IncQuery-D: Incremental Graph Search in the Cloud

Benedek Izsó, Gábor Szárnyas, István Ráth and Dániel Varró
Fault Tolerant Systems Research Group
Department of Measurement and Information Systems
Budapest University of Technology and Economics
H-1117, Magyar Tudósok krt. 2.
Budapest, Hungary
szarnyasg@gmail.com, {izso, rath, varro}@mit.bme.hu *

ABSTRACT

Queries are the foundations of data intensive applications. In model-driven software engineering (MDE), model queries are core technologies of tools and transformations. As software models are rapidly increasing in size and complexity, traditional MDE tools frequently exhibit scalability issues that decrease productivity and increase costs. While such scalability challenges are a constantly hot topic in the database community and recent efforts of the NoSQL movement have partially addressed many shortcomings, this happened at the cost of sacrificing the powerful ad-hoc query capabilities of SQL. Unfortunately, this is a critical problem for MDE applications, as their queries can be significantly more complex than in general database applications. In this paper, we aim to address this challenge by adapting incremental graph search techniques – known from the EMF-IncQuery framework – to the distributed cloud infrastructure. IncQuery-D, our prototype system can scale up from a single-node tool to a cluster of nodes that can handle very large models and complex queries efficiently. The feasibility of our approach is supported by early experimental results.

1. INTRODUCTION

Nowadays, model-driven software engineering (MDE) plays an important role in the development processes of critical embedded systems. Advanced modeling tools provide support for a wide range of development tasks such as requirements and traceability management, system modeling, early design validation, automated code generation, model-based testing and other validation and verification tasks. With the dramatic increase in complexity that is also affecting critical embedded systems in recent years, modeling toolchains are facing scalability challenges as the size of design models constantly increases, and automated tool features become more sophisticated.

*This work was partially supported by the CERTIMOT (ERC_HU-09-01-2010-0003) project and the János Bolyai Scholarship.

BigMDE '13, June 17, 2013 Budapest, Hungary

Many scalability issues can be addressed by improving query performance. *Incremental evaluation* of model queries aims to reduce query response time by limiting the impact of model modifications to query result calculation. Such algorithms work by either (i) building a cache of interim query results and keeping it up-to-date as models change (e.g. EMF-IncQuery [4]) or (ii) applying impact analysis techniques and re-evaluating queries only in contexts that are affected by a change (e.g. the Eclipse OCL Impact Analyzer [8]). This technique has been proven to improve performance dramatically in several scenarios (e.g. on-the-fly well-formedness validation or model synchronization), at the cost of increasing memory consumption. Unfortunately, this overhead is combined with the increase in model sizes due to in-memory representation (found in state-of-the-art frameworks such as EMF [17]). Since single-computer heaps cannot grow arbitrarily (as response times degrade drastically due to garbage collection problems), memory consumption is the most significant scalability limitation.

An alternative approach to tackling MDE scalability issues is to make use of advances in persistence technology. As the majority of model-based tools uses a graph-oriented data model, recent results of the NoSQL and Linked Data movement [12, 1, 2] are straightforward candidates for adaptation to MDE purposes. Unfortunately, this idea poses difficult conceptual and technological challenges: (i) property graph databases lack strong metamodeling support and their query features are simplistic compared to MDE needs, and (ii) the underlying data representation format of semantic databases (RDF [9]) has crucial conceptual and technological differences to traditional metamodeling languages such as Ecore [17]. Additionally, while there are initial efforts to overcome the mapping issues between the MDE and Linked Data worlds [10], even the most sophisticated NoSQL storage technologies lack efficient and mature support for executing expressive queries incrementally.

We aim to address these challenges by adapting incremental graph search techniques from EMF-IncQuery to the cloud infrastructure. We introduce IncQuery-D, a prototype system based on a distributed Rete network [7] that can scale up from a single-workstation tool to a cluster to handle very large models and complex queries efficiently (Sec. 2). We carry out an initial performance evaluation in the context of on-the-fly well-formedness validation of software design models (Sec. 3), discuss related work in Sec. 4 and conclude the paper in Sec. 5.

