Data Provenance in Distributed Propagator Networks

Ian Jacobi

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
jacobi@csail.mit.edu

Abstract. The heterogeneous and unreliable nature of distributed systems has created a distinct need for the inclusion of provenance within their design to allow for error correction and redundancy. Many traditional distributed systems have limited provenance tracing abilities, usually included in generic workflow generation or in an application-specific way. The novel programming paradigm of distributed propagator networks allows for the inclusion of provenance from the ground up.
In this paper, I present the concept of propagator networks and demonstrate how provenance may be easily integrated into programs built using them. I also demonstrate the possibility of converting non-provenance-aware applications built using propagator networks into provenance-aware applications by simply performing a transformation of the existing program structure.

1 Introduction

Data provenance, that is, the derivation history of a piece of data [1], is an integral need of distributed systems like Google's MapReduce algorithm [2], or BOINC [3]. In distributed systems such as these, it is difficult to infer provenance of a particular result as the result may be generated by any one of thousands of systems. As a result, provenance handling must be explicitly factored into distributed system design.

Depending on how well an existing distributed architecture is designed, it may be difficult to support many use cases of provenance in applications that use the architecture. Such programs may need to explicitly include provenance in the design of the application. It would be far easier for developers of distributed applications not to need to worry about how provenance is handled in their distributed system; this would reduce complexity of program design. Data propagation, a model of concurrent [4] and distributed computation [5], allows for the transformation of programs that use it so they may track provenance.

2 Propagator Networks

Propagator networks, developed by Radul [4], are a general-purpose concurrent programming paradigm. These bipartite networks are constructed by connecting

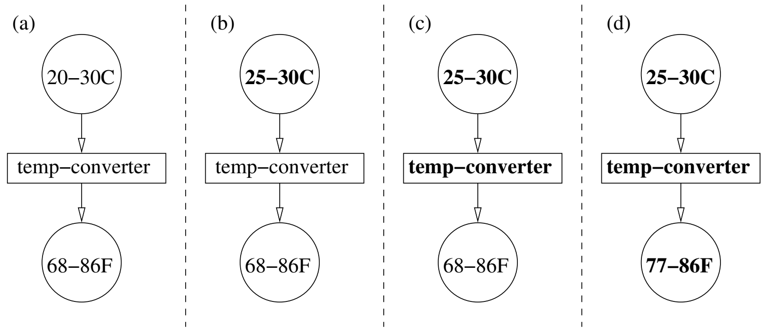

Fig. 1. A refinement of the contents of the top cell in (b) causes `temp-converter` to fire in (c), which then refines the bottom cell in (d).

"cells" that permanently store state and stateless propagators which perform computation and update the cells they are connected to.

Cells are a form of memory which may be assigned a partial value that can be refined. Upon receiving an update to this value, a cell accumulates knowledge of the value by applying a user-defined merge operation to unify the information contained in the update with the partial value currently stored there.

After a cell has changed its state by merging an update, any propagators that have registered an interest in the cell wake up and begin to process, as in (c) in Figure 1. These propagators may then send updates to other cells (d) and cause another cycle of cell merging and notification of propagators; this drives continued computation.

Networks of propagators have no constraints on their topology and may contain cycles. The order of operations in propagator networks is undefined other than the ordering enforced by propagation itself (i.e. a cell must update before propagators attached to it may fire). This, along with the separation of state and computation makes propagation a flexible framework for concurrency.

2.1 Propagators in a Distributed System

The modularity of propagator networks makes it relatively simple to extend their use to distributed systems. [5] In order to push updates across a network, we may bridge the network with propagators that duplicate cells on different computers. By implementing a "synchronization propagator" on each host to forward updates between copies of cells, local updates can trigger remote ones, effecting remote computation. This computation may update other cells that then cause cell synchronization and update more remote cells.

To ensure that no inconsistencies arise due to network issues, we require four properties of a cell merge operation: idempotency, associativity, commutativity, and monotonicity. We also require all cell copies to have the same merge operation. Although the only operations that adhere to these constraints may be the operations of logical conjunction and disjunction or comparable operations

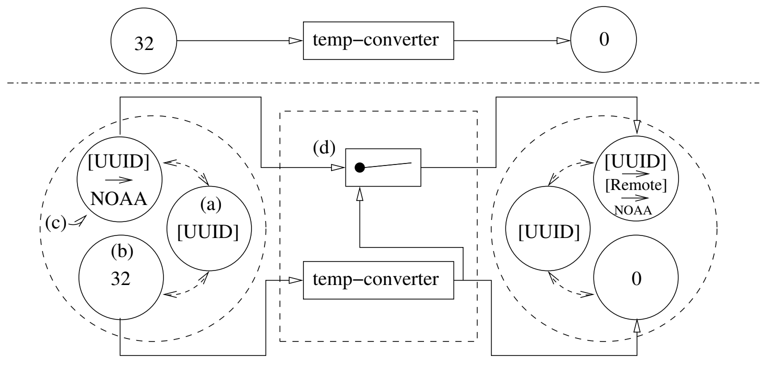

Fig. 2. We transform a propagator `temp-converter` (top) into a provenance-aware propagator network (bottom). Solid arrows indicate the flow of computation in the propagator network. Note that the provenance (c) and data (b) cells contain pointers to the main identifier cell (a) (and vice versa), marked with dashed arrows.

over alternate domains, merge operations that do not adhere to these principles may be modified into operations that do. Standard distributed database implementations account for the transitivity and implicit monotonicity of network communications when performing non-transitive and non-monotonic deletions [6]. Similar adaptations of other merge operations may be possible.

