Towards a distributed infrastructure for evolving graph analytics

Vera Zaychik Moffitt
Drexel University
zaychik@drexel.edu

Julia Stoyanovich
Drexel University
stoyanovich@drexel.edu

ABSTRACT

Graphs are used to represent a plethora of phenomena, from the Web and social networks, to biological pathways, to semantic knowledge bases. Arguably the most interesting and important questions one can ask about graphs have to do with their evolution. Which Web pages are showing an increasing popularity trend? How does influence propagate in social networks? How does knowledge evolve?

In this paper we present our ongoing work on the Portal system, an open-source distributed framework for evolving graph analytics. Our system implements a declarative query language by the same name. Portal streamlines exploratory analysis of evolving graphs, making it efficient and usable, and providing critical tools to computational and data scientists.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

Keywords

evolving graphs, query languages, distributed computation

1. INTRODUCTION

The development of SQL, a declarative query language for relational data analysis, had a tremendous impact on the usability of database technology, leading to its wide-spread adoption. At the same time, the separation between the logical and the physical representations paved the way for powerful performance optimizations. The motivation for our research is to provide a similar tool for analyzing *evolving graphs*, an area of interest in many research communities, including sociology, epidemiology, networking, etc.

Arguably the most interesting and important questions one can ask about networks have to do with their evolution, rather than with their static state. Analysis of *evolving graphs* has been receiving increasing attention, with most progress taking place in the last decade [2, 6, 12, 18, 21, 23].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Some areas where evolving graphs are being studied are social network analysis [8, 15, 16, 22], biological networks [3, 4, 24] and the Web [7, 20].

Despite much recent interest and activity on the topic, and despite increased variety and availability of evolving graph data, systematic support for scalable querying and analytics over evolving graphs still lacks. This support is urgently needed, due first and foremost to the scalability and efficiency challenges inherent in evolving graph analysis, but also to considerations of usability.

In this paper we present our ongoing work on the Portal system, an open-source distributed framework that fills this gap. Portal streamlines exploratory analysis of evolving graphs, making it efficient and usable, and providing critical tools to computational and data scientists. Importantly, Portal implements a declarative query language for evolving graphs. In what follows, we briefly describe the query language (Section 2) and the Portal system (Section 3) that implements it. We then describe several physical representations of evolving graphs that we developed (Section 4), and show results of a preliminary experimental evaluation (Section 5), which illustrates interesting performance trade-offs when different physical representations are used for different analytics.

2. PORTAL BY EXAMPLE

Portal is a declarative query language for evolving graphs. The language operates on a novel kind of a relation, called a TGraph, which can be provided as a base relation or computed as a view. A TGraph associates a sequence of consecutive non-overlapping open-closed time periods of the same duration with a sequence of snapshots. An example of a 4-snapshot TGraph is given in Figure 1, with vertex and edge relations in Figure 3.

Portal supports unary and binary operations on TGraphs, and is fully compositional. Portal uses SQL-like syntax, and has the form TSelect ... From ... TWhere ... TGroup. We prefix temporal keywords with T, to make the distinction between Portal and SQL operations explicit.

Consider query Q1 below. This query is concise, yet it specifies a sophisticated analysis task.

```
Q1:  TSelect V [vid, pagerank()] ;
              E [vid1, vid2, sum(cnt)]
     From     T1 TOr T2
     TWhere   Start >= 2010 And End < 2014
     TGroup by 2 years
```

Q1 combines TGraphs T1 and T2, restricts the result to the

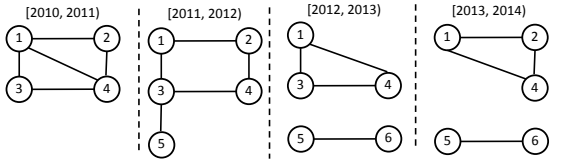


Figure 1: TGraph T1 with 4 snapshots.

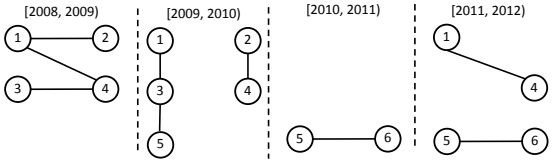


Figure 2: TGraph T2 with 4 snapshots.