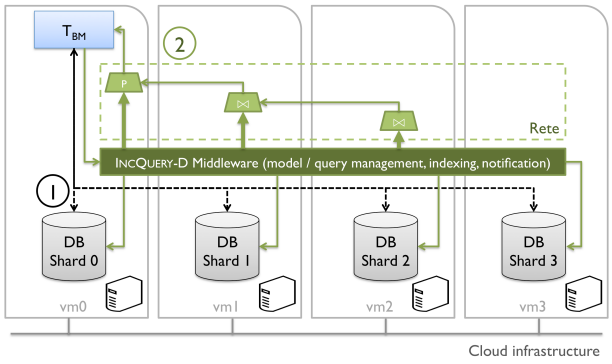

Figure 1: A distributed, Rete-based model store and query system

2. OVERVIEW OF THE APPROACH

The primary goal of our proposal is to provide a scalable architecture for executing incremental queries over large models. Our approach is based on the following foundations: (i) a distributed model storage system that (ii) supports a graph-oriented data representation format, and (iii) a graph query language (adapted from the EMF-IncQuery framework). The novel contribution of this paper is an architecture that consists of a (i) distributed model management middleware, and a (ii) distributed and stateful pattern matcher network based on the Rete algorithm. IncQuery-D provides incremental query execution by *indexing model contents* and *capturing model manipulation operations* in the middleware layer, and *propagating change tokens* along the pattern matcher network to *produce query results and query result changes* (corresponding to model manipulation transactions) efficiently. As the primary sources of memory consumption, i.e. both the indexing and intermediate Rete nodes can be distributed in a cloud infrastructure, the system is expected to scale well beyond the limitations of the traditional single workstation setup.

2.1 Architecture

The IncQuery-D architecture in an example configuration scenario is shown in Fig. 1.

Storage and middleware. In contrast to a traditional setup, where the distributed model repository (consisting of four shards in the example) is accessed on a per-node basis by a model manipulation transaction (such as a model transformation benchmark, depicted as T_{BM} in Fig. 1), IncQuery-D provides a middleware layer that offers three core services (shown in green in Fig. 1). In *distributed model management*, the primary task is to provide a *surrogate key* mechanism so that each model element in the entire distributed repository can be uniquely identified, and located within storage shards. *Model indexing* is the key to high performance model queries. As MDE primarily uses a metamodeling infrastructure, the IncQuery-D middleware maintains type-instance indexes so that all instances of a given type (both edges and graph nodes) can be enumerated quickly. Finally, *model change notifications* are required by incremental query evaluation, thus model changes are captured and their effects propagated in the form of *notification objects* (NOs). The middleware layer achieves this by providing a facade for model manipulation operations.

Conceptually, the architecture of IncQuery-D allows the usage of a wide scale of model representation formats. Our first prototype has been evaluated in the context of a low abstraction level *property graph* [15] data model, but other mainstream metamodeling and knowledge representation languages such as Ecore [17] and RDF [9] could be supported, as long as they can be mapped to an efficient and distributed storage backend (like key-value stores or column-family databases).

Distributed pattern matcher. On top of the middleware, IncQuery-D constructs a distributed and asynchronous network of communicating nodes that implement the Rete [7] algorithm (shown within the dashed region in Fig. 1). This layer is essentially a dataflow network, with two types of nodes. Change notification objects (tokens) are propagated to intermediate *worker nodes* that perform operations (like filtering tokens based on constant expressions, or performing join or antijoin operations based on their contents) and store partial (interim) query results in their own memory. In contrast, *production nodes* are terminators that provide an interface for fetching query results and also their changes. Connections between nodes can be *local* (within one host) or *remote* (when two Rete nodes are allocated to different hosts). It is important to emphasize that the database shards and Rete nodes are two distinct levels of distribution that do not directly depend upon each other.

```
1  pattern test(
2    V1:Type1, V2:Type2,
3    V3:Type3, V4:Type4) {
4    Type1.edgeType1(V1, V2);
5    // join 1
6    Type2.edgeType2(V2, V3);
7    // join 2
8    Type3.edgeType3(V3, V4);
9    // antijoin
10   neg find anti(V4, V1);
11 }
12 pattern anti(V4, V1) {
13  Type1.edgeType4(V4, V1);
14 }
```

