2015) [1], it is complemented with numerical results about key operating areas of highly normalized big data warehouse, collected over several (1–3) years of commercial operation. Also, the limitations, imposed by using a single MPP database cluster, are described, and cluster fragmentation approach is proposed.

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 the implementation of platforms fast enough to load, store and execute ad-hoc analytical queries over Big Data [2].

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). Until now, Hadoop has been considered a universal solution, because it can solve issues of *Volume* and *Velocity* by almost unlimited horizontal scaling, while the issue of *Variety* can be solved by storing data without schema (schema-less databases).

The Data Lake approach was proposed as an ideal method of building Big Data infrastructure: create a single store ("lake") of all available data, structured and unstructured. Store all available data, without spending time and resources on modeling or structuring. The term "single" means that all types of data from various sources can be combined and analyzed together to test various hypothesis. Hadoop offers a relatively cheap way of scaling such single store according to growing volumes and velocities (you can pay just for hardware, no additional licensing fees).

As the first implementations of Data Lake were launched, it turned out, that schema-less data storage is really a fake. Schema-less storage just replaces the need to structure data "on write" with the necessity to structure them "on read". So, not "schema-less" against "schema-on-write", but "schema-on-read" against "schema-on-write". In a short term perspective "schema-on-read" sounds promising. But if data over a long time period must be analyzed, than a risk of schema changes is possible. In addition, there is a risk of analyzing the same data differently by different analysts, because of different schemes on read. So, as time passes, Data Lake is prone to transform into "Data Swamp" – unstructured, ungoverned, and out of control Data Lake where due to a lack of process, standards and governance, data is hard to find, hard to use and is consumed out of context.

* Corresponding author.
E-mail addresses: ngolov@avito.ru (N. Golov), lars.ronnback@anchormodeling.com (L. Rönnbäck).

http://dx.doi.org/10.1016/j.csi.2017.01.009
Received 4 April 2016; Received in revised form 17 January 2017; Accepted 20 January 2017
Available online 26 January 2017
0920-5489/

Therefore, even Data Lake approach does not eliminate the need to model and structure incoming big data. It enables to put the process of modeling aside, for a short period of time.

Speaking of performance, basic technologies of Hadoop, such as HDFS, MapReduce, Pig and Hive, are cheap and scalable, but they have their own drawbacks, especially in their ability to process difficult queries (big join), to support fast ad-hoc analysis [3].

This paper introduces a new processing approach, using Anchor modeling in massively parallel processing (MPP) databases. The approach simplifies and automates tasks of data modeling and data structuring, so it helps to face the issue of Variety of Big Data, without putting it aside, as Data Lake approach does. The approach significantly increases the volume of data that can be analyzed within a given time frame. It has been implemented in HP Vertica [4], 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 [5] connecting millions of buyers and sellers on a daily basis. The approach gives its 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.

This paper is an extension of an initial paper, presented at The 34th International Conference on Conceptual Modeling (ER 2015) [1]. Initial paper, which was focused mostly on performance and optimization issues, solved in the first two years of adopting the approach. Extensions were added after one more year of developing the approach, after several attempts of adopting it for other companies. During this year, the Anchor modeling-based data warehouse of Avito were expanded with data from dozens of heterogeneous source systems and turned into some kind of structured Data Lake.The extension contains a checklist for adopting Anchor Modeling on some MPP database. It also describes limitations, imposed by using a single MPP database cluster to operate the ever-increasing set of tables.

2. The Avito case for normalized Big Data

Based in Moscow, Avito is Russia's fastest growing e-commerce site and portal, “Russia's Craiglist”. It is growing ≈50% a year and is now second only to Craiglist and Chinese site 58 in the rating of classified sites [5]. Its number of unique daily visitors in the middle of 2016 is 6 times higher, than in the middle of 2013. Avito's monetization is based on data and data analysis. Avito stores all types of data into its data warehouse. There are over 20 streams of data:

- Data from OLTP systems
  - CRM
  - Payment systems
  - Back-office system of a main site
  - Back-office system of side-sites (specialized vertical classifieds)
- Clickstream data
  - Clickstream (all clicks and actions of a site visitors, up to 1bln. records a day), desktop and API
  - Logs of DSP/SSP systems (online ads).
  - Logs of CTR prediction auctions (online ads).
  - Logs of email and sms sending systems (SendGrid).
- External data
  - Social media data
  - Open data sources (of government, of enterprises)
  - API data of partners

In terms of 3Vs, Avito clickstream data have over 1 billion user actions a day (Volume). User profiles, which help to reject non-humans and generate personalized content, have to be recalculated in near real-time, as a CTR prediction coefficients (Velocity). The business model of Avito is constantly evolving, where new features are constantly added, which affects both, OLTP and a click stream data (Variety). Regarding Variety, social media data are also a great example – their data model is constantly evolving without notifications.

There are dozens of data analysis cases, applicable for the data, described above. 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 [6]. 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 [7]. 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.

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 [4], and highly normalized data model, Anchor Modeling [8], were selected.

