Instrumenting a Logic Programming Language to Gather Provenance from an Information Extraction Application

Christine F. Reilly
University of Texas - Pan American
1201 West University Drive
Edinburg, TX 78539
reillycf@utpa.edu

Yueh-Hsuan Chiang
University of Wisconsin - Madison
1210 West Dayton Street
Madison, WI 53706
yhchiang@cs.wisc.edu

Jeffrey F. Naughton
University of Wisconsin - Madison
1210 West Dayton Street
Madison, WI 53706
naughton@cs.wisc.edu

ABSTRACT

Information extraction (IE) programs for the web consume and produce a lot of data. In order to better understand the program output, the developer and user often desire to know the details of how the output was created. Provenance can be used to learn about the creation of the output. We collect fine–grained provenance by leveraging ongoing work in the IE community to write IE programs in a logic programming language. The logic programming language exposes the semantics of the program, allowing us to gather fine–grained provenance during program execution. We discuss a case study using a web–based community information management system, then present results regarding the performance of queries over the provenance data gathered by our logic program interpreter. Our findings show that it is possible to gather useful fine–grained provenance during the execution of a logic based web information extraction program. Additionally, queries over this provenance information can be performed in a reasonable amount of time.

Categories and Subject Descriptors

H. Information Systems [**H.m Miscellaneous**]

General Terms

Design, Performance

Keywords

Provenance, Information Extraction, Logic Programming

1. INTRODUCTION

Information extraction (IE) applications for the web process text data in a multi-step workflow [1, 2]. These applications take a large amount of text as input and use a series of programs to extract items of interest from the text and then discover relationships between those items of interest. After an IE program runs, it is often useful for the developer and user to be able to obtain information about how the results were produced. The information needed to answer questions about the results of a computation is called provenance [3]. The developers of the DBLife community information management system [1] expressed a desire for using provenance to help debug their information extraction application. In the context of information extraction, provenance can be used to discover what input data and processing programs were used to create a given piece of output data.

WWW 2012 Companion, April 16–20, 2012, Lyon, France
ACM 978-1-4503-1230-1/12/04.

2. PROVENANCE AWARE XLOG

Xlog [6] is a logic programming language that adds user defined procedures to Datalog. It was designed to be used for information extraction tasks. We created Provenance Aware Xlog (PAXlog) by writing a Xlog interpreter that collects provenance during the execution of a program. The interpreter interacts with a relational database which stores the collected provenance information. As the interpreter parses a program, it detects and records information about program structure. While each rule in the program is evaluated, the instantiation of the predicates that are part of the rule is logged to the database. We then use SQL to write provenance queries.

In order to make the best use of PAXlog and obtain the most provenance information, application programmers should follow five guidelines. First, we assume the program associated with each user defined procedure is stateless. Second, each procedure should do a small amount of work. Because PAXlog sees each procedure and sees the inputs and outputs to that procedure, having many procedures that each do a small amount of work exposes a lot of information to PAXlog. The third guideline is that reading/writing from/to external files should be done by a predicate of the program. Using predicates for I/O exposes the data being read/written to PAXlog. Our fourth guideline is that if a procedure directly access a file, it specify the file identification as one of its attributes. This ensures that PAXlog will record information about the file. Finally, when a Xlog program has multiple rules that add data to the same predicate, keep the rules in separate predicates until the final program output. This ensures that PAXlog is able to track which of the rules a particular piece of data originates from.

3. CASE STUDY: DBLIFE

DBLife [1], a community information management system for the database research community, crawls the web pages of community members, extracts information about the community, and finds relationships between the various entities. Because DBLife is a very large and complex system, we created a small Xlog program modeled after DBLife for our case study. Our program uses two input files: a list of conference names, and a list of community web pages. The

procedures in our program search for conference names and paper titles in these web pages. The program then matches each paper with the conference where it was presented.

We wrote three queries to examine the provenance collected by PAXlog. The first query starts with an output tuple and searches for the instantiation of the Xlog program that produced that tuple. We chose an output tuple that we suspected to have errors and were able to discover what procedures were used to create that tuple. With this knowledge, we can identify the procedure that needs to be debugged. Our second query searches for all output tuples that used a specific input tuple from the list of web pages. We found that our program identified twelve conference papers on a given web page and that the program misidentified many strings that are not paper titles. We know that the given web page contains more than twelve paper titles and can conclude from this query that our method for finding paper titles needs to be refined. The final query identifies output tuples that were created using a given procedure. This query can be used to identify out of date output tuples when the procedure has been modified.

4. QUERY PERFORMANCE

Using the PAXlog interpreter to gather provenance from applications will only be useful if the queries over the provenance data can be evaluated in a reasonable amount of time. Our example queries use recursive SQL. It has been suggested that using recursive SQL to query provenance is expensive and unnatural [5]. For our query performance experiments we generated artificial Xlog programs of various shapes and sizes. We also created artificial input data.

In the first experiment, we examine how the shape of a Xlog program affects the performance of recursive SQL queries over the provenance for that Xlog program. We found that the query run time decreases as Xlog programs become wider. One possible reason for this finding is that wider programs are naturally shallower and therefore have fewer levels of recursion. Another reason is that as programs become wider the size of the provenance information collected decreases.

The second experiment explores how the query execution time is affected by the number of input tuples. We found that even with 100,000 tuples in each input table, Queries 1 and 2 have relatively reasonable performance (5 to 6 seconds). This query time may not be acceptable in an interactive application, but is likely to be acceptable in a batch query processing scenario.

Our example queries use a nesting operation that allows the queries to search for an ordered list of attributes that we store in separate tuples. This nesting operation creates a new table with these attribute values listed as a single attribute. When we ran a query that performs the nesting operation as part of the query, that query took seven and one half hours to complete. If we instead create a materialized view of the nested table (taking 16 seconds), the same query takes less than one second to complete.

5. CONCLUSIONS

We have presented PAXlog, a logic programming language for web IE programs that is instrumented to gather provenance during program execution. Through our case study using DBLife, we have demonstrated that PAXlog gathers the necessary information for answering queries of interest to both developers and users of the web IE system.

Our performance study showed that queries over the provenance information gathered by PAXlog are able to run in a reasonable amount of time. It should be noted that we did not write the Xlog interpreter with performance in mind. Therefore, we did not measure the performance of the interpreter.

We have a number of possible directions for future work. One direction is to improve the performance of the Xlog interpreter. Another avenue for future work is to examine whether a relational database system is the proper place for storing this type of provenance information. Other models, such as the Open Provenance Model [4] may provide allow queries that are less complex. Finally, we would like to explore how well PAXlog extends to applications beyond information extraction.

6. ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation Award SCI-0515491. Many thanks to the DBLife team.

7. REFERENCES

[1] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. DBLife: A community information management platform for the database research community. In *CIDR-07*, 2007.

[2] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases.*, 29(1), 2006.

[3] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, May/June 2008.

[4] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[5] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Margo, M. Seltzer, and R. Smogor. Layering in provenance systems. In *Proceedings of the 2009 USENIX Annual Technical Conference*, San Diego, California, June 2009.

[6] W. Shen, A. Doan, J. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *Proceedings of the 33rd VLDB Conference*, pages 1033–1044. VLDB Endowment, 2007.