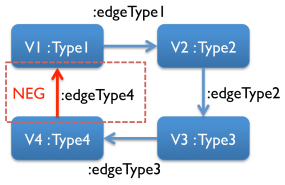

Figure 2: Example graph query

Example. An example graph query is shown in Fig. 2 (the query is intentionally domain-independent to emphasize the generalizability of the approach), as a graph pattern definition in EMF-IncQuery syntax [4] on the left and graphically on the right. This query represents a typical pattern that is used in MDE applications (such as well-formedness validation or complex model transformations), whereby a subgraph of 4 connected vertices ($V1, V2, V3, V4$) is sought after with a *negative application condition* prescribing that a typed edge (between vertices $V4$ and $V1$) must not exist. The Rete network constructed for matching this graph pattern is depicted in Fig. 3. In IncQuery-D, type-instance indexers for edge types enumerate all source and target vertices, thus the intermediate join nodes perform join operations on vertex pairs that are connected by typed edges as prescribed by the definition in Fig. 2. The join nodes all store the intermediate query results (e.g. connected $V1-V2$ and $V2-V3$ tuples in the case of the leftmost join node), and keep these caches up-to-date as change tokens arrive from the middleware (whenever model changes are performed).

Information representation and distributed operation. The Rete layer of IncQuery-D is *domain and storage agnos-*

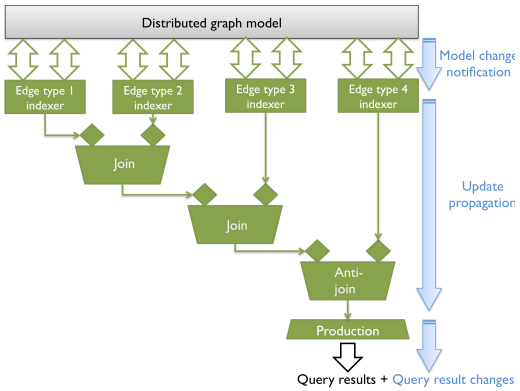

Figure 3: Rete layout

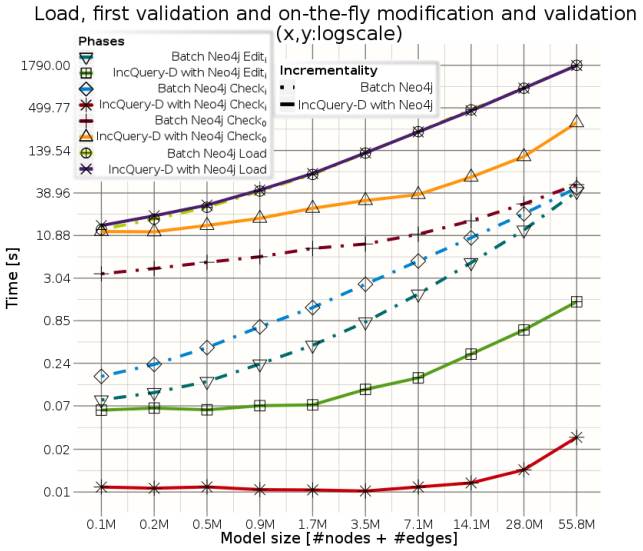

Figure 4: Benchmark results

tic as it stores only tuples constructed from model element identifiers and literals, thus it can be used independently of the model representation format (metamodeling language) of the model repository. As illustrated in Fig. 1, the Rete nodes can be allocated to different hosts in a cloud computing infrastructure (as the communication protocol supports remoting). As change propagation is asynchronous, IncQuery-D implements a *termination protocol* to ensure that the query results can be retrieved consistently with the model state after a given transaction (i.e. by signaling when the update propagation has been terminated).

2.2 Scalability considerations

For the storage layer, the most important issue from an incremental query evaluation perspective is that the indexers of the middleware should be filled as quickly as possible. This favors technologies where model sharding can be performed efficiently (i.e. with balanced shards in terms of type-instance relationships), and elementary queries (or model graph traversals) can be executed efficiently.

