UDC 004.8

B.F. Trofimov, A.V. Arsiriy, E.A. Arsiriy

MODEL TO REPRESENT LARGE GRAPHS OF USER PROFILES IN HBASE ON DSP SIDE

Annotation: Success of Real-Time Bidding and advertising campaigns depends on building precise target audience profiles on Demand Side Platoform side. Modern Demand Side Platoform's user databases contain billions of profiles with multiple connections between each other. The challenge is to remain database efficient on usual read/write operations and allow to query information about connections in a graph manner. This article addresses a data model to represent a large profile databases with subset of graph-based operations and corresponding mapping to modern NoSQL databases like HBASE.

Keywords: databases, hbase, graph, real-time bidding, demand side platform, profile bridging.

Introduction. Today's strategic vision of Advertising (Ad) business and related technologies tightly depends on real-time bidding (RTB) and programmatic Ad technology. The growth of RTB market showed 36% [1] last year. According to PRNewswire, RTB market will hit $42 Billion by 2018 [2].

The Ad market is highly-competitive, so outstanding Ad players are trying to convince advertisers like Nike, Coca-Cola and others that game rules are fair, metrics and budget spending is clear.

One of the biggest problems, Ad players have faced with, is profile fragmentation [3]. Nowadays almost everyone has multiple Internet entry points like tablets, phones, home PC and workstations (Fig. 1). Every such device has own unique traceable identifier (ID). In some cases like web serving that ID might be ordinary web cookie, for mobile devices it is device ID. From Demand-Side Platform (DSP) it turns out that database has multiple profiles assigned to different IDs however connected to the same user in fact. The problem here is that it prevents from building efficient AD campaigns. For instance, two user profiles are given with own ID (assuming that the user accessed WEB via home and work PCs). The first profile provides particular user interest (segment), that he is a male. The second profile provides information that the user is a higher educational person. Splitting information about gender and education between two different profiles prevents from involving this user in complex campaigns like delivering Ad to all males with higher education, just because from DSP perspective there profiles are two different persons.

The process of identifying profiles, connected to the same user, is called as a profile bridging. From mathematical perspective profile database is a huge graph $< V, E >$ where vertexes V are user profiles and edges E are bridging rules. Once bridging rules are well-defined then the task might be reduced easily to well-known problem of connected components identification.

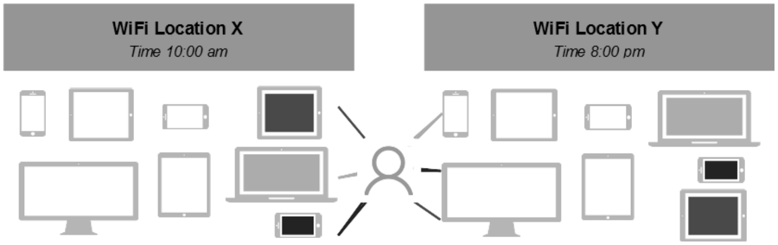

Figure 1 – Profile fragmentation problem

The challenge is to keep profile database up to date and consistent in terms of high concurrency. Another challenge is an efficient database schema to store profiles and connections between them an a way to keep major DSP operations fast and cheap.

Graph databases. There are many specific databases to address graph problems and store data in a graph manner, like neo4j [4], titan [5], s2graph [6]. The major problem with them is that they compel specific proprietary data schema and storage engine. In many cases this is unacceptable. For instance, a lot of companies already have databases with well-defined data schema, so migration to new schema breaks existent API and requires data transformation, that is quite painful.

Another example is a scale. Just a few graph databases (e.g. Titan) are able to process graphs with billions of edges and vertexes with small response time (low latency).

For many cases Titan over HBASE is a nice choice. It is distributed graph database focused on high scalability and distributed processing. In addition, it provides modern graph API based on Blueprints interface [7] and user-friendly query language Gremlin [8].

The down side of Titan is the follows:

- Titan compels own HBASE schema and the obfuscated data representation [9]
- It requires exclusive access to HBASE rows and columns.
- Source code of Titan provides obfuscated and complicated support of multiple HBASE versions based on shims [10] which prevents from seamless integration with distributed computation frameworks like Scalding [11].

For companies that already have large HBASE profile databases with well-developed infrastructure based on Scalding/Spark integration, these issues might be critical.