V

[2010, 2011)		
vid	name	salary
1	Alice	$150K
2	Bob	$103K
3	Cathy	$98K
4	Dave	$55K

[2011, 2012)		
vid	name	salary
1	Alice	$155K
2	Bob	$113K
3	Cathy	$105K
4	Dave	$55K
5	Eve	$80K

[2012, 2013)		
vid	name	salary
1	Alice	$155K
3	Cathy	$105K
4	Dave	$55K
5	Eve	$80K
6	Frank	$73

[2013, 2014)		
vid	name	salary
1	Alice	$160K
2	Bob	$100K
4	Dave	$55K
5	Eve	$90K
6	Frank	$70

E

vid1	vid2	cnt
1	2	3
1	3	4
1	4	1
2	4	8
3	4	1

vid1	vid2	cnt
1	2	3
1	3	2
2	4	9
3	4	2
3	5	1

vid1	vid2	cnt
1	3	2
1	4	2
3	4	4
5	6	1

vid1	vid2	cnt
1	2	2
1	4	2
2	4	7
5	6	5

Figure 3: Vertex and edge attributes of TGraph T1.

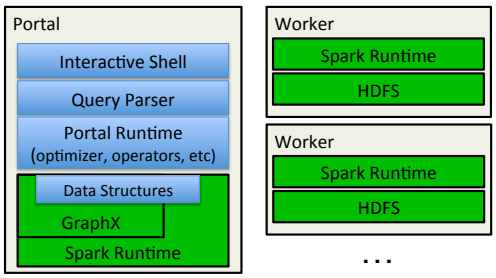


Figure 4: Portal system architecture.

[2010, 2014) range, groups the result into 2-year windows, computes pagerank() for each vertex, and sums values of the attribute cnt for each edge.

Note the use of TOr in Q1. This is one a kind of a temporal join supported in Portal, returning the union of the snapshots of the two operands. (Portal also support temporal intersection TAnd, illustrated in Q2). Temporal join is a binary operation that requires its operands to be *union-compatible*. There are two parts to TGraph union-compatibility, structural and temporal. Structural union-compatibility states that vertex and edge relations of the two TGraphs must be union-compatible. Temporal union-compatibility states that temporal sequences of T1 and T2 must have the same resolution, and they must align.

As part of query Q1, Portal performs *structural aggregation*. This operation is part of temporal aggregation (TGroup) and temporal join (TOr). This default is to take the union of vertices and edges; it can be over-ridden to compute an intersection of the edges, or of the vertices, or both.

Portal supports two families of analytics. The first are snapshot analytics, which are executed on each graph in a series of temporally-adjacent snapshots. The second are trend analytics, computed across groups of temporally-adjacent snapshots. Both are illustrated in Q2 below.

```
Q2:  TSelect V [vid, trend(pr)];
           E [vid1, vid2]
     From    ( TSelect V [vid, pagerank() as pr];
                       E [vid1, vid2]
               From    T1 TAnd T2 )
     TGroup   by Size
```

Q2 executes a temporal join of T1 and T2, invokes pagerank() on consecutive snapshots of the result, and then uses the trend() analytic to compute the PageRank trend across all snapshots of the result.

3. SYSTEM

The Portal system builds on GraphX [10], an Apache Spark library, as depicted in Figure 4. Green boxes indicate built-in components, while blue are those we added for Portal. We selected Apache Spark because it is a popular open-source system, and because of its in-memory processing approach. All language operators on TGraphs are available through the public API of the Portal library, and may be used like any other library in an Apache Spark application.

Query evaluation. Portal query execution follows the traditional query processing steps: parsing, logical plan generation and verification, and physical plan generation. Portal re-uses and extends SparkSQL abstractions for these steps. A Portal query is rewritten into a sequence of operators, and some operators are reordered to improve performance. For example, pushing temporal aggregation before temporal join can sometimes lead to better performance. A temporal join query may be rewritten to include additional temporal selection conditions, based on information about the temporal schema of the TGraphs being joined, which in turn significantly reduces data load time.

