Optimization of Incremental Queries in the Cloud

József Makai, Gábor Szárnyas, Ákos Horváth, István Ráth, 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
jozsef.makai@inf.mit.bme.hu, {szarnyas, ahorvath, rath, varro}@mit.bme.hu

***Abstract*—Database and model queries are the foundations of data-driven applications. Their performance is of primary importance in model-driven software engineering (MDE), especially with the evergrowing complexity of software modeling projects. To address the scalability issues of traditional MDE tools, distributed frameworks are being developed – however, query optimization in a distributed context brings a whole new set of challenges, including capacity limits of individual nodes and network communication. In this paper, we aim to address the *allocation optimization challenge* in the context of distributed incremental query evaluation. Our methods are based on a combination of heuristics-based resource consumption estimations and constraint satisfaction programming. We evaluate the impact of the optimization techniques by conducting benchmark measurements.**

I. Introduction

Model-driven software engineering (MDE) plays an important role in the development processes of critical embedded systems. 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.

Many scalability issues can be addressed by improving query performance. *Incremental model queries* aim to reduce query response time by limiting the impact of model modifications to query result calculation. This technique has been proven to improve performance dramatically in several cases (e.g. on-the-fly well-formedness validation or model synchronization), at the cost of increased memory consumption.

IncQuery-D [1] aims to address the memory consumption problem by offloading memory-intensive components to a distributed architecture. IncQuery-D is a system based on a distributed Rete network [2] that can scale up to handle very large models (typically stored in distributed NoSQL or graph databases) and complex queries efficiently.

However, the introduction of the cloud and grid-like architecture introduces a whole set of query optimization challenges. In this paper, we focus on the (static) *allocation optimization problem*, which in our context means the assignment of memory-intensive distributed components to computation resources according to a strategy that is optimal in terms of overall cost, or favourable in terms of query evaluation performance. We propose to use combinatorial optimization methods based on constraint satisfaction programming, aided by heuristics to estimate i) resource usage of individual system components and ii) network traffic between distributed computation nodes. Our aim is to provide a fully automated optimization allocation mechanism for IncQuery-D.

II. Background

A. The Allocation Problem

In the context of distributed systems, allocation means the assignment of computation resources to computation tasks. The input of the allocation consists of the computation tasks and the edges between them (which represents data-flow between two tasks), the available machines with their capacities and possibly the quality of network links between machines. Computation tasks have to be assigned to processes first, as we run processes (Java Virtual Machines specifically for IncQuery-D) on machines, and the resource consumption of a process consists of the collective resource consumption of tasks assigned to this particular process. The output of the allocation is a possible mapping of processes to machines.

1) Allocation constraints: Allocation constraints are derived from the observation that certain subsets of computation tasks use so much resources together that allocating them to a set of machines without sufficient resources would cause a bottleneck in the query evaluation or even cause the failure of the computation. We call an allocation *valid*, if the constraints are satisfied. We assume that the tasks of query evaluation are memory-intensive computations, therefore the allocation constraints consist of memory constraints in our case. The goal of *allocation optimization* is to provide valid allocation optimized for certain optimization targets.

B. Optimization Targets

In this paper, we consider two possible optimization targets.

This work was partially supported by the MONDO (EU ICT-611125) project and Lendület programme (MTA-BME Lendület Cyber-Physical Systems Research Group).

1) Communication Minimization: This objective is supposed to increase the performance of query evaluation by reducing the overhead of network communication originated from the data transmission in the distributed system. This is mainly useful when time spent with network communication is significant compared to the time of local computations performed by tasks, that is true for distributed query evaluation.

2) Cost Minimization: Cost Minimization aims to reduce the monetary cost of computation infrastructure the system is operated on. This technique is useful for most distributed systems, as large systems usually require a lot of resources and thus, are expensive to operate, especially when the system runs in a public cloud infrastructure.

C. IncQuery-D *and the Rete Algorithm*

IncQuery-D [1] is a distributed, incremental model query engine that aims to address scalability issues of incremental queries over large models. The high-level architecture of the system is shown in Figure 1. IncQuery-D stores the models in distributed databases, as the models can be arbitrarily large. A typical IncQuery-D installation uses a distributed graph database such as 4store[1].

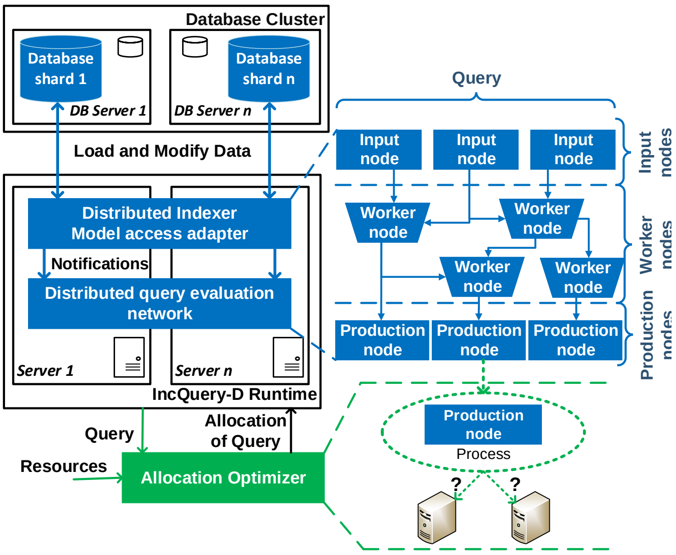


Fig. 1: Architecture of IncQuery-D.