Requirements and constraints. This article addresses the issues of schema simplicity at scale. From DSP's perspective HBASE data schema and underlying data model should conform to the following requirements (*):

- an efficient access to user profile (corresponding segment list) by any of connected identifiers;
- an efficient check whether two profiles are connected, comparable by time to HBASE row lookup;
- an efficient retrieval of all linked profiles for a specific profile;
- simple and extensible data schema,
- Respecting super-node issue [12].
- In addition to requirements, the schema should address the following constraints:
- minimize amount of read operations to HBASE;
- no need to implement complete operations over graph database;
- simple integration with third-party frameworks like Scalding or Spark.

Structural data model. Given: finite set of user profiles $D = \{P\}$, each profile is just a triple $P =< ID, SEGMENTS, ASSOCIATIONS >$ where ID is unique identifier of profile, $SEGMENTS \subset \Re$ – finite list of linked segments (every segment is just unique integer) and $ASSOCIATIONS = \{ID\}$ – finite set of IDs, e.g. associations with other profiles.

From implementation perspective the model should reflect some features from NoSQL world:

$$PROFILES \cup EDGES \tag{1}$$

where

$$PROFILES = \{< ID, \underbrace{< TYPE, MASTERID >}_{\text{system family}}, \underbrace{\{SEGMENT\}}_{\text{segments family}}, \underbrace{\{ID\}}_{\text{associations family}} >\}$$

ID – is unique row identifier,
TYPE = 'profile' – type of row, indicates that the row belongs to *PROFILES* subset,

$MASTERID$ – reference to the master profile's ID, NULL value indicates that the profile is a master,

$SEGMENT$ as a part of Segment family – integer, identifying particular user's interest,

Association family – defines a list of connected profiles; each connected profile should provide complete list of connected IDs;

$$EDGES = \{< ID, \underbrace{TYPE}_{\text{systemfamily}}, \underbrace{WEIGHT}_{\text{properties family}} \} >,$$

ID – is an edge identifier, its value is *alphabetical* concatenation of IDs from two connected profiles, one of these two profiles needs to be a master, *TYPE='edge'* – type of row, indicates that the row belongs to *EDGES* subset,

$WEIGHT$ – probability of connection between profiles to cover case with fuzzy connections.

The master profile should be the oldest profile between connected profiles in the model. The reason is to reduce number of edges by using star-like graph topology instead of full graph. The down side of this constraint is that the model does not allow efficient one-read queries to check if two non-master profiles are connected.

Operations over data model. The defined above data model reflects just static data structure while real life assumes interaction with the model and its continuous changing. Let's define a set of operations over the data model. The operations from the first set are directly derived from requirements (*):

- *Add profile (P)*. Given: user profile P. The operation just adds one more row to existent data model M in a way like
$$\begin{aligned} M \cup \{< \ & P.ID,'profile', NULL, \\ & P.SEGMENTS, P.ASSOCIATIONS >\} \end{aligned}$$
- Any relationship and mastership should be defined during off-line operation "*Find Connected Components*".
- *Check Connection* $(ID1, ID2)$. Given: two profile *IDs*, on of them must be a master. Assuming that $ID1 < ID2$ by alphabetical order, then the model (1) should provide lookup of the united key $ID1 + ID2$.
- *Retrieve list of segments* (ID). Given: profile's ID. The model (1) should just retrieve partial row (Segments family) by ID.
- *Retrieve complete list of unique profiles*. The model (1) should retrieve all rows which column *PARENTID is NULL and TYPE='profile'*.

In addition to real-time operations there are background ones (**), that are focused on model evolution and eventual consistency:

- *Find Connected components*. Implementing algorithm of identifying connected components. The only one requirement is that it must distinguish master profiles and specify corresponding $MASTERID$ column for dependent profiles.
- *Profile Compaction*. Operation synchronizes list of segments between all connected profiles in a two steps:
  (a) assemble complete list of segments based on all connected profiles;
  (b) clone the list between all connected profiles.

Eventual consistency means that the profile compaction might take some time, for that moment the read operations might return inconsistent results (incomplete segment list). However once compaction is finished, then the model becomes consistent (lists of segments for all connected profiles are the same).

In order to make this operation more efficient, the model (1) might be slightly modified with supporting one more column family $UNSYNCED\_SEGMENTS$ – list of recently added segments that have not been synchronized yet. Once synchronized, the segments are removed from this family.

Considering super-node problem, the fact is that indeed *Associations* family might consist of millions connected profiles in general case. Retrieving all of them affects performance and does not make sense for real-time operations. That's why operation *Check Connection* is designed to not to use this family at all.

HBASE mapping. By design the data model (1) is tuple-based, thus its implementation on top of HBASE schema assumes just seamless mapping between tuple elements and corresponding columns and column families. Also, it worth using some special HBASE features like:

- efficient querying and filtering by column name (not a value);
- values might have multiple versions.

