A Graph Database Approach for Efficient and Scalable Management of Simulations

Jeong-Hyon Hwang, Jeremy Birnbaum, Rohini Vabbalareddy, S. S. Ravi
Department of Computer Science
University at Albany, State University of New York
Albany, NY 12222, USA
{*jhh, jbirn, rohiniva, ravi*}*@cs.albany.edu*

Chanyeol Park
Department of Supercomputing Support
KISTI Supercomputing Center
Daejeon, 305-806, Korea
chan@kisti.re.kr

***Abstract*—Relational database technology has enabled easy and efficient management of data in a variety of applications. This technology, however, has been used for computer simulations to a limited extent. In this context, users may need to analyze the results of multiple simulation runs, which typically capture interactions between various entities in the simulated world. To these users, storing such simulation data only in the form of relational tables may seem counter-intuitive or challenging, particularly if they are not familiar with database normalization theory. Furthermore, some complex queries, including those that examine the cumulative effect of an action (e.g., disease propagation after an initial outbreak), can neither be easily expressed nor efficiently executed on traditional database systems. We propose a new database approach that aims to achieve convenient and highly efficient storage and querying of simulation data by adopting a graph data model. We also discuss new challenges that arise in this research, with a focus on language design, coordination of simulations, data storage, and query processing.**

***Keywords*-graph; simulation; database; query; storage;**

I. Overview and Challenges

Large-scale simulations that predict the behavior of a dynamic system have been key applications for high performance computing. These simulations include disease propagation [1], molecular dynamics [2], weather prediction [3], opinion formation on social networks [4], immune system simulation [5], propagation of malware over computer networks [6], and computational economics [7].

Computer programs that carry out the above simulations typically instantiate objects that represent entities (e.g., hundreds of millions of people) and then, assuming certain interactions between entities (e.g., disease transmission), update the states of these objects (e.g., from "susceptible" to "exposed") at simulated time steps. When interactions between entities are modeled as a stochastic process, results from multiple simulation runs must be analyzed together in order to obtain statistically sound results.

There have been prior approaches that store simulation data in relational databases to enable ad-hoc queries on the data [8]. These approaches, however, have inherent limitations in terms of convenience and efficiency. For example, consider collections of simulation results, each of which captures interactions or relationships among diverse entities. Storing such data only in tabular form may seem counter-intuitive or challenging to users, particularly if they are not familiar with database normalization theory. Furthermore, when relationships between a pair of entities are stored as records within a table, queries that examine the chain effect of a phenomenon (e.g., how far did the disease spread?) can neither be easily expressed nor efficiently executed. The reason for this difficulty is that relational database systems need to process a chain of n relationships using $n-1$ binary join operations, each of which adds substantial complexity in both the specification and execution of queries.

We propose a new data management approach for overcoming the above limitations. The key idea of this approach is to support a *logical schema* in which the result of each simulation run is represented as a sequence of *graphs* capturing the simulated world at different simulation steps (e.g., in a sequence of graphs $(G_{i,1},\ G_{i,2},\ \cdots,\ G_{i,m})$ for the i-th simulation run, graph $G_{i,j}$ represents the states of entities and their relationships at the end of simulation step $j \in [1, m]$). This logical schema is mainly for providing users with both a natural and an intuitive representation of the simulation results. On the other hand, the actual data storage on disk needs to be optimized for space and query efficiency and therefore can be substantially different from the logical schema. Our data storage mechanism that takes advantage of the commonalities between graphs is discussed later.

We intend to support our logical schema while enabling nesting of objects so that users can organize simulation results in their preferred way (e.g., hierarchical grouping of graphs based on simulation parameter values). We also plan to extend an object-oriented query language [9] to allow users to conveniently express queries on various combinations of graphs. One benefit of our approach is that it enables a standard way of writing simulation programs, which also creates, as explained later, opportunities for optimizing the coordination of simulations. Our approach models a simulation as a process of repeatedly creating graphs. Therefore, it allows us to write a simulation program using a loop which creates, after each iteration, a new logical graph based on the previous graph and additional interactions between entities. We are currently developing

a language which can create a new graph (similar to the `create view` command of SQL) while adding or deleting vertices or edges (similar to the `insert` and `delete` commands), or changing the states of vertices and edges (similar to the `set` command). This language also aims to support the use of custom code (i.e., user defined functions) for implementing complex state transition mechanisms, such as those that require solving differential equations [3].