We developed several different physical representations and partitioning strategies that are selected at the physical plan generation stage. These are described in Section 4. The TGraphs are read from the distributed file system HDFS and processed by Spark Workers, with the tasks assigned and managed by the runtime.

Integration with SQL. The Portal system includes an interactive shell for exploratory data analysis. Shell users can define (materialized) TGraph views, inspect query execution plans and execute SQL queries with an embedded Portal view. Consider query Q3, a SQL query that returns vid and tr values of 20 vertices with the most significantly increasing pagerank trend.

```
Q3:   Select   VF.vid, VF.tr
      From     T5.toVerticesFlat() as VF
      Order by tr
      Limit    20
```

An important part of Q3 is the use of T5.toVerticesFlat() in the From clause. This is an operation provided by the Portal framework, which collects all vertices in the union of snapshots of T5 into a single nested vertex collection and flattens it into VF (vid:int, start:date, end:date, tr:float, mx:float). VF can be used in SQL queries. Portal also

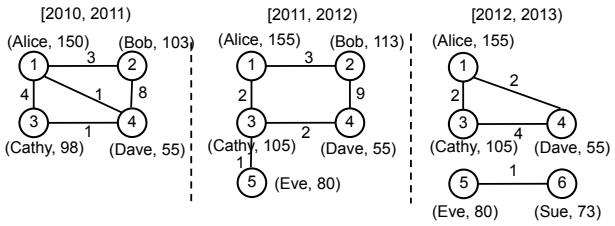
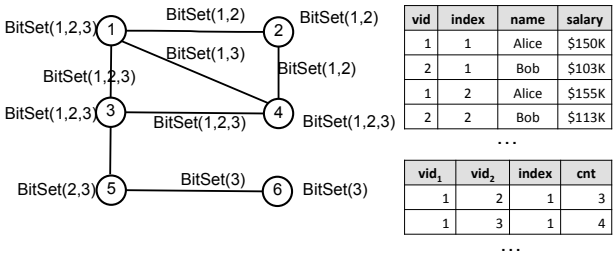


Figure 5: SG representation of T1 from Figure ??.



vid	index	name	salary
1	1	Alice	$150K
2	1	Bob	$103K
1	2	Alice	$155K
2	2	Bob	$113K

…

vid_1	vid_2	index	cnt
1	2	1	3
1	3	1	4

…

Figure 6: OGC of T1 from ??.

provides an operation that returns a flattened collection of edges, called toEdgesFlat().

4. PHYSICAL REPRESENATATIONS

We developed several in-memory representations of evolving graphs to explore the trade-offs of compactness, parallelism, and support of different query operators.

SnapshotGraph (SG). The simplest way to represent an evolving graph is by representing each snapshot individually, a direct translation of our logical data model. We call this data structure SnapshotGraph, or SG for short. An example of an SG is depicted in Figure 5. SG is a collection of snapshots, where vertices and edges store the attribute values for the specific time interval. A TSelect operation on this representation is a slice of the snapshot sequence, while TGroup and temporal joins (TAnd and TOr) require a group by key within each aggregate set of vertices and edges.

While the SG representation is simple, it is not compact, considering that in many real-world evolving graphs there is a 80% or larger similarity between consecutive snapshots [18]. In a distributed architecture, however, this data structure provides some benefits as operations on it can be easily parallelized, by assigning different snapshots to different workers, or by partitioning a snapshot across workers.

OneGraphColumn (OGC). The most topologically compact representation of graph structure is to store each vertex *and* each edge only once for the whole evolving graph, by taking a union of the snapshot vertex and edge sets. The OneGraphColumn data structure, or OGC for short, uses this representation in our system.

The OGC data structure provides some benefits in addition to compactness, since it reduces the total communication between vertices in Pregel-based analytics in batch mode. The drawback is that OGC is much denser than individual snapshots of SG. OGC stores vertex and edge attribute information separately. This is not as compact as storing attributes within the graph elements, but is faster in many operations where only graph topology is required.

Third data structure
Partitioning, briefly.