Achieving scalability of the distributed Rete architecture is an equally complex challenge. The overall performance of the system is influenced by a number of factors, including (i) the *layout of the Rete network* (which can be optimized depending on both query and instance model characteristics, e.g. to keep the resource requirement of intermediate join operations to a minimum), (ii) the *allocation* of Rete nodes to host computers (e.g. to optimize local resource usage, or to minimize the amount of remote network communication), and (iii) *dynamic adaptability* to changing conditions (e.g. when the model size and thus query result size grows rapidly, the Rete network may require dynamic reallocation or node sharding due to local resource limitations).

3. EVALUATION

We implemented IncQuery-D as an initial prototype to evaluate the feasibility of the approach, and to experiment with various optimization possibilities. As the storage, we used the popular graph database Neo4j [12] featuring automatic indexes and two core query technologies (Gremlin and Cypher) that were used as a low-level model access interface by our middleware layer. The prototype of the distributed middleware and Rete network were implemented in Java using Akka [18], the Scala-based toolkit for building applications based on the Actor model, since it is well-suited for asynchronous applications. The communication protocol was built on top of Akka's built-in serialization support.

Benchmark scenario. In order to measure the efficiency of model queries and manipulation operations over the distributed architecture, we designed a benchmark to measure tool response times in a well-formedness validation use case. The benchmark transaction sequence consists of four phases: (i) during the *load* phase, the serialization of the model is loaded into the database; (ii) a test query (Fig. 2) is executed ($check_0$); finally, in a cycle consisting of several repetitions, some elements are programmatically modified ($edit_i$) and the test query is re-evaluated ($check_i$). We ran the benchmark on pseudo-randomly generated instance models of growing size, each model containing about twice as many elements (vertices and edges) as the previous one and having a regular fan-out structure. As the current version of Neo4j does not have built-in support for graph sharding, the benchmark uses a manually sharded strategy where each shard contains a disjoint partition of the model.

Evaluation aspects and benchmark environment. To compare the performance characteristics of IncQuery-D to a traditional case, we defined two scenarios. The *batch* scenario uses only Neo4j to manage models and evaluate the queries (formulated in Cypher) in a parallelized way (where each shard is queried asynchronously, depicted as ① in Fig. 1). This serves as a baseline for the *incremental* scenario, which uses IncQuery-D (shown as ② in Fig. 1). For these initial experiments, the layout and allocation of the Rete network was determined manually. As the testbed, we deployed our system to a private cloud consisting of 4 virtual machines on separate host computers[1].

Results. The measurement results of our experiments are shown in Fig. 4 (aggregated from several complete sets to filter transient effects). As expected, *load* phases take about

[1]Each VM using dual 2.5 GHz Intel Xeon L5420 CPUs with 16 GBs of RAM, running on Ubuntu 12.10 64 bit with Neo4j 1.8 and Akka 2.1.2. More details at `http://incquery.net/publications/incquery-d`

the same time for both scenarios, and IncQuery-D is about half an order of magnitude slower when evaluating the query at first (*check 0* phase) due to the Rete construction overhead. However, IncQuery-D is several orders of magnitude faster during the $edit_i - check_i$ cycles, making on-the-fly query (re)evaluation feasible even for models larger than 50 million elements. Once initialized, IncQuery-D provides characteristically *linear scalability with respect to model size*, since the size of Rete network grows linearly with growing models.

4. RELATED WORK

A wide range of special languages have been developed to support *graph based* representation and querying of computer data. A class diagram-like modeling language is Ecore of the Eclipse Modeling Framework (EMF [17]), where classes, references between them and attributes of classes describe the domain. Extensive tooling helps the creation and transformation of such domain models. For EMF models, OCL is a declarative constraint description and query language that can be evaluated with the local-search based Eclipse OCL [6] engine. To address scalability issues, impact analysis tools [8, 13] have been developed as extensions or alternatives to Eclipse OCL.