Addressing that, master profiles, instead of having NULL-valued $MASTERID$ column, should be just suppressed, that speeds up the operation "*Retrieve complete list of unique profiles*". Control over row versions allows to track modifications on *Segments* column family, that speeds up the operation "*Profile Compaction*".

The intention is to to store all rows inside single HBASE table, distinguishing rows by $TYPE$ column. However this is not necessarily and rows might be treated inside different tables depending on type.

The outcome table schema shall address the following items:

- each row is identified by ID;
- "*System*" column family consists of just two columns: $TYPE$ and $MASTERID$;
- "*Segments*" column family to track segments, each segment might be just separated column;
- "*Associations*" column family to keep information about all connected profiles;
- "*Properties*" column values for edges to keep information about edge credibility.

The background operations (**) might be implemented on top of HBASE CoProcessor API [13]. This makes operation execution more transparent and controllable. However it is still feasible to implement the operations on top of dedicated Spark/Scalding applications.

Conclusions. Proposed data model and corresponding operations provide consistent view and API on user profile storage with graph-like connections between each other. Unlike graph databases this model has limited support of graph-like operations and thus is focused primarily on the operations being vital for DSP. The data model comes with mapping to HBASE database. One of the strong schema's sides is its simplicity. That gives an opportunity to adapt the schema for existent HBASE profile databases and enhance them with the model's features.In addition, the model is designed to address strong advantages of NoSQL databases, that gives an opportunity to adapt schema for other NoSQL databases like Cassandra or Aerospike.

References

1. Forecast: RTB Ad Spending To Be One-Fifth of Total Display Advertising in 2013. [Electronic resource] // AdExchanger | News and Views on Data-Driven Digital Advertising. – Access mode: http://adexchanger.com/data-nugget/forecast-rtb-ad-spending-to-be-one-fifth-of-total-display-advertising-in-2013/
2. Real Time Bidding Market (RTB) to Hit $42 Billion by 2018: 41% CAGR for 2014–2019. [Electronic resource] // PR Newswire: press release distribution, targeting, monitoring and marketing. - Access mode: http://www.prnewswire.com/news-releases/real-time-bidding-market-rtb-to-hit-42-billion-by-2018-41-cagr-for-2014-2019-282863381.html
3. Trofimov B.F. User Identification Problems on DSP Side in Terms of Advertising RTB Auctions/ B.F. Trofimov, E.A. Arsiriy, A.V. Arsiriy // Information Technologies in Innovation Business conference (ITIB). – 7 – 9, October, Kharkiv, 2015. – pp. 97 – 100.
4. Neo4j database official web site [Electronic resource] // Neo4j®, Cypher®, and Neo Technology® are registered trademarks of Neo Technology, Inc. – Access mode: – http://neo4j.com/
5. Titan database official web site [Electronic resource] //TITAN Distributed Graph Database – Access mode: – http://thinkaurelius.github.io/titan/
6. S2Graph database official web site [Electronic resource] // GitHub, Inc., 2015 – Access mode: – https://github.com/kakao/s2graph
7. Blueprints official web site [Electronic resource] // GitHub, Inc., 2015 – Access mode: – https://github.com/tinkerpop/blueprints/wiki
8. Gremlin official web site [Electronic resource] // GitHub, Inc., 2015 – Access mode: – https://github.com/tinkerpop/gremlin/wiki

9. Titan Data Model [Electronic resource] – Titan Documentation Chapter 54. Titan Data Model – Access mode: – http://s3.thinkaurelius.com/docs/titan/0.5.0/data-model.html

10. Hbase shims inside Titan Source Code [Electronic resource] // GitHub, Inc., 2015 – Access mode: – https://github.com/thinkaurelius/titan/tree/titan10/titan-hbase-parent

11. Scalding Framework Official web site Code [Electronic resource] // GitHub, Inc., 2015 – Access mode: – https://github.com/twitter/scalding

12. Titan: The Rise of Big Graph Data [Electronic resource] / Rodriguez, M.A., Broecheler, M., // Public Lecture at Jive Software, Palo Alto, 2012.– Access mode: – http://thinkaurelius.com/2012/10/25/a-solution-to-the-supernode-problem/

13. Hbase CoProcessor Introduction [Electronic resource] /Trend Micro Hadoop Group: Mingjie Lai, Eugene Koontz, Andrew Purtell // The Apache Software Foundation Blogging in Action, .– Feb 01, 2012 .– Access mode: – https://blogs.apache.org/hbase/entry/coprocessor_introduction

Отримано 10.10.2015 р.