Compressing Provenance Graphs

Yulai Xie[†‡], *Kiran-Kumar Muniswamy-Reddy*[§], *Darrell D. E. Long*[‡]
Ahmed Amer[¶], *Dan Feng*[†], *Zhipeng Tan*[†]

[†]*Huazhong University of Science and Technology*, *Wuhan National Laboratory for Optoelectronics*, [‡]*University of California, Santa Cruz*, [§]*Harvard University*, [¶]*Santa Clara University*

Abstract

The provenance community has built a number of systems to collect provenance, most of which assume that provenance will be retained indefinitely. However, it is not cost-effective to retain provenance information inefficiently. Since provenance can be viewed as a graph, we note the similarities to web graphs and draw upon techniques from the web compression domain to provide our own novel and improved graph compression solutions for provenance graphs. Our preliminary results show that adapting web compression techniques results in a compression ratio of 2.12:1 to 2.71:1, which we can improve upon to reach ratios of up to 3.31:1.

1 Introduction

Provenance, though extremely valuable, can take up substantial storage space. For instance, in the PReServ [9] provenance store, the original data was 100 KB, while the provenance reached 1 MB. For MiMI [10], an online protein database, the provenance expands to 6 GB while the base data is only 270 MB. Similar results are observed in other systems [3, 11]. This makes provenance data an increasing overhead on the storage subsystem.

A provenance dataset can be represented as a provenance graph [12]. Thus, efficient representation of provenance graphs can fundamentally speed up provenance queries. While, inefficient multi-GB provenance graphs will not fit in limited memory and dramatically reduce query efficiency.

We propose to adapt web compression algorithms to compress provenance graphs. Our motivation comes from our observation that provenance graphs and web graphs have similar structure and some common essential characteristics, *i.e.*, similarity, locality and consecutiveness. We have further discovered that provenance graphs have their own special features, and we propose two new techniques to improve the provenance graph compression.

We test our hypothesis by compressing the provenance graphs generated by the PASS [3] system. Our results show that our web compression algorithms can compress such graphs, and that our improved algorithms can further raise the compression ratio up to 3.31:1.

2 Web & Provenance Graphs

A web graph is a directed graph where each URL is represented as a node, and the link from one page to the other page is a directed edge. There are some key properties exploited by current web graph compression algorithms:

- **Locality:** Few links would go across URL domain, and therefore the vast majority tend to point to pages nearby.
- **Similarity:** Pages that are not far from each other have common neighbors with high probability.
- **Consecutiveness:** The node numbers of successors of a page are in sequential order.

Provenance graphs also have a similar organizational structure and characteristics as web graphs. Figure 1 shows the conversion from a snapshot of a NetBSD provenance trace (generated in the PASS system [3]) to an adjacency list that represents the provenance graph. The notation "A INPUT[ANC] B" in the provenance trace means that B is an ancestor of A, indicating that there exists a directed edge from A pointing to B. In this way, a provenance graph is also a directed graph and each node (*e.g.*, node 2 or 3) has a series of out-neighbors.

Provenance nodes 2 and 3 are similar, as they have common successors in the form of nodes 4, 7, 9 and 14. The reason for this is that many header files or library

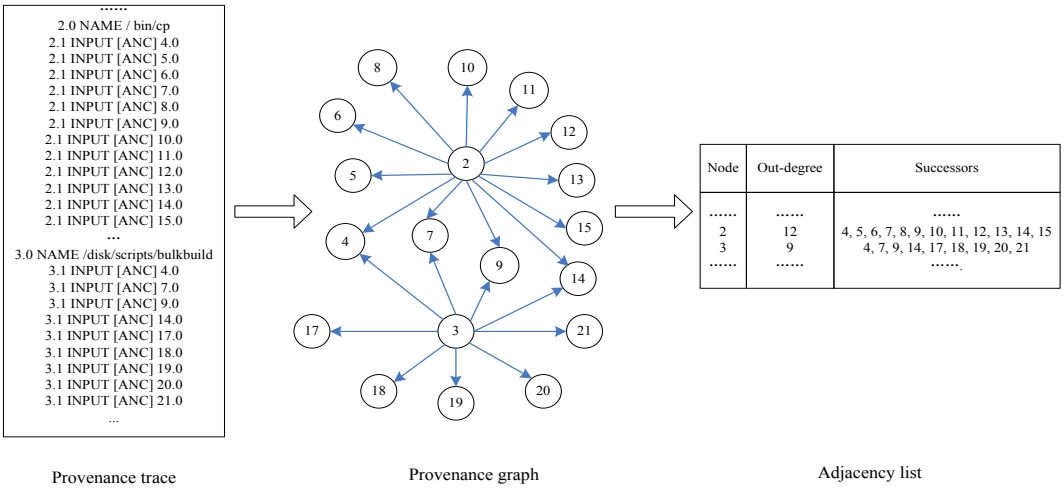

Figure 1: Mapping from the provenance trace to adjacency list that represents the provenance graph. The expression "2.1 INPUT[ANC] 4.0" indicates that node 4 is an ancestor of node 2, resulting in a directed edge from node 2 pointing to node 4. This figure also shows that provenance graph exhibits the similar characteristics (*i.e.*, similarity, locality and consecutiveness) as web graph.