5. EXPERIMENTAL EVALUATION

Experimental environment. All experiments in this section were conducted on an 8-slave in-house Open Stack cloud, using Linux Ubuntu 14.04 and Spark v1.4. Each node has 4 cores and 16 GB of RAM. Spark Standalone cluster manager and Hadoop 2.6 were used.

Because Spark is a lazy evaluation system, a materialize operation was appended to the end of each query, which consisted of the count of nodes and edges. In cases where the goal was to evaluate a specific operation in isolation, we used warm start, which consisted of materializing the graph upon load. Each experiment was conducted 3 times, we report the average running time, which is representative because we took great care to control variability. Standard deviation for each measure is at or below 5% of the mean except in cases of very small running times.

Data. We evaluate performance of our framework on two real open-source datasets. DBLP [1] contains co-authorship information from 1936 through 2015, with over 1.5 million author nodes and over 6 million undirected co-authorship edges. Total data size: 250 MB. nGrams [11] contains word co-occurrence information from 1520 through 2008, with over 1.5 million word nodes and over 65 million undirected co-occurrence edges. Total data size: 40 GB.

The nGrams dataset is of comparable size to the LiveJournal dataset in [25] and is commensurate with our cluster size. DBLP and nGrams differ not only in size, but also in the evolutionary properties: co-authorship network nodes and edges have limited lifespan, while the nGrams network grows over time, with nodes and edges persisting for long duration. All figures in the body of this section are on the larger nGrams dataset. Refer to the Appendix for the DBLP figures, which show similar trends as nGrams. We plan to carry out further experiments with a larger DELIS[1] dataset as we grow the cluster in the near future.

6. RELATED WORK

Querying and analytics. There has been much recent activity in developing analytics for evolving graphs, see [] for a recent survey. This work is synergistic with ours, in that we build a platform that supports efficient execution of a variety of existing analytics.

Portal is the first proposed declarative query language for evolving graphs. Several researchers have proposed individual queries, without the unifying syntax or generality. For example, Kan, et al. [12] propose a query model for evolving graphs, where queries are for discovering matches for a specific spatio-temporal pattern. The only operator in this model is selection, with two inputs: a temporal pattern and an arbitrary predicate. Semertzidis, et al. [23] also provide querying of evolving graphs, with the focus on building indexes to support historical reachability queries. The most extensive system to date for querying of evolving graphs is ImmortalGraph [18], an in-memory execution engine for temporal graph analytics. ImmortalGraph distinguishes between queries that do one-time random-storage IO (query vertex/edge and query vertex/edge changes) and repeated

[1] law.di.unimi.it/webdata/uk-union-2006-06-2007-05

graph traversals in memory such as analytics. Our contribution is an implementation-independent declarative language to support both iterative analytics and building-block operations such as joins and aggregations.

There are other areas that are related to our work but not closely. For example, this work is different than work on query processing of dynamic graphs (e.g., [19]), where the history of changes is not important, and the focus is on updating the results as the graph undergoes changes. Similarly, work on querying evolving graphs is different than the problem of mining evolving graph streams (e.g., [17]), where the focus is on discovery of significant changes over a small window of consecutive graphs.

Data representation. Another important area of research is efficient storage, retrieval, and in-memory representation of evolving graphs. Various variants of delta-based storage have been investigated in the literature [**?**, 18, 14] and a combination of deltas with selected materialized snapshots appears to provide the best balance of performance and compactness. Most notably, Khurana and Deshpande [**?**] investigate efficient physical representations using deltas for a variety of datasets to support snapshot retrieval, including different cases of skew. Miao, et al. [18] use a similar approach, but demonstrate that there is a tradeoff between temporal and spatial co-location format based on the type of query. For example, large temporal range queries benefit from the time-locality layout in their experiments.

Hybrid delta-snapshot approach provides an ability to support an arbitrary evolving graph resolution. Portal system basic building block is a snapshot with the limitation that only resolutions no smaller than the ones at which snapshots are taken can be computed. It would be fairly easy, however, to modify our system to store data as deltas rather than snapshots, using Khurana's approach. Boldi et al. [5] present a space-efficient non-delta approach for storing a large evolving web graph that they harvested. Their approach for encoding the presence or absence of nodes/edges at each time interval using bits is similar to our MGC data structure and use of BitSets. The primary difference is that their work represents purely topological information and does not address vertex and edge attributes, which still need to be represented and accessed by some queries.