2.1. Anchor modeling

Anchor modeling [8] 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 [9], 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* [10]. 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. Theoretical pros and cons of highly normalized data model for Big Data

Here is a list of points for and against choosing a high level of normalization for Big Data. Points for are unconditional. Points against can be interpreted as open issues, which must be solved to make highly normalized big data real. Those issues can be solved within chosen database management system, ETL engine or BI tool. Otherwise, hand-made applications must be implemented.

1. **(+) Pros. Ease of expansion.** An important effect of Anchor modeling is the ease with 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 [8], and can be dealt with to incorporate new features when time permits. This classical benefit of anchor modeling and a highly normalized data model is important for supporting an increasing variety of data (3-rd V).
2. **(+) Pros. Data compression.** When data is sparse, arising though the addition of new attributes or when an attribute does not apply to all instances, normalization provides another 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.
3. **(+) Pros. No update.** Database Management systems, suitable for big data, such as hadoop or column based MPP databases, are unsuitable for update/delete operations (it is the price for scalability). Historized attributes and ties of Anchor modeling use only one date (From Date) for slow-changing dimension type 2 historicity, instead of pair of dates (From Date+To Date). If pair of dates is used, than closing date (To Date) have to be updated when a new value of attribute/tie arrives. Single date approach requires only inserts, which are better suited for big data environment.
4. **(−) Cons. Time series merge joins required.** This con is caused by same reason, as previous pros - single date historicity. Assume, that name of a user is stored inside a historized attribute table. It has columns UserId, Name, ActualDate. The process of loading new data can be partially represented as merging of tables T and T_{new} with same columns. Row from T_{new} can be inserted into T only if there is no same UserID value in T (first attribute value for a new user), or Name value from T, which has same UserID as row from T_{new} and a maximum ActualDate, lesser than ActualDate from T_{new} (see example on Fig. 2).

   The algorithm, described above, can be implemented as a nested query, or a nested-loop query execution plan with a quadratic complexity.

```
insert into T
select T_new.* from T_new, T
where T.UserID =T_new.UserId and T.Value<>T_new.Value
  and T.ActualDate = (
            select max(ActualDate) from T where UserID = T_new.UserID
            and ActualDate < T_new.ActualDate
            )
```

   If T contains billion and T_{new} contains millions of rows, such operation can be very time consuming. In contrast, time series merge join can have linear complexity for this task. Assume that rows of T and T_{new} are sorted by UserID (first) and ActualDate (second). Than both tables can be loaded, row by row, and stitched together using two simple conditions: $T.UserId = T_{new}.UserId$, $T.ActualDate < T_{new}.ActualDate$. While conditions are true, algorithm must take next rows from T, storing last value of $T.Name$ as $Name_{last}$. If conditions are wrong, value $Name_{last}$ has to be compared with $T_{new}.Name$. If values are not equal, row from T_{new} has to be inserted into T, otherwise row from T_{new} has to be ignored. After comparison next row from T_{new} has to be taken. Given algorithm illustrates possibility of linear complexity for time series merge join, as well as parallelization (parts of T and T_{new} for different users can be merged in parallel). Therefore, efficient use of anchor modeling for big data is possible, if given (or more efficient) algorithm is supported by chosen DBMS or implemented manually. HP Vertica supports efficient time series join as part of SQL syntax.
5. **(+) Pros. Uniform rules for data distribution.** Big data environments are usually based on utilization of multiple processing units (servers), therefore data model have to be somehow distributed across servers. The section about Anchor Modeling contains uniform rules of data distribution, that significantly ease expanding of data model.
6. **(−) Cons. Multiple projection support required.** If two tables are identically distributed across servers, they can be joined locally (if rows from both tables with identical join keys are always located on the same server). Otherwise, data redistribution is required, which is resource-intensive task for big data environment (for terabytes and petabytes). Tie is a perfect example of a table, which can require two different types of data distribution. It contains two keys, and can be joined with another table using either first one, or second one. If Tie table is distributed across servers according to first key, then first key join will be processed locally, while second key join will be processed via redistribution, and vice versa. Optimal performance for both joins can be achieved, if chosen DBMS supports storing of multiple copies of table data with different distribution. In case of Tie T(UserId, CookieId), it can be stored as two tables T_{userid} and $T_{cookieid}$, with identical columns and rows, but different distribution. Therefore, if join on UserId is required, table T_{userid} have to be used, otherwise - $T_{cookieid}$. Such duplicate

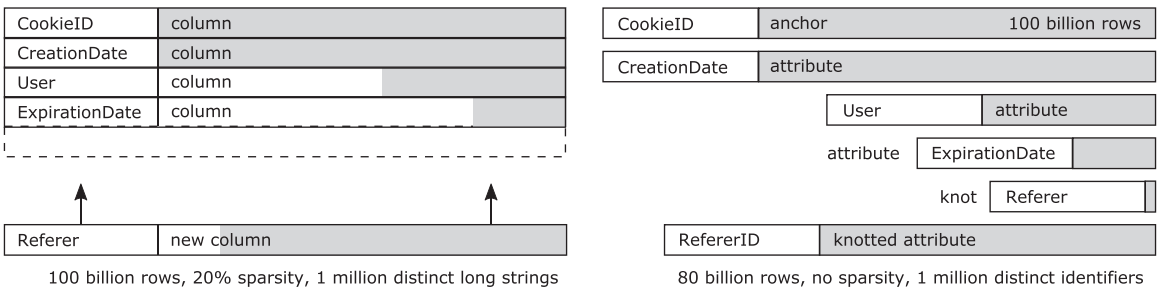

Fig. 1. Expanding a less normalized model (left side) and a corresponding Anchor model (right side).