We are planning to carry out the proposed research while tackling the following new challenges:

Language Design. Just as SQL allows users to conveniently create, update, and query relational tables, our language needs to support easy development and configuration of simulations as well as efficient analysis of simulation results. We are currently designing the language while incorporating features of procedural languages (to enable repeated execution of certain commands) and graph processing capabilities (e.g., identification of connected components, computation of a vertex's centrality).

Coordination of Simulations. Since we propose to write simulation programs using an SQL-like language, we intend to explore ways of coordinating simulations in an optimized fashion. As in traditional database systems, our approach will express each simulation run as a network of operators and then strive to coordinate them in a manner that minimizes the overall completion time (e.g., by automatically determining an appropriate set of servers for each simulation run). We plan to tackle new challenges that arise due to the need for handling loops in simulations and estimating the behavior of custom code which implements a complex state transition mechanism.

Data Storage. We have been constructing a scalable data storage mechanism that can store graph data on multiple servers. This technique also achieves highly efficient data storage by taking advantage of the commonalities between graphs. For example, if a vertex representing person A is in the "infected" state in graphs $G_{1,10}$ through $G_{1,30}$ and $G_{2,5}$ through $G_{2,30}$, then this technique stores that version of the vertex only once on disk and then associates it with all of the related graphs using a fast, compact index. We are currently exploring techniques for quickly optimizing the storage of graph data in the main memory before they are gradually saved on disk. This technique strives to reduce the disk I/O cost during query execution by improving data locality and to avoid slowing down ongoing simulations due to the storage of data on disk.

Query Processing. We have been developing techniques that can efficiently execute queries on multiple graphs by taking advantage of the commonalities between graphs (i.e., by sharing computations on common vertices across multiple graphs). To support a variety of analytic queries on graphs, we have been constructing operators that implement popular graph processing algorithms. We intend to develop techniques for optimizing queries, as in traditional database systems, based on estimated query execution costs.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under CAREER award IIS-1149372 and by the KISTI Supercomputing Center.

REFERENCES

[1] C. L. Barrett, K. R. Bisset, S. Eubank, X. Feng, and M. V. Marathe, "EpiSimdemics: An Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks," in *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC)*, 2008, p. 37.

[2] A. Troisi, V. Wong, and M. A. Ratner, "An Agent-Based Approach for Modeling Molecular Self-Organization," *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, vol. 102, no. 2, pp. 255–260, 2005.

[3] P. Houtekamer, L. Lefaivre, J. Derome, H. Ritchie, and H. Mitchell, "A system simulation approach to ensemble prediction," *Monthly Weather Review*, vol. 124, no. 6, pp. 1225–1242, 1996.

[4] C. J. Kuhlman, V. S. A. Kumar, and S. S. Ravi, "Controlling Opinion Bias in Online Social Networks," in *Proceedings of the International Conference on Web Science (WebSci)*, 2012, pp. 255–264.

[5] V. Folcik, G. An, and C. Orosz, "The Basic Immune Simulator: An Agent-Based Model to Study the Interactions between Innate and Adaptive Immunity," *Theoretical Biology and Medical Modelling*, vol. 4, no. 1, p. 39, Sep. 2007.

[6] K. Channakeshava, K. R. Bisset, V. S. A. Kumar, M. V. Marathe, and S. M. Yardi, "High Performance Scalable and Expressive Modeling Environment to Study Mobile Malware in Large Dynamic Networks," in *Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2011, pp. 770–781.

[7] L. Tesfatsion, "Agent-Based Computational Economics: Modeling Economies as Complex Adaptive Systems," *Information Sciences*, vol. 149, no. 4, pp. 262–268, 2003.

[8] G. Heber and J. Gray, "Supporting Finite Element Analysis with a Relational Database Backend, Part I: There is Life beyond Files," *CoRR*, vol. abs/cs/0701159, 2007.

[9] A. M. Alashqur, S. Y. W. Su, and H. Lam, "OQL: A Query Language for Manipulating Object-oriented Databases," in *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB)*, 1989, pp. 433–442.