Streaming Provenance Compression

Raza Ahmad[1], Melanie Bru[2⋆], and Ashish Gehani[1]

[1] SRI International, Menlo Park CA, USA
[2] Ecole Polytechnique, Palaiseau, France
{raza.ahmad, melanie.bru, ashish.gehani}@sri.com

Abstract. Operating system data provenance has a range of applications, such as security monitoring, debugging heterogeneous runtime environments, and profiling complex applications. However, fine-grained collection of provenance over extended periods of time can result in large amounts of metadata. Xie *et al.* describe an algorithm that leverages the subgraph similarity and locality of reference in provenance graphs to perform batch compression. We build on their effort to construct an online version that can perform streaming compression in SPADE. Our optimizations provide both performance and compression improvements over their baseline.

1 Introduction

Constructing streams of provenance online facilitates a range of real-time applications, including debugging runtime environments and profiling workflows comprised of diverse components. Systems like SPADE[2] are challenged to process, store, and query large streams efficiently within short windows of time. One solution to alleviate the problem is to reduce the size of the provenance metadata before committing it to persistent storage. To this end, several methods have been presented for specific use cases [1,3,4].

Xie *et al.* [5] describe how to store provenance efficiently using techniques from web graph compression. They divide the information contained in a provenance graph into *identity* and *ancestor* information. Identity information is comprised of annotations on edges and vertices. It is compressed using dictionary encoding to eliminate information duplication. Ancestor information describes dependencies between vertices. It consists of a set of edges represented as an adjacency list. This list is encoded using three steps: *reference compression*, *run-length encoding*, and *delta encoding*. Reference compression finds a reference r for each vertex v, such that their adjacency lists have maximum overlap. This overlap is stored only once with non-overlapping elements stored using run-length encoding and delta encoding. These methods save space by utilizing consecutive subsequences and storing differences between the identifiers of successive elements, respectively.

We improve Xie *et al.*'s algorithm with several optimizations. Our implementation provides better performance and compression when compared with the baseline, as shown in section 2.

⋆ While visiting SRI.

2 Contributions

We implemented the improvements in SPADE, an open source data provenance framework with a decentralized architecture. In our evaluation, data was collected from Linux Audit over 78 minutes. It is comprised of 73 thousand vertices and 200 thousand edges. To realize a reproducible stream processing setting, the audit log was replayed in SPADE while performing online compression. We optimize Xie *et al.*'s algorithm as follows:

Bidirectional Traversal: Xie *et al.* only stored the parent information for vertices. This information is insufficient to satisfy common provenance queries efficiently, such as finding all neighbors of a given vertex or finding descendants of a vertex. We include numeric identifiers for both the *parents* and *children* of every vertex. During insertion, we separately compress and store them in the adjacency list.

Node	Parents
54000	41000, 45000, 48000, 50000, 51000, 53000
55000	42000, 43000, 47000, 48000
56000	42000, 43000, 44000, 45000, 49000, 50000, 51000, 52000

Reference Encoding

Node	Reference	Bit List	Remaining Nodes
56000	2	010110	42000, 43000, 44000, 49000, 52000

Run Length Encoding

Node	Reference	Bit List	Sequence Start	Length	Remaining Nodes
56000	2	010110	2	3	49000, 52000

Delta Encoding

Node	Reference	Bit List	Sequence Start	Length	Remaining Nodes
56000	2	010110	2	3	-7, 3

Fig. 1. Three steps of the web compression algorithm by Xie *et al.*, illustrated in a step-by-step example.

Reference Selection (ref): During *reference compression*, the reference y for a new vertex x is selected to maximize the overlap between the adjacency lists of x and y. This overlap is stored in x as a bit list of size equal to that of adjacency list of y. If the size of the adjacency list of y is significantly larger than that of x, space is used to store many zeroes. To improve this, we search for a y that optimizes for the maximum number of 1's and miminum number of 0's in the resultant bit list.

Delta Encoding of Sequences (delta): In the second step of run length encoding, each sequence of consecutive identifiers is encoded using the first vertex's identifier followed by the sequence length, as shown in Figure 1. However, the starting vertex identifiers could be very large for big datasets, occupying significant storage for multiple sequences. Hence, we perform delta encoding at this step as well, storing only the differences between successive starting vertex identifiers.

Uncompressed Buffer (buffer): During insertion of an edge $e(x, y)$, a fast lookup of the adjacency lists of previous vertices is needed. Retrieving this information from the disk becomes temporally expensive as database size grows. We buffer the uncompressed adjacency lists for a subset of vertices. The required adjacency list of references can then often be found in memory during compression, eliminating the time needed to search the disk and uncompress the list.

Adjacency List Caching (cache): Even with the above uncompressed buffer, some queries may need to be resolved using slower persistent storage. We implemented an in-memory cache of compressed adjacency lists to improve performance. When an element is not found in the uncompressed buffer, this cache is consulted. To maintain consistency, the cache is periodically synchronized with the underlying database. This allows end user queries to be satisfied while provenance elements continue to stream into the system.