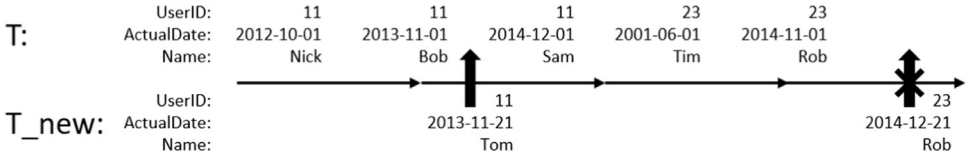

Fig. 2. Merging of new values into highly normalized historized attribute table. First value from T_{new} has to be added, while second one - not, because last preceding record from T for same user stores same value.

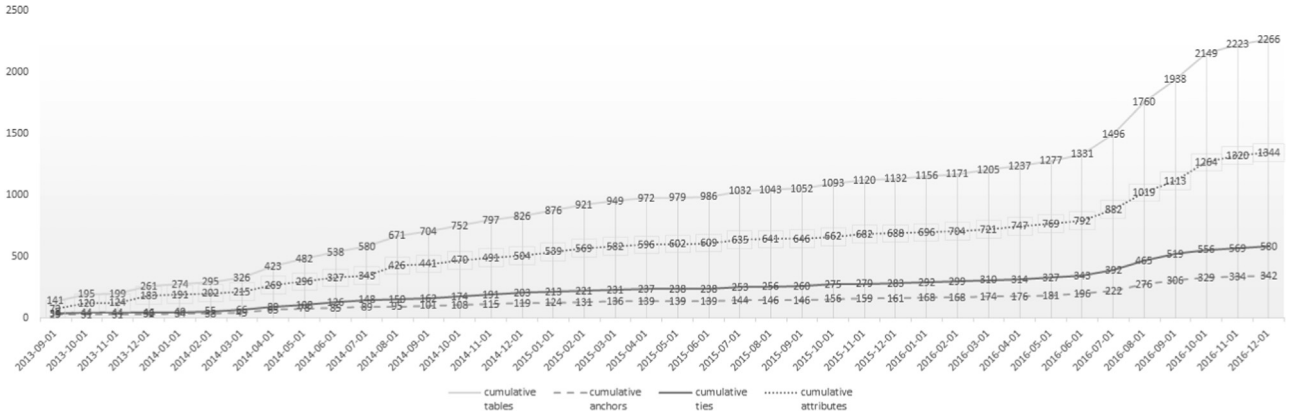

Fig. 3. Cumulative numbers of Anchors, Ties and Attributes in Avito data warehouse over three years.

tables are called “projections”, and they are critical for efficient analysis of big data in an Anchor model.

7. **(+) Pros. Big number of comparable normalized tables are easier to maintain, than few enormous denormalized tables** The usage of multiple narrow highly normalized tables gives additional benefits in terms of maintenance. If table is big (tens-hundreds of bln. of rows, $>1Tb$), its repartitioning, resegmentation (change of logic of data distribution among servers of clusters) or adding indexes will require a lot of time and system resources, slowing processes, related to any given table. While listed maintenance operations are in process, affected tables significantly decrease performance of related processes. Multiple highly normalized tables can be processed one-by-one, in hours of low system load, while maintenance of big denormalized table can take days of partial inoperability of the data warehouse.
8. **(–) Cons. Efficient merge joins required.** High normalization assumes usage of many tables, which have to be joined together for analysis. Big data tasks assumes that the total size of joined tables can be enormous, too big for available RAM, therefore making hash-join approaches inefficient.
9. **(–) Cons. Efficient query execution plans for ad-hoc analytical queries.** Even efficient merge join implementation inside a DBMS can not guarantee optimal query execution plans for ad-hoc queries, which, in case of anchor modeling methodology, can take data from dozens of tables, joined together. Given issue is the most complex one, it is observed in a separate section below.

2.3. The evolution of the Avito Data Warehouse, numerical results

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 14 nodes in order to scale up performance.

Clickstream data are loaded every 15 min. At the beginning of 2014 each such batch contained 5 million rows (≈1.5 GB) and 15 million (≈5 Gb) one year later. Avito has evolved its data model over the years. The clickstream records originally had less than 30 attributes and now contains 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. 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. All anchors are loaded in parallel in 5 min, all ties and attributes are loaded in parallel in 11 min.
- 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. All anchors are loaded in parallel in 4 min, all ties and attributes are loaded in parallel in 6 min.