files that are represented as nodes like 4, 7, 9 and 14 are repeatedly used as input by many processes (*e.g.*, nodes 2 and 3). Nodes 2 and 3 also exhibit locality. The successors of provenance node 2 are only between 4 and 15, and the successors of node 3 are only between 4 and 21. This is because many header files or library files (*e.g.*, the successors of nodes 2 and 3) that are used as input by a process are probably in the same directory, so the ordering (and therefore assigned numbers) of these files are usually close to each other. Consecutiveness is also clear in the successors of nodes 2 and 3, from 4 to 15 and from 17 to 21 respectively. The existence of consecutiveness is because a process or file may be generated by many files that do not belong to a PASS volume, and such files are usually numbered sequentially.

3 Related Work

Barga *et al.* [1] presented a layered provenance model for workflow systems that can efficiently reduce provenance size by minimizing the repeated operation information during provenance generation. This is similar to our approach that exploits similarity between the neighbors of different nodes representing the same manipulation. Chapman *et al.* [4] proposed two classes of algorithms to compress provenance graphs: *provenance factorization* and *provenance inheritance*. The former aims to find common subtrees between different nodes and the latter focuses on find similarities between data items that have ancestry relationships or belong to a particular type. These algorithms achieve good compression ratios, but can only be applied to a flat data model, *i.e.*, where each node has complete provenance, and therefore cannot be used to compress provenance generated by a provenance system such as PASS [3]. Our methods are more general and are hence applicable to wider range of systems.

There has been considerable work [5, 6, 7, 8] in the domain of web graph compression. Adler *et al.* [5] proposed to utilize reference compression to compress web graphs. Randall *et al.* [6] suggested a set of technologies such as delta codes and variable-length bit encoding to compress a database providing fast access to web hyperlinks. The critical web graph compression framework was presented by Boldi and Vigna [7]. They obtained good compression performance by fully exploiting the locality and similarity of web pages. The algorithm we use is based on this framework.

There are also classical techniques like LZ-based compression algorithms [2]. These techniques present an upper bound on the compression that is possible. However, since they do not preserve the structure of the data, the resulting compressed graph will not be amenable to querying.

4 Compression Algorithms

Three critical ideas lie behind web compression algorithms [7]: First, encoding the successor list of one node by using the similar successors of another node as a reference, thus efficiently avoiding encoding the duplicate data; Second, encoding consecutive numbers by only recording the start number and length, reducing the num-

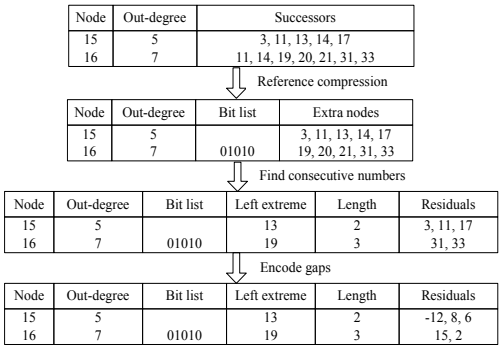

Figure 2: An example on web compression algorithm

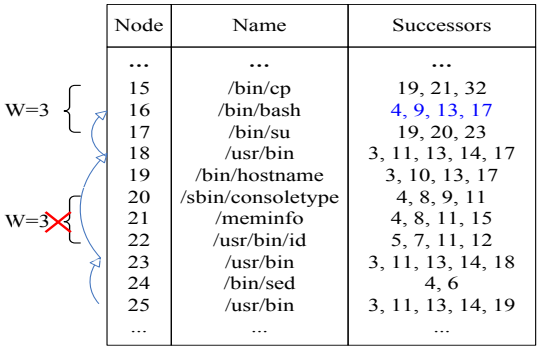

Figure 3: Name-identified reference list

ber of successors to be encoded; and Third, encoding the gap between the successors of a node rather than the successors themselves, which typically requires fewer bits to be encoded.

As we can see in Figure 1, a provenance graph can be represented by a set of provenance nodes which have a series of ancestors as their successors. The provenance nodes are numbered from 0 to $N-1$, in order, during provenance generation. We use Out(x) to denote the successor list of node x. The web compression algorithm to encode this list is detailed as the follows:

1. Reference compression: Find the node with the most similar successor list in the preceding W successor lists. W is a window parameter. Let y be such a reference node, Out(y) is its corresponding successor list (called reference list). The encoding of Out(x) can be divided into two parts: a bit list to identify the common successors between Out(x) and Out(y), and *Extra nodes* that identifies the rest of the successors in Out(x).