In the case that the workload is small enough or sufficiently compressible, the entire adjacency list may fit in memory. This case results in the highest-performance insertion and querying.

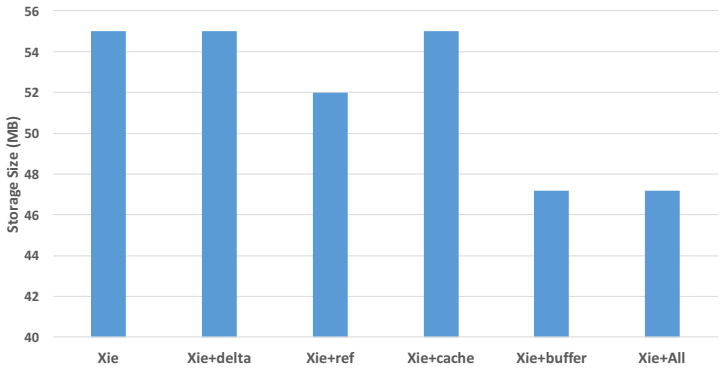

Fig. 2. Effect of individual optimizations on total storage size. *Xie+All* is the case when they are combined.

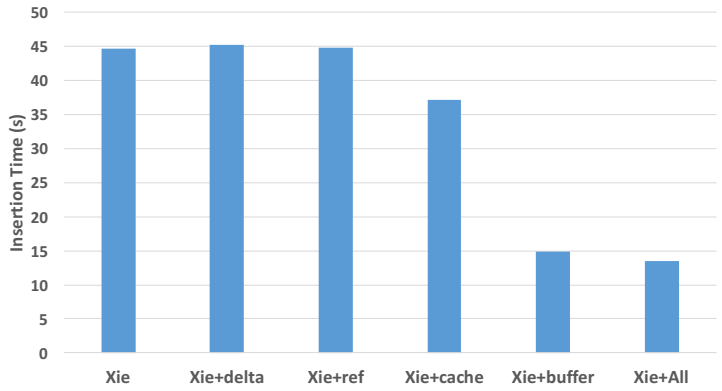

Fig. 3. Effect of individual optimizations on time taken to insert all records in the database.

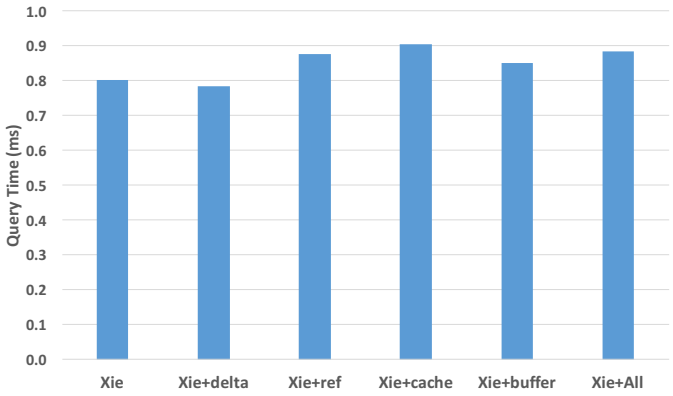

Fig. 4. Effect of individual optimizations on query execution time. Query time is the average time taken to execute 1000 lineage descendant queries of depth 5, starting from randomly chosen vertices.

We implemented five optimizations and studied their effect on storage size, insertion time, and query time. The baseline for comparison is our reimplementation of Xie *et al.*'s algorithm. When all our optimizations are employed, the size of the compressed provenance significantly decreases when compared to the baseline, as illustrated in Figure 2. Our approach improves insertion times by a factor of three, when compared to the baseline, as seen in Figure 3. This is of particular import in a streaming setting. Finally, we report query time performance in Figure 4.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant ACI-1547467. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Adriane Chapman, Hosagrahar Jagadish, and Prakash Ramanan, **Efficient provenance storage**, *34th ACM International Conference on Management of Data (SIGMOD)*, 2008.
2. Ashish Gehani, Hasanat Kazmi, and Hassaan Irshad, **Scaling SPADE to "big provenance"**, *8th USENIX Workshop on Theory and Practice of Provenance (TaPP)*, 2016.
3. Emmanuel Jeannot, Bjorn Knutsson, and Mats Bjorkman, **Adaptive online data compression**, *11th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2002.
4. Xiang Li, Xiaoyang Xu, and Tanu Malik, **Interactive provenance summaries for reproducible science**, *12th IEEE Conference on e-Science*, 2016.
5. Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Dan Feng, Yan Li, and Darrell Long, **Evaluation of a hybrid approach for efficient provenance storage**, *ACM Transactions on Storage (TOS)*, Vol. 9(4), 2013.