Three years of Avito Data Warehouse evolution provided us with numerical proofs of Anchor Modeling benefits:

1. **Ease of expanding.** Data model can evolve and grow by adding new Anchors, Ties and Attributes. Fig Fig 3 demonstrates constant growth of data model over three years.
2. **Data compression.** Click stream data of Avito demonstrated best benefits of data compression. Rather than repeating the relatively few long strings representing referers (URLs) for each row of click stream, these are stored as unique values of a knot. The knotted attribute instead contains identifier references, much smaller than the strings they represent. Big data tasks often require complex analysis of semi-structured data, such as strings (URL, UserAgent, Cookie attributes), via regular expressions. Usage of knotted attributes gives analysts possibility to analyze only unique values (tens of millions values) instead

Table 1
Data compression rates for click stream data.

	71 billion click stream events, key size=8B		
Attribute	Unique values	Value size	Compression
URL address	13 bln.	170 B	0.28
Referer	2.6 bln.	212 B	0.11
User Agent	19 mln.	109 B	0.15

of analyzing all rows of click stream (tens-hundreds of billions). 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. Compression rate can be estimated as $\frac{(2 * N_{rows} * KeySize + N_{uniqValues} * ValueSize)}{N_{rows} * ValueSize}$. Table 1 contains compression rates for some real types of click stream data.

3. **No update.** 1108 of 1368 attributes and 267 of 581 ties (see Fig. 3) are historized and loaded at least once per day. Total number of rows, inserted inside historized tables daily, varies from 50 millions to 300 millions.
4. **Efficient time series merge joins required.** All attribute and tie tables of a model are filled using efficient time series merge join algorithm of HP Vertica. Candidate row for historized tie/attribute is loaded only if value is new, or changed in comparison with previous one. Candidate row for unhistorized tie/ attribute is loaded only if attribute/tie value is new. Otherwise candidate row is ignored Fig. 4 shows, that ETL processes of Avito load up to 50 billion candidate rows a day, inserting up to 20 billion of new rows after time series merge join checks. Up to 50% of candidate rows are discarded. Efficiency of time series merge join checks gives etl engineers an ability to easily reload batches of data - previously loaded values will be automatically ignored, while new values will be inserted.
5. **Uniform rules of data distribution** guarantees, that the process of adding new entities inside a data model is swift. This process in Avito is almost fully automated and supported by a single engineer. Fig. 3 shows, that up to 264 new tables (and correspondent ETL processes) can be added within a month, without affecting performance of a cluster or ETL processes of previously added tables. Process of adding a new tables must be not only swift, but it also must produce a scalable load. Clustered database [4] is theoretically able to scale load by adding more servers into cluster, but it works only if data tables are properly designed, with proper algorithm of distribution between servers of the cluster. Adding few inefficiently distributed/undistributed big data tables can produce unscalable load. Fig. 5 demonstrates, that tables, created to anchor modeling methodology, with uniform rules of data distribution, produce scalable etl load.
6. **Multiple projection support** helps Avito to facilitate results from a preceding paragraph, in cases of adding a new *ties*.
7. **Ease of maintenance of big number of normalized tables.** Anchor modeling guarantees, that schema of data table will never change. Attribute and tie will always contain 2 business fields (3 for historized ones), anchor - 1 business field. Therefore, any part of data model can be unloaded inside convenient storage formats (CSV, ORC, Apache Parquet) and loaded back. Risk of schema change, which will prohibit/complicate loading of old data, is minimal. Avito uses this feature of Anchor Modeling to implement custom archive/backup tools, to transfer data between multiple Vertica/Hadoop/ClickHouse cluster. Old data, unloaded from Vertica cluster in 2013 for licensing reasons, were successfully loaded back in 2016 without any issues.
8. **Efficient merge joins – essential for data marts.** Although anchor modeling is extremely efficient for data warehousing tasks, it is not so convenient for reporting tasks, for OLAP tools. Avito team implemented data mart layer of denormalized tables for analysts and reporting tools. The main idea is based on Bill Inmon approach [11]: all detailed data have to be stored inside highly normalized data model (Anchor model - equivalent of "integrated repository of atomic data" of Inmon), while data aggregates, augmented with some business logic, can be materialized as a denormalized data marts (dimensional model). Ideally, all data marts must support full purging and repopulating from detailed data of Anchor modeling. Data model of Anchor modeling frightens analysts, because it requires a lot of joins of historized tables. Denormalization of anchor model parts inside a single data marts hides those complexities and simplify further analysis. In Avito, process of expanding data model (Anchor modeling part) is constantly supported by expanding set of data marts. New data marts are mostly created by business users and analysts. Avito's ratio of detailed data size/data mart data size is approximately 9 to 1 (not all data needed regularly, most data marts show only latest value of all attributes (thereby, compressing historized tables), some data marts show only aggregated figures, not detailed ones). Fig. 6 shows evolution of data mart area of Avito. Number of data marts doubled in one year. Number of rows, produced in one daily data marts recalculation, has tripled in one year. Regardless of growth, daily process of data mart recalculation takes up to 4 hours and is performed at night time.
9. **Efficient query execution plans for ad-hoc analytical queries** Previous article about Avito and Anchor Modeling [1] contained brief description of efficient semi-automatic query execution plans, used to significantly improve performance of ad-hoc analytical queries, ran over highly normalized data model. High level of normalization leads to big number of joins in such queries. Query plan optimization is mostly based on intellectual preliminary filtering of data, on avoiding of reading from disc and loading into main memory those rows, which will be eventually rejected because of join conditions. Creators of HP Vertica implemented first iterations of this approach (called SIP filters – Side Information Passing filters) [12]. Nowadays it has significantly evolved, reducing necessity to manually optimize plan of a new query to once or twice per month