In-memory evolving graph representation work in the literature aligns with the data structures we described in Section **??**. For example, Ren, et al. [21] compute and store representative graphs, whch are equal to our All and Any aggregations over snapshot clusters. The main approach is to store representative graphs in memory for each cluster rather than all the snapshots, and to store deltas from representative graphs to each snapshot if access to an individual snapshot is required. A further optimization is to store deltas to only the first snapshot in the cluster, and then changes to successive snapshots from the first instead of deltas. This is similar to the change tree approach in [13], although for memory-based representation. With a further optimization of exploring inter-cluster similarity, the total storage savings are large (reported only 0.86% of a raw dataset needed in one example). The OneGraph data structure we employ can be thought of as a representative graph for the whole selected time period. The GraphPool in-memory graph storage [13] maintains a single graph representing all retrieved snapshots, and is thus similar to our MultiGraph and OneGraph data structures. The GraphPool goes further and stores only dependencies from a materialized snapshot in those cases where the deltas between two snapshots are small. Our implementation does not take that step because evaluation of queries involving multiple snapshots, such as TGroup, requires fully materialized views in memory. TimeReach [23] uses a representation called a version graph, where each node and edge is annotated with the set of time intervals which which it existed. This representation is similar to our OneGraph data structure, with the difference that we also store non-topological attribute information and OneGraph does not compress consecutive existence intervals with the same value into one. While this optimization is compact, it makes some queries, such as TGroup and snapshot analytics, highly complex.

Distributed frameworks, partitioning. In this paper we build upon, and non-trivially extend, the graph processing abstractions of Apache Spark, a popular open-source distributed data processing engine, and specifically of GraphX [10]. GraphX provides an API for working with regular graphs (snapshots), without the time dimension. As described in **??**, we build our TGraph data structures on top of GraphX Graph, and provide a Portal language interpreter in addition to scala API. In order to provide better performance, we modified GraphX graph loading code to enable concurrent distributed multi-file loading with tuned number of partitions. Default number of partitions used to read in graphs, in contrast, is slow and, for large graphs, leads to OOM errors.

Distributed graph computation requires balanced graph partitioning dependent on the nature of the operation being applied to the graph. There are two basic types of graph partitions: edge-cut and vertex-cut. An edge-cut approach distributes vertices across the available machines and replicates the edges as necessary. A vertex-cut approach does the opposite. Based on extensive theoretical and experimental [9] studies, GraphX chose the vertex-cut approach, which minimizes the communication overhead. The first GraphX paper [25] provides an in-depth explanation of how this is supported. One of the built-in partition strategies, Edge 2D, provides a performance guarantee as a maximum number of replications of a vertex. We developed additional partition strategies (all vertex-cut) based on the temporal dimension as well as hybrids of structural and temporal.

Another framework build on top of Spark is Shark (now SparkSQL) [26]. Shark provides mid-query optimization of the distributed execution plan in addition to other optimizations over spark such as better data representation. Query optimization is one of our goals for the Portal system, starting with dynamic adjustment of the number of partitions and the partition strategy, but that work is still in its infancy and we do not report on it here. One immediately relevant aspect of the Shark work is in partitioning. Namely, in Shark tables can be copartitioned on a particular key at creation time which facilitates much more efficient joins. This is similar to what some of our partition strategies do for tgroup and temporal joins to reduce communication overhead.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented Portal, a declarative query language for evolving graphs. We also proposed an implementation of Portal in scope of Apache Spark, a distributed open-source processing framework. We implemented several physical representations of evolving graphs, and several par-

titioning strategies, and studied their relative performance for Portal operators.

Our experiments demonstrate interesting trade-offs between spatial and temporal locality. This work opens many avenues for future work. It is in our immediate plans to start work on a query optimizer for Portal. We will also implement and evaluate additional TGraph representations that explore the trade-off between density and compactness, and between temporal and structural locality. Finally, we are working on extending the class of trend analytics, and on optimizing the performance of snapshot and trend analytics.