2. Find consecutive numbers: Separate the consecutive numbers from the *Extra nodes*, and then represent each set of consecutive numbers by using its left extreme and its length.

3. Encode gaps: Let x_1, x_2, x_3, ..., x_k be the successors of x that have not be encoded after the above step. If $x_1 \leq x_2 \leq ... \leq x_k$, then encode them as $x_1 - x$, $x_2 - x_1$, ..., $x_k - x_{k-1}$.

Figure 2 shows an example on these three steps. In this example, node 15 is the reference for node 16 and has no reference itself. The case in Figure 1 is simpler. The successor list of Node 2 can be encoded using step 2. The successor list of node 3 can be encoded using step 1 and 2, with node 2 serving as the reference list for node 3.

Our Improved Approach

We now describe two improvements beyond existing web compression algorithms. These are motivated by the observed properties of datasets generated by the PASS [3] system.

(a) Name-identified Reference list:

Web compression algorithms find the most similar reference list in the preceding W nodes. The greater the similarity, the better the compression performance achieved. However, sometimes, it's hard to find a very similar reference list with a small W, and while a larger W value would produce a better compression ratio because it enlarges the scope of the possible reference lists, this would be at the expense of slower compression and decompression.

We have found that many provenance datasets record the name of provenance nodes. The nodes with same name usually have a large set of common successors. For instance, in PASS provenance traces, a process that is represented as a provenance node would be scheduled to execute many times, and each time it is scheduled, it will use many of the same header files or library files as input, which are actually the common successors between the process nodes with same name. So we propose to use name as an indicator to help find similar reference lists.

Figure 3 describes how this technology functions in an example. When node 18 with name /usr/bin first appears, the algorithm finds reference list in the preceding W(if $W = 3$) nodes. But when node 23 with name /usr/bin appears the second time, we take the successor list of node 18 as reference list rather than using W. In this algorithm, we need only use a hash table to identify the nodes with same name. So each time we encode the successor list of a node, we just have to confirm whether it is in the hash table, but we need not go backwards through the window list (especially beneficial when W is larger). This would incur only a minimal time overhead on compression and decompression.

Table 2: Performance of web compression algorithm and improved algorithms that incorporate two new technologies with respect to different W for various trace workloads

W	**NetBSD**			**am-utils**			**blast-lite**		
	web	web-name	web-gap	web	web-name	web-gap	web	web-name	web-gap
1	2.00	2.62	2.27	1.60	2.57	1.86	1.66	1.98	1.80
5	2.14	2.62	2.31	1.73	2.57	2.02	1.96	1.98	2.16
10	2.32	2.63	2.69	1.87	2.57	2.23	2.03	2.02	2.25
100	2.71	2.63	3.23	2.58	2.58	3.31	2.12	2.08	2.38

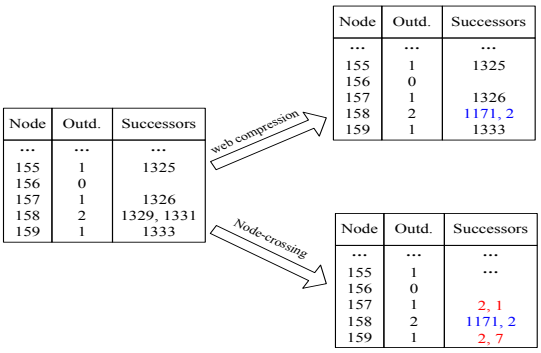

Figure 4: Node-crossing gap

Table 1: The properties of provenance traces

Trace	n	Size
NetBSD	140146	60.2MB
am-utils	46100	24.9MB
blast-lite	240	68kB

(b) Node-crossing gap: Current web compression algorithms exploit graph locality by encoding the gaps between the successors of a node to improve compression ratio. However, in PASS provenance traces, we find that many provenance nodes have only one successor, and these successors cannot be encoded with the current web compression technology. So we propose to exploit gaps between the successors of different nodes. Figure 4 compares the results of using the approach to gap encoding used in current web compression algorithms and with our node-crossing gap encoding approach. Current encodings can only encode the successors of node 158 ($1329-158=1171$, $1331-1329=2$), while our node-crossing approach can further encode the successors of node 157 ($157-155=2$, $1326-1325=1$) and node 159 ($159-157=2$, $1333-1326=7$).

