SUPERSCALABLE MODELING SYSTEMS

Benedek IZSÓ
Advisors: István RÁTH, Dániel VARRÓ

I. Introduction

Nowadays, model-driven engineering (MDE) plays a crucial role in the development of critical embedded systems. Designer tools used for software or system development *represent data as graph*. Processing of design models involve *complex queries* expressed by graph patterns that must be evaluated on *large models*. Such operations are involved in rule-based data processing (used in expert systems), in model transformation (with model synchronization) or in the validation of design models.

In these scenarios *efficient incremental query answering* is required. This means that while a client application edits the model, match sets of (previously fixed) graph pattern queries must be available instantaneously (within milliseconds) between transactions. Transactions of typical MDE applications modify small portion of the model, while large batch-like model edits are rare. The client application must be able to efficiently load data from permanent storage, as it highly impacts initialization time.

Rete [1] is a state-saving algorithm suitable to form the basics of such a query engine. Returning matches takes less than a millisecond [2], but in exchange for speed, the cache must be created initially, it must be maintained on each modification and it must be stored. However, current implementations scale up to hundreds of thousands of elements in model size [2], in today's software development larger models come into picture, consisting ten million or more elements. To create a database that handles *large models* and offers *efficient evaluation of complex queries*, larger computers could be built (which is very costly), or as described in Sec. II. *the algorithm can be scaled up to a loosely coupled cluster*.

II. Distributed model query system

In this paper the architecture of a distributed, Rete based database is proposed, which runs on a grid of computers. It will be capable of storing large models, and evaluating complex queries efficiently.

A. Architecture

A user *application* can store or retrieve model elements, and subgraphs can be fetched matched by graph pattern queries.

The middle layer is responsible for the fast storing and retrieving of individuals, and for efficient data persistence. High performance storage technologies (like key-value stores or column-family databases) use *data shards* for the distribution of model elements. This layer should be able to send notifications to the type indexers of the distributed Rete layer about elements of their associated types.

The *distributed Rete* layer is a dataflow network, with three type of nodes. Entry nodes are the *type indexers* which sort elements (objects or relations) into sets, based on their type. These typed elements (as tokens) flow into intermediate *worker nodes* which perform operations (like filtering tokens based on constant expressions, or joining them on variables). Workers store partial results of a query in their own memory. Production nodes terminate the Rete network, consisting immediately returnable results. Connections between nodes can be *local* (within one computer) or *remote* (between computers).

The bottom layer of the database system is the *physical grid* of computers, connected by ethernet network. The database server runs on this architecture: instance data is sharded between computers, and Rete network nodes are distributed. The implementation must allow *dynamic relocation of nodes* from one computer to another and *dynamic reconfiguration of the network* to enable elastic scaling and autonomous run-time optimization.

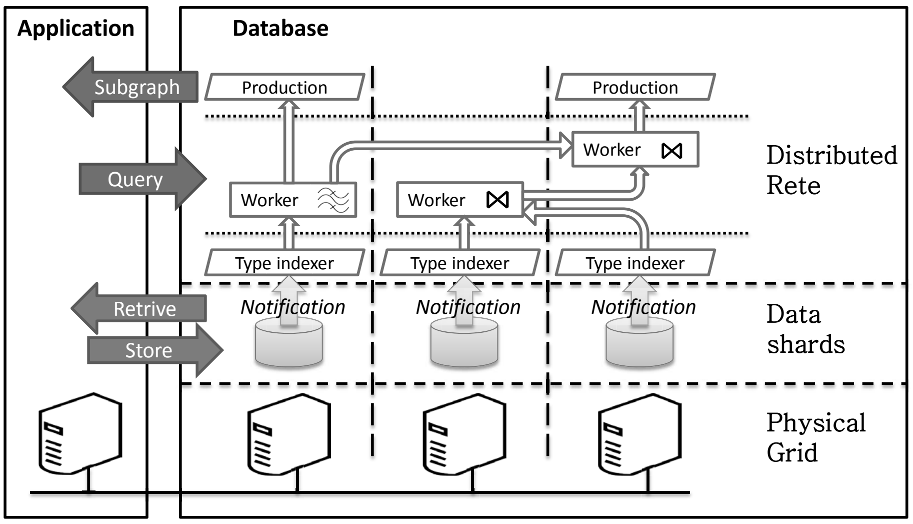

Figure 1: A distributed, Rete based model store and query system

B. Scalability considerations

The following guidelines should be implemented and evaluated during the prototype testing:

- Rete nodes should be in RAM, but the ones partitioned into one computer must use less memory, than physically available. When the limit is reached, *dynamic node relocation* can be used.
- Slow remote (ethernet) channels should be used, where source workers have the smallest memory. In that case token flow will be kept at minimum between computers.
- Order of Rete nodes highly affect evaluation performance. Initially heuristics for node placement based on the pattern or the initial data can be used, if it is known beforehand. At runtime, *dynamic reconfiguration of the network* allows to reconnect nodes in a different order.
- When matched subgraphs are small, it can be affordable to place the application and the database on different computers. However, placing them on one computer would enhance performance.
- The data sharding layer should sort model elements by their type and send notification to only one indexer, instead of broadcasting tokens and let type indexers to filter elements.
- Data on permanent storage should be in a compact and easily parseable format.

III. Related Work and Conclusion

Incremental, distributed query engines were proposed earlier. However they were only simulated [3], or they scaled up in the number of rules [4], and not in the number of model elements. Although, big data storages (usually based on MapReduce) provide fast object storing and individual retrieving for large models, query engines realized directly on these data structures don't provide fast, incremental query evaluation. That is, why this approach separates the sharded data store from the query engine.

As future work, we plan to address scalability and load balancing challenges of ad-hoc queries and query evolution, as well as the applications of this approach to collaborative modeling environments.

References

[1] C. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligences*, 19(1):17–37, 1982.

[2] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös, "Incremental evaluation of model queries over EMF models," in *Model Driven Engineering Languages and Systems*, vol. 6394 of *LNCS*. Springer, 2010.

[3] A. Acharya, M. Tambe, and A. Gupta, "Implementation of production systems on message-passing computers," *IEEE Trans. Parallel Distrib. Syst.*, 3(4):477–487, July 1992.

[4] B. Cao, J. Yin, Q. Zhang, and Y. Ye, "A mapreduce-based architecture for rule matching in production system," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pp. 790–795, Washington, DC, USA, 2010. IEEE Computer Society.