8. REFERENCES

[1] DBLP, computer science bibliography. `http://dblp.uni-trier.de/`. [Online; accessed 14-November-2015].

[2] C. C. Aggarwal and K. Subbian. Evolutionary network analysis. *ACM Comput. Surv.*, 47(1):10:1–10:36, 2014.

[3] S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *TKDD*, 3(4), 2009.

[4] A. Beyer, P. Thomason, X. Li, J. Scott, and J. Fisher. Mechanistic insights into metabolic disturbance during type-2 diabetes and obesity using qualitative networks. *T. Comp. Sys. Biology*, 12:146–162, 2010.

[5] P. Boldi, M. Santini, and S. Vigna. A Large Time-Aware Web Graph. *ACM SIGIR Forum*, 42(2):33–38, 2008.

[6] J. Chan, J. Bailey, and C. Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowl. Inf.Syst.*, 16(1):53–96, 2008.

[7] J. Chan, J. Bailey, and C. Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowl. Inf. Syst.*, 16(1):53–96, 2008.

[8] M. Goetz, J. Leskovec, M. McGlohon, and C. Faloutsos. Modeling blog dynamics. In *ICWSM*, 2009.

[9] J. Gonzalez, Y. Low, and H. Gu. PowerGraph: Distributed graph-parallel computation on natural graphs. In *OSDI'12 Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, pages 17–30, 2012.

[10] J. E. Gonzalez et al. GraphX: Graph processing in a distributed dataflow framework. In *OSDI*, 2014.

[11] Google. The google books Ngram dataset. `http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`. [Online; accessed 14-November-2015].

[12] A. Kan, J. Chan, J. Bailey, and C. Leckie. A Query Based Approach for Mining Evolving Graphs. In *AusDM*, 2009.

[13] U. Khurana and A. Deshpande. Efficient Snapshot Retrieval over Historical Graph Data. In *ICDE*, pages 997 – 1008, Brisbane, QLD, 2013.

[14] G. Koloniari. On Graph Deltas for Historical Queries. In *Proceedings of 1st Workshop on Online Social Systems (WOSS) 2012*, Istanbul, Turkey, 2012.

[15] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1), 2007.

[16] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *ACM SIGKDD*, pages 462–470, 2008.

[17] Z. Liu and J. Yu. Discovering burst areas in fast evolving graphs. In H. Kitagawa, Y. Ishikawa, Q. Li, and C. Watanabe, editors, *Database Systems for Advanced Applications*, volume 5981 of *Lecture Notes in Computer Science*, pages 171–185. Springer Berlin Heidelberg, 2010.

[18] Y. Miao et al. ImmortalGraph: A system for storage and analysis of temporal graphs. *TOS*, 11(3):14, 2015.

[19] J. Mondal and A. Deshpande. Managing large dynamic graphs efficiently. In *ACM SIGMOD*, page 145, 2012.

[20] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *J. Internet Services and Applications*, 1(1):19–30, 2010.

[21] C. Ren, E. Lo, B. Kao, X. Zhu, and R. Cheng. On Querying Historical Evolving Graph Sequences. *PVLDB*, 4(11):726–737, 2011.

[22] P. Sarkar, D. Chakrabarti, and M. I. Jordan. Nonparametric link prediction in dynamic networks. In *ICML*, 2012.

[23] K. Semertzidis, K. Lillis, and E. Pitoura. TimeReach: Historical Reachability Queries on Evolving Graphs. In *EDBT*, 2015.

[24] J. M. Stuart, E. Segal, D. Koller, and S. K. Kim. A gene-coexpression network for global discovery of conserved genetic modules. *Science*, 5643(302):249—255, 2003.

[25] R. S. Xin, J. E. Gonzalez, M. J. Franklin, I. Stoica, and U. C. Berkeley. GraphX : A Resilient Distributed Graph System on Spark. In *GRADES*, New York, New York, USA, 2013.

[26] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, I. Stoica, and U. C. Berkeley. Shark : SQL and Rich Analytics at Scale. In *ACM SIGMOD*, 2013.