Note that the successor of node 155 is not encoded as $1325-155=1170$ in the current web compression algorithm, which mainly focuses on exploiting the locality between the successors of a node in the usual sense, while our improved approach further exploits the locality between the successors of different nodes.

5 Evaluation

Our experimental datasets are generated by the PASS system, drawn from different applications. They are:

1. NetBSD trace: Build of several components of NetBSD.
2. Compilation workload trace: Compilation of am-utils.
3. Scientific trace: A simple instance of the blast biological workload.

In Table 1, we summarize the number of nodes and the size of these provenance traces.

Table 2 shows the compression performance with respect to different W for various trace workloads. "Web" means to compress the provenance graph using current web compression algorithm. "Web-name" means to use web compression algorithm that incorporates name-identified reference list technology. While "web-gap" means to compress provenance graph using web compression algorithm that incorporates node-crossing gap technology.

For all three trace workloads, the web compression algorithm achieves a better performance with increases in W. This is because a bigger window would increase the likelihood of finding similar reference lists. We also find that for all workloads, the algorithm with name-identified reference lists exhibits very stable performance, *e.g.* 2.62–2.63 times for **NetBSD** trace. With our name-identification approach, window size appears to have limited impact. It results in performance considerably better than basic web compression, requiring a much larger window size for the latter in order to match its performance.

When we consider the algorithm with node-crossing gap encoding, we see it also outperforms web compression, and even outperforms our name-identification approach as window sizes are increased. *E.g.*, when

$W = 10$, $W = 100$ for NetBSD trace, $W = 100$ for **amutils** and $W = 5$, $W = 10$, $W = 100$ for **Blast-lite**. We also note that our node-crossing gap encoding approach achieves the best compression ratios (when $W = 100$) for all the cases.

6 Summary and Future Work

We successfully compressed provenance graphs by adapting techniques from web graph compression, achieving compression ratios of 2.12 to 2.71 times. We also introduced two new techniques based on provenance graph properties, and demonstrated both increased compression ratios (by up to 28%) and computational efficiency.

In the future, we intend to evaluate the impact of web compression on provenance query performance. In addition, we plan on comparing our web compression algorithms with LZ-based compression algorithms and a hybrid scheme involving LZ-based and web-compression based schemes.

Acknowledgments

This work was supported in part by the National Basic Research 973 Program of China under Grant No. 2011CB302300, 863 project 2009AA01A401/2, NSFC No. 61025008, 60933002, 60873028, Changjiang innovative group of Education of China No. IRT0725. This material is based upon work supported in part by: the Department of Energy under Award Number DE-FC02-10ER26017/DE-SC0005417, the Department of Energy's Petascale Data Storage Institute (PDSI) under Award Number DE-FC02-06ER25768, and the National Science Foundation under awards CCF-0937938 and IIP-0934401 (I/UCRC Center for Research in Intelligent Storage).

References

[1] R. S. Barga and L. A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. In Concurrency and Computation: Practice and Experience, 2007

[2] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. IEEE Trans. on Information Theory, 23(3):337-343, 1977.

[3] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer. Provenance-aware storage systems. Proc. USENIX Annual Tech. Conf., 2006.

[4] A. P. Chapman, H. V. Jagadish, P. Ramanan. Efficient Provenance Storage. Proc. SIGMOD, 2008.

[5] M. Adler and M. Mitzenmacher. Towards Compressing Web Graphs. Proc. IEEE Data Compression Conf., 2001.

[6] K. Randall, R. Wickremesinghe, and J. Wiener. The link database: Fast access to graphs of the Web. Research Report 175, Compaq Systems Research Center, Palo Alto, CA, 2001.

[7] P. Boldi and S. Vigna. The webgraph framework I: Compression techniques. Proc. 13th WWW, 2004.

[8] T. Suel and J. Yuan. Compressing the graph structure of the web. Proc. IEEE Data Compression Conf., 2001.

[9] P. Groth, S. Miles, W. Fang, S. C. Wong, K. Zauner and L. Moreau. Recording and using provenance in a protein compressibility experiment. Proc. HPDC, 2005.

[10] M. Jayapandian, A. P. Chapman, V. G. Tarcea, C. Yu, A. Elkiss, A. Ianni, B. Liu, A. Nandi, C. Santos, P. Andrews, B. Athey, D. States, and H.V. jagadish. Michigan Molecular Interactions (MiMI): Putting the jigsaw puzzle together. Nucleic Acids Research, 2007.

[11] Y. Simmhan, B. Plale and D. Gannon. A framework for collecting provenance in data-centric scientific workflows. Proc. ICWS, 2006.

[12] D. A. Holland, U. Braun, D. Maclean, K.-K. Muniswamy-Reddy, M. I. Seltzer. Choosing a data model and query language for provenance. Proc. 2nd Int'l. Provenance and Annotation Workshop, 2008.