3 Adding Provenance to Propagator Networks

Provenance may be easily implemented on top of existing propagator networks without adding any additional mechanisms or basic primitives to the propagator paradigm. Rather than treating a cell as a single object with a number of simple propagators attached to it, we may apply a simple transformation that adds the cells and propagators needed to make the propagator provenance-aware.

We choose to separate provenance from the data itself, as in Figure 2. In that example, a cell containing a measurement of 32 (degrees Fahrenheit) made by the National Oceanic and Atmospheric Administration (NOAA) is divided into three sub-cells. One cell contains the data (b), another the provenance (c), and a third (a) that points to both sub-cells and contains a Universally Unique Identifier (UUID) along with associated metadata, linking the three cells together.

Separating the sub-cells in this way allows these separate aspects of the data to be refined separately. It also allows for separate access control of provenance and data, as the auditors allowed to view provenance may be different from general users. [7] Each of these three components, data, provenance, and metadata, are placed into one of three sub-cells where they may be refined.

Transforming a propagator to be provenance-aware is even simpler; it requires the propagator to be modified with an additional sub-propagator for each input/output cell pair (d). This sub-propagator will only allow the provenance of an input to be sent to the output cell when data has been sent by the main propagator to the output, effectively acting as a switch.

Provenance may be constructed by gradually aggregating the graph of provenance stored in each provenance sub-cell. Contents of new provenance updates may be added to an existing provenance graph, and then propagated through other provenance-aware propagators. Changes to existing provenance will also be forwarded through a provenance-aware propagator, and these changes may be merged into the existing knowledge of the provenance sub-cell. Thus, data propagation may be used for both computation and provenance construction.

4 Related Work

The work of Moreau, et al. [8] features the automatic construction of provenance, much as provenance-aware data propagation does. However, Moreau focuses on querying the provenance after its construction rather than detailing its generation. Moreau also assumes that provenance may be general in his system, able to document the *purpose* of an action. Although propagator networks may be able to do so, we make no claims about the creation of subjective provenance.

Altintas, et al.'s extension of the Kepler Scientific Workflow System [9] is also similar to the work presented here. Just as we may extend existing propagator networks to support provenance, Altintas demonstrates an extension of an existing framework, Kepler, to support provenance. Altintas's centralized approach for collecting provenance is not suitable for propagator networks however, as propagator networks are inherently decentralized. Adapting Altintas's approach would scale poorly with the propagator network as the number of messages sent to the propagator server grows.

The Matrioshka system presented by da Cruz, et al [10] proposes another mechanism for provenance tracking in distributed workflows. Unlike provenance propagation, Matrioshka requires a single centralized provenance store rather than distributing the provenance with the data. Matrioshka is also somewhat more brittle than the system proposed here, as it assumes that logging is aleady performed, and relies on the generation of a log prior to constructing provenance.

5 Contributions and Future Work

The power of data propagation may resolve many of the difficulties encountered in concurrent and distributed processing, and we should consider the role of integrating provenance into systems that make use of this technique. In this paper I have demonstrated the value of the data propagation paradigm not only by allowing for the distribution of provenance, but also by permitting the extension of existing programs to support provenance.

I have currently implemented the system described in this paper on top of a distributed propagator framework (DProp) that I have designed. While the implementation of both DProp and the provenance-aware component are Python-based, the more generic nature of propagators allows this design to be useful more generally. I hope to eventually test the system across a number of hosts to ensure that the system is fully scalable.

6 Acknowledgements

I would like to thank Joe Pato of HP Labs, and Gerry Sussman and other members of the Decentralized Information Group at MIT for their advice and criticism. I would like to extend particular thanks to Alexey Radul for his assistance with the architectural design of distributed propagation. Finally, I would also like to acknowledge that this work was supported in part by the National Science Foundation under NSF Cybertrust Grant award number CNS-0831442 and by IARPA under Grant FA8750-07-2-0031.

References

1. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. ACM SIGMOD Record **34**(3) (September 2005) 31–36
2. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), USENIX Association (2004)
3. Anderson, D.P.: BOINC: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society (2004) 4–10
4. Radul, A.: Propagation Networks: A Flexible and Expressive Substrate for Computation. PhD thesis, Massachusetts Institute of Technology (2009)
5. Jacobi, I., Radul, A.: A RESTful messaging system for asynchronous distributed processing. In: Proceedings of the First International Workshop on RESTful Design, Raleigh, NC, USA, ACM (2010)
6. Birrell, A.D., Levin, R., Needham, R.M., Schroeder, M.D.: Grapevine: An exercise in distributed computing. Communications of the ACM **25**(4) (April 1982) 260–274
7. Braun, U., Shinnar, A.: A security model for provenance. Technical Report TR-04-06, Computer Science Group, Harvard University (2006)
8. Moreau, L., Groth, P., Miles, S., Vazquez-Salceda, J., Ibbotson, J., Jiang, S., Munroe, S., Rana, O., Schreiber, A., Tan, V., Varga, L.: The provenance of electronic data. Communications of the ACM **51**(4) (March 2008) 52–58
9. Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance collection support in the kepler scientific workflow system. In: Provenance and Annotation of Data: International Provenance and Annotation Workshop, IPAW 2006. Volume 4145/2006 of Lecture Notes in Computer Science., Chicago, IL, USA, Springer (May 2006) 118–132
10. da Cruz, S.M.S., Barros, P.M., Bisch, P.M., Campos, M.L.M., Mattoso, M.: Provenance services for distributed workflows. In: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, IEEE Computer Society (2008) 526–533