Outside the Eclipse ecosystem, the Resource Description Framework (RDF [9]) is developed to support the description of instances of the semantic web, assuming sparse, ever-growing and incomplete data. Semantic models are built up from triple statements, which can be queried using the SPARQL [19] graph pattern language with tools like Sesame [2] or Virtuoso [1]. Property graphs [15] provide a more general way to describe graphs by annotating vertices and edges with key-value properties. They can be stored in graph databases like Neo4j [12] which provides the Cypher [16] query language. Even though big data storage (usually based on MapReduce) provides fast object persistence and retrieval, query engines realized directly on these data structures do not provide dedicated support for incremental query evaluation.

In the context of event-based systems, distributed evaluation engines were proposed earlier [3], scaling up in the number of rules [5] rather than in the number of data elements. As a very recent development, Rete-based caching approaches have been proposed for the processing of Linked Data (bearing the closest similarity of our approach). INSTANS [14] uses this algorithm to perform complex event processing (formulated in SPARQL) on RDF data, gathered from distributed sensors. Diamond [11] evaluates SPARQL queries on Linked Data, but it lacks an indexing middleware layer so their main challenge is efficient data traversal.

The conceptual foundations of our approach is based on EMF-IncQuery [4], a tool that evaluates graph patterns over EMF models using Rete. Up to our best knowledge, IncQuery-D is the first approach to promote distributed scalability by *distributed incremental query evaluation* in the MDE context. As IncQuery-D separates the data store from the query engine, we believe that the scalable processing of property graphs as supported by our approach can open up interesting applications outside of the MDE world.

5. CONCLUSION

We presented IncQuery-D, a novel approach to adapt distributed incremental query techniques to large and complex model driven software engineering scenarios. Our proposal is based on a distributed Rete network that is decoupled from sharded graph databases by a middleware layer, and its feasibility has been evaluated using a benchmarking scenario of on-the-fly well-formedness validation of software design models. The results are promising as they show nearly instantaneous query re-evaluation as model sizes grow well beyond 10^7 elements. For future work, we plan on providing more sophisticated automation for sharded Ecore models, and further exploring advanced optimization challenges such as dynamic reconfiguration and fault tolerance.

Acknowledgements. The authors wish to thank Gábor Bergmann for the help with the adaptation of the Rete implementation of EMF-IncQuery.

6. REFERENCES

[1] OpenLink Software: Virtuoso Universal Server. `http://virtuoso.openlinksw.com/`.

[2] Sesame: RDF API and Query Engine. `http://www.openrdf.org/`.

[3] Acharya, A. et al. Implementation of production systems on message-passing computers. *IEEE Trans. Parallel Distr. Syst.*, 3(4):477–487, July 1992.

[4] Bergmann, Gábor et al. Incremental evaluation of model queries over EMF models. In *MODELS*, volume 6394 of *LNCS*. Springer, 2010.

[5] B. Cao, J. Yin, Q. Zhang, and Y. Ye. A mapreduce-based architecture for rule matching in production system. In *CLOUDCOM*. IEEE, 2010.

[6] Eclipse MDT Project. Eclispe OCL website, 2011. `http://eclipse.org/modeling/mdt/?project=ocl`.

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

[8] T. Goldschmidt and A. Uhl. Efficient OCL impact analysis, 2011.

[9] R. C. W. Group. Resource Description Framework (RDF). `http://www.w3.org/RDF/`, 2004.

[10] G. Hillairet, F. Bertrand, J. Y. Lafaye, et al. Bridging emf applications and rdf data sources. In *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, SWESE*, 2008.

[11] Miranker, Daniel P et al. Diamond: A SPARQL query engine, for linked data based on the Rete match. *AImWD*, 2012.

[12] Neo Technology. Neo4j. `http://neo4j.org/`, 2013.

[13] A. Reder and A. Egyed. Incremental consistency checking for complex design rules and larger model changes. In *MODELS'12*. Springer-Verlag, 2012.

[14] M. Rinne. SPARQL update for complex event processing. In *ISWC'12*, volume 7650 of *LNCS*. 2012.

[15] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *CoRR*, abs/1006.2361, 2010.

[16] A. Taylor and A. Jones. Cypher Query Lang., 2012.

[17] The Eclipse Project. Eclipse Modeling Framework. `http://www.eclipse.org/emf/`.

[18] Typesafe, Inc. Akka documentation. `http://akka.io/`, 2013.

[19] W3C. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.