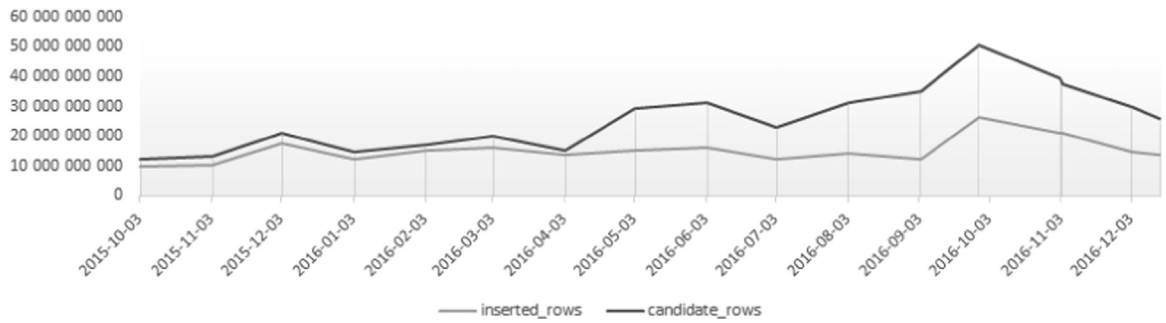

Fig. 4. Dynamics of daily etl – chart of loaded (candidate rows), and chart of inserted rows. Significant part (up to 50 percent of rows) are not loaded because of historicity.

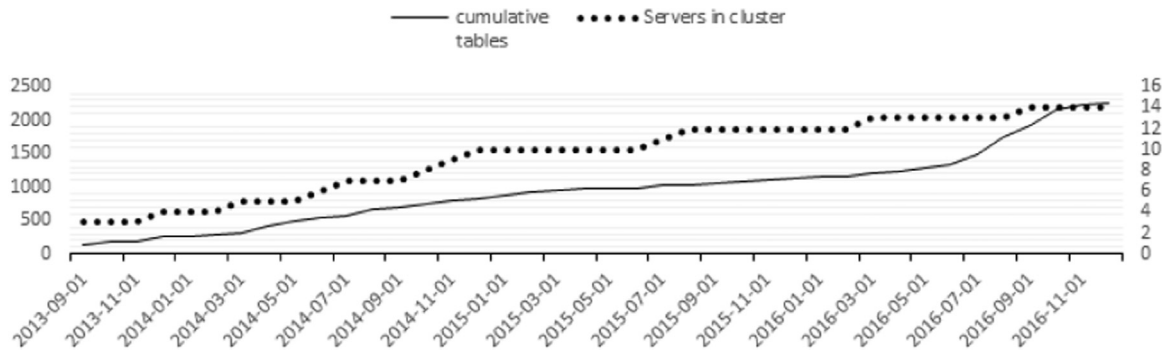

Fig. 5. Chart demonstrates that uniform rules of data distribution produce scalable load, therefore Avito can supports adding new data tables (and correspondent ETL procedures) by adding new nodes into cluster.

(see. Fig. 6 with numbers of monthly added data marts). Most complex data marts require complex join of up to 20 tables of few billions of rows, which produce ≈850 mln. rows and take up to 50 min to execute.

2.4. Open issues – cluster fragmentation

Modern big data platforms, such as HP Vertica or Hadoop databases, are able to somehow scale, however, this scalability is not unbounded, because they still have some "singe point of failure", unscalable part of functionality. Usually this unscalable part includes metadata storage and global data catalog. Although HP Vertica can process hundreds of queries in parallel, using multiple nodes of a cluster, each DDL query must get an exclusive lock on a single global data catalog, once or multiple times. Global catalog can reside on dedicated nodes (case of Hadoop), or can be reproduced on all the nodes (case of Vertica), it must be accessed only sequentially to avoid data corruption.

Fig. 3 shows number of data tables, processed by the system. Each table of Anchor data model require dedicated ETL process. All those processes can be launched in parallel (one parallel batch for all Anchors, second – for all Attributes and Ties). Current version of HP Vertica can process hundreds of parallel ETL processes (SQL insert-select queries), up to ≈500. Further growth of parallelism is blocked by competition for unscalable resources (global data catalog is best, but not only example). Moreover, it is impractical to use all resources of database cluster for ETL tasks – some part of cluster resources must be available for analytic tasks, reporting and ad-hoc queries. Highly normalized approach prohibits to extend data model by adding columns to existing tables. Model can be extended only by adding new tables, which in turn require adding new ETL processes, which will increase competition for unscalable resources.

Competition for unscalable resources of database cluster can be solved by increasing performance of this resources (task for vendor), or by cluster fragmentation: the division of single cluster into several pieces, with separate instance of unscalable resource for each piece. Cluster fragmentation can help with ETL processes, but it can spoil data integrity (data warehouse must be holistic, this feature is critical for business users [11]). Avito experiments showed, that Anchor model can be safely separated into few pieces for scalability of ETL processes, at the same time preserving integrity of all data for analytic queries.

Anchor data model looks like a graph, where Anchors are connected by Ties and surrounded by Attributes. Each Anchor, Attribute or Tie can be loaded from one or multiple data sources. Theoretically, there can be isolated parts of the graph, but the main benefit of data warehouse is based on its integrity – existence of connections between parts of data model, loaded from various sources. To enable ETL cluster fragmentation one must separate graph into sub-graphs, independently load tables data to tables of sub-graphs and solve collisions over tables, which form connections between sub-graphs. Here is a list of cases about separation of a single data-graph into two sub-graphs to enable independent loading (resulting sub-graphs can be further separated until required scale is reached). It is also important to create sub-graphs, which correspond to some business areas/source systems, making it possible to load separate business areas independently, choosing load rate according to business need of each area.

1. Simplest case – data graph can be separated into two sub-graphs without any paths (Ties) between. Two separate clusters can be used without any additional efforts, without competing for unscalable resources between them. See Fig. 7. Both cluster load all Anchors in parallel, then all Ties and Attributes in parallel, using local Anchors. This case perfectly fits to data model, being loaded from two completely unconnected data sources.
2. Two sub-graphs with common set of Anchors. Can be loaded using two clusters, if each common Anchor is stored inside separate fast

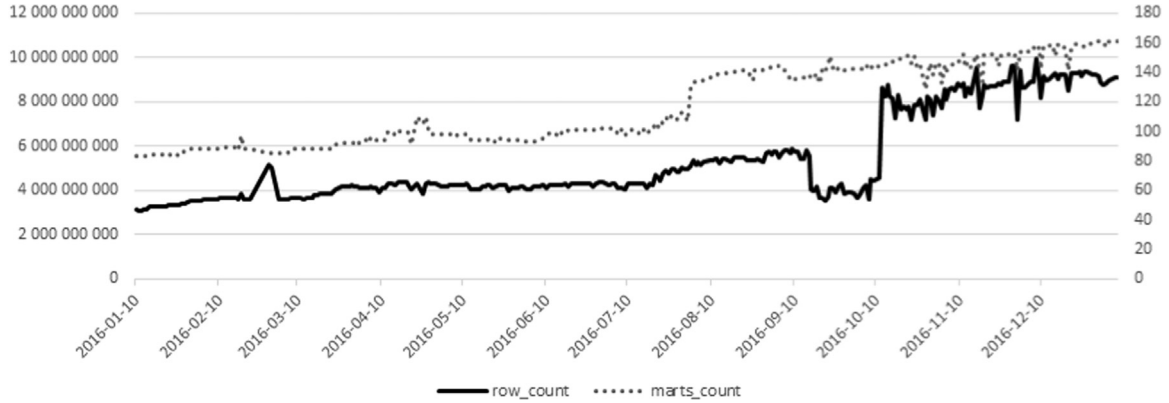

Fig. 6. Evolution of data marts of Avito data warehouse. From 79 data marts and 3 bln. rows of daily increment at the end of 2015 to 161 data marts and 9 bln. rows of daily increment at the end of 2016.

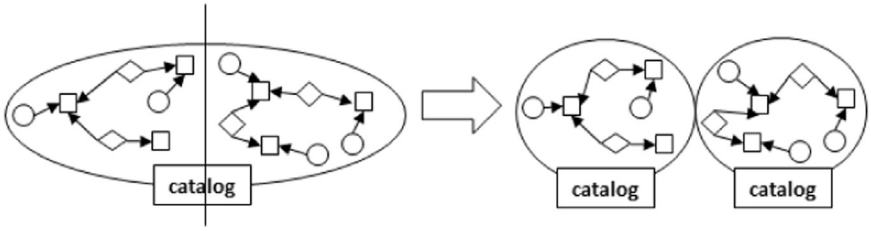

Fig. 7. Fully separable graph. Easy to load using two clusters. Anchors - squares, Ties - diamonds, Attribute - circles.

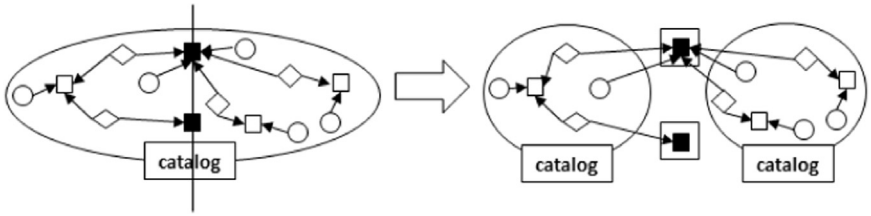

Fig. 8. Graph, separable into two sub-graphs with a common set of Anchors (filled black squares).

storage, external to both clusters, see Fig. 8. This fast store can be implemented using in-memory key-value databases, such as Redis. Both clusters can load all Anchors in parallel (local Anchors – to tables, common Anchors – to fast storage, to avoid loading duplicates). Afterwards, both clusters load all Ties and Attributes in parallel, using local and global Anchors. For example, such a graph can correspond to a data model with two source systems – clickstream and operational database. Two common Anchors are Users and Payments, while each system provides local information about those Anchors: detailed attributes of users and payments from operational database and clicks of users from clickstream.

3. Most complex case – two sub-graphs with common set of Anchors, Attributes and/or Ties. Can this case be completely avoided? It is technically possible to move all common Attributes and Ties into one sub-graph and reduce the problem to the previous. This solution can contradict with requirement to have correspondence between sub-graphs and business areas. As with the example from previous section, we can assume that Login (Attribute) of an User (Anchor) and Link between User and Payment can be loaded from both source-systems, clickstream and operational database. Login and Link between User and Payment can be moved to a single sub-graph (for example to sub-graph, correspondent to operational database), therefore those data from clickstream must be ignored. Also it is technically possible to duplicate Attributes and Ties, to create, for example, Attribute *OperationalDatabaseLogin*, local to operational database, and *ClickStreamLogin*, local to clickstream. This solution can cause misunderstanding for business users and analysts. Both solutions have drawbacks. There is a third solution – create a copy of common Attributes and Ties inside each sub-graph, and load them independently. See Fig. 9. This solution looks dangerous for integrity, but further section contains simple limitations, which can make the solution safe.

List of cases above describe all possible issues with a data model graph. Case three, in a worst possible case, can contain all objects of a graph. Experiments of Avito showed, that case 3 is rather rare. In most cases sub-graphs, corresponding to business areas/source systems, can be separated according to case 1 or 2. Nevertheless, all three cases assume data graph segmentation, although ad-hoc analysis requires combining sub-graphs back into single graph.

Sub-graph combination from separate loading clusters is possible because of “only-insert” nature of loading data into Anchor model. All new data of Anchors, Attributes and Ties are added as new data pages, without affecting old pages. Direct loading of data pages is significantly faster, than performing complete ETL processes, than insert-select queries. Same process (direct loading of data pages) is used for backup/restore operations. Fig. 10 shows, that “local” tables (B, C) get new data pages from a single source, therefore, no risk for data integrity. Moreover, this is true not only for tables from case 1, not only for Attributes and Ties from case 2, but also for common Anchors from case 2 and 3, because each such Anchor is really loaded from a single source – correspondent fast in-memory key-value store.

Therefore, multi-source increment is possible only for common Attributes and Ties from case 3. According to case 3, table with initial data must be copied to both sub-graphs - L and R. Afterwards, both copies have to process loading of new data and be combined back for ad-hoc analysis. Lets analyze case of Attribute, Login of a User - (u, l, d), where u - id of a User, l - login of a user, and d - start datetime of a login of a user (single date historicity, can be NULL for unhistorized attribute). At the initial moment, Attribute table contains row u_0, l_0, d_0). If new value of an attribute for user u_0 arrive to a single sub-graphs - L or R, than situation for u_0 is identical to cases 1 or 2 - single source of new data. Data integrity risk exists only if new row arrives to each sub-graph - u_0, l_L, d_L) and u_0, l_R, d_R). Here is list of possible cases:

- $l_L = l_0$, $d_L = d_0$ or $l_R = l_0$, $d_R = d_0$ - row u_0, l_L, d_L) or u_0, l_R, d_R) will not be loaded, so this case is identical to arriving of new value to single sub-graph, no possible conflict.
- $l_L <> l_0$, $d_L > d_0$, $l_R <> l_0$, $d_R > d_0$ - both rows, u_0, l_L, d_L) and u_0, l_R, d_R) will be loaded to sub-graphs and, finally, to a single graph. This case can lead to minor issue - it is possible that l_L=l_R, therefore

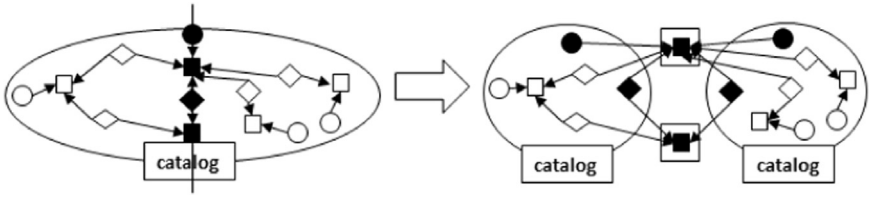

Fig. 9. Unseparable graphs, with significant common subgraph (filled black objects).

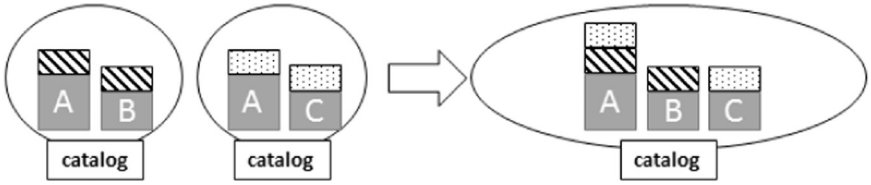

Fig. 10. Recombining new data from sub-graphs into single data graph by moving only new data pages.

two consequent rows for single attribute of a user will be loaded. This case can confuse inexperienced analyst, but it did not spoil data integrity.

- $l_L = l_0$, $d_L > d_0$, $l_R <> l_0$, $d_R > d_0$, $d_L > d_R$ - this case is the most dangerous one. Trivial algorithm of loading historized attributes will discard row (u_0, l_L, d_L), only row (u_0, l_R, d_R) will be loaded, do the history of user u_0 will contain only those consequent logins - l_0, l_R, although correct history must contain those logins: l_0, l_R, l_L, or l_0, l_R, l_0 (login changed once, then changed back to initial value). Therefore, algorithm of loading historized attributes has to be modified to load row $u_0, l_L, d_L)$ in case of $l_L = l_0$, $d_L > d_0$. This algorithm can load sequential duplicate values of an attribute.

3. Conclusions

Three years of using highly normalized data lake-like data warehouse in Avito proved the viability of this approach. Constant extension of Avito data warehouse demonstrated, what tasks and volumes can be processed by such a combination of data platform and data modeling approach. However, this extension showed possible limitations, imposed by using a single MPP cluster (unscalable resources, competition for global data catalog). Cluster fragmentation techniques can theoretically overcome those limitations, which will be tested in further researches.

References

[1] N. Golov, L. Rönnbäck, Big data normalization for massively parallel processing databases, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag, vol. 9382, 2015, pp. 154–163.

[2] M. Chen, S. Mao, Y. Liu, Big data: a surey, Mob. Netw. Appli. 19 (2) (2014) 171–209. http://dx.doi.org/10.1007/s11036-013-0489-0.

[3] V. Kalavri, V. Vlassov, MapReduce: Limitations, Optimizations and Open Issues, in: 2013 Proceedings of the 12th IEEE International Conference on Trust Security and Privacy in Computing and Communications (TrustCom), 2013, pp. 1031–1038. ⟨http://dx.doi.org/10.1109/TrustCom.2013.126⟩.

[4] A. Lamb M. Fuller R. Varadarajan N. Tran B. Vandier L. Doshi C. Bear. The vertica analytic database C-store 7 years later, in: Proceedings of the VLDB Endowment, vol. 5 (12), 2012, pp. 1790–1801 ⟨http://dx.doi.org/10.14778/2367502.2367518⟩, URL http://arxiv.org/abs/1208.4173.

[5] Russia's Avito Becomes World's 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/⟩, 2013.

[6] A. Banerjee, J. Ghosh, Clickstream clustering using weighted longest common subsequences, in: Proceedings of the Workshop on Web Mining SIAM Conference on Data Mining, 2001, pp. 33–40. ⟨http://citeseerx.ist.psu.edu/viewdoc/download?Doi=10.1.1.135.9092 & rep=rep1 & type=pdf⟩.

[7] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, B. Y. Zhao, You Are How You Click: Clickstream analysis for sybil detection, in: Proceedings of the 22nd USENIX Conference on Security, SEC'13, USENIX Association, Berkeley, CA, USA, 2013, pp. 241–256. ⟨http://dl.acm.org/citation.cfm?Id=2534766.2534788⟩.

[8] L. Rönnbäck, O. Regardt, M. Bergholtz, P. Johannesson, P. Wohed, Anchor modeling – agile information modeling in evolving data environments, Data Knowl. Eng. 69 (12) (2010) 1229–1253. http://dx.doi.org/10.1016/j.datak.2010.10.002 (http://dx.doi.org/10.1016/j.datak.2010.10.002).

[9] H. Darwen, A.C.J. Date, H. Darwen, N.A. Lorentzos, Temporal Data and The Relational Model The Book s Aims Contents (Part III) Part I : Preliminaries CS319 : Theory of Databases Example : Current State Only CS319 : Theory of Databases, 2013, pp. 1–17.

[10] Hans Hultgren, Modeling the Agile Data Warehouse with Data Vault 1, Brighton Hamilton, 2012 (November 16, 2012)https://books.google.ru/books/about/Modeling_the_Agile_Data_Warehouse_with_D.html?d=4tTZmAEACAAJ & redir_esc=y, https://www.amazon.com/Modeling-Agile-Data-Warehouse-Vault/dp/061572308X.

[11] W.H. Inmon, Building the Data Warehouse, 3rd ed., John Wiley & amp; Sons, Inc., New York, NY, USA, 2002.

[12] L. Shrinivas, S. Bodagala, R. Varadarajan, A. Cary, V. Bharathan, C. Bear, Materialization strategies in the Vertica analytic database: Lessons learned, in: 2013 IEEE Proceedings of the 29th International Conference on Data Engineering (ICDE), 2013, pp. 1196–1207. ⟨http://dx.doi.org/10.1109/ICDE.2013.6544909⟩.