1) The Rete Algorithm: The theoretical foundations of IncQuery-D are provided by the Rete algorithm [2]. For each query, an asynchronous dataflow network of communicating nodes is constructed, consisting of three main types of Rete nodes: i) the indexer layer of IncQuery-D consists of *input nodes* that are responsible for caching model element identifiers by type and for propagating change notifications; ii)*worker nodes* implement relational data operations (such as joins and projections) that make up a query and they also store the partial results to be propagated to their children nodes; iii) *production nodes* are responsible for storing and maintaining the final results of queries, as they reside at the endpoints of the network. Furthermore, they provide an interface for accessing query results and result deltas.

III. Allocation Optimization in IncQuery-D

A. *Formalization of the Allocation Problems*

1) Communication Minimization: For this problem, we need to represent communication intensity between processes and quality of network connection between machines in a mathematical model to approximate data transmission time. For communication intensity, we defined our own logical data unit, the *normalized tuple*, which is based on the simplification that all data inside the Rete network is represented by tuples made up of identical scalar elements (strings).

Definition 1 (Normalized tuple): Given a set of model elements organized in tuples (with the same arity), the *normalized tuple* is defined as the product of the tuple arity (number of fields) and the number of tuples.

Definition 2 (Overhead multiplier): To describe the quality of network connections, we use a simplification called *overhead multiplier*. This value represents all important parameters of the network connection as a scalar that allows us to compare the transmission times associated to identical (logical) volumes of data across various connections.

The use of *normalized tuple* and *overhead multiplier* have been validated by empirical measurements, as shown in Figure 2. We can also observe that transmission time within the same host is characteristically smaller than between different hosts.

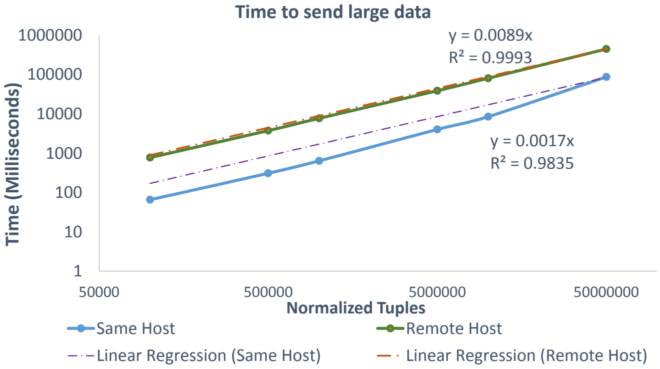


Fig. 2: Measurements of large volume data traffic.

With these notions, the *Communication Minimization* problem can be formalized as follows. The *input*:

- Memory requirements of processes: Vector with the predicted memory consumption of processes. The ith element belongs to the ith process.

$$S = [s_1, s_2, \cdots, s_n] \tag{1}$$

- Memory capacity of machines: Memory capacity of machines that can be used by the processes. The ith element belongs to the ith machine.

$$C = [c_1, c_2, \cdots, c_m] \tag{2}$$

- Communication edges between processes: The matrix E contains the communication intensity between each two

[1] http://4store.org/

processes measured in *normalized tuples*. For the ith process the ith row describes these values.

$$E_{n,n} = \begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,n} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n,1} & e_{n,2} & \cdots & e_{n,n} \end{bmatrix} \tag{3}$$

- Communication overheads between machines: The matrix O contains the overhead multipliers between machines. For the ith machine the ith row describes these values.

$$O_{m,m} = \begin{bmatrix} o_{1,1} & o_{1,2} & \cdots & o_{1,m} \\ o_{2,1} & o_{2,2} & \cdots & o_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ o_{m,1} & o_{m,2} & \cdots & o_{m,m} \end{bmatrix} \tag{4}$$

The W matrix contains the calculated communication weights between processes for valid allocations.

$$W_{n,n} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{bmatrix} \tag{5}$$

Valid allocations have to satisfy: i) for each machine, the memory capacity cannot be exceeded by the processes

$$\begin{aligned} s_1 \cdot x_{1,1} + s_2 \cdot x_{1,2} + \cdots + s_n \cdot x_{1,n} &\leq c_1 \\ s_1 \cdot x_{2,1} + s_2 \cdot x_{2,2} + \cdots + s_n \cdot x_{2,n} &\leq c_2 \\ &\vdots \\ s_1 \cdot x_{m,1} + s_2 \cdot x_{m,2} + \cdots + s_n \cdot x_{m,n} &\leq c_m \end{aligned} \tag{6}$$

ii) each process must be allocated to exactly one machine.

$$\forall j : \sum_{i=1}^{m} x_{i,j} = 1, x_{i,j} \in \{0,1\} \tag{7}$$

If two processes are placed to particular machines, the communication intensity has to be multiplied with the overhead value between the machines.

$$\forall i, j, k, l, k \neq l : x_{i,k} + x_{j,l} \geq 2 \rightarrow w_{k,l} = e_{k,l} \cdot o_{i,j} \tag{8}$$

As the objective, we minimize the sum of elements in the W matrix:

$$\text{weight} = \min \left\{ \sum_{\substack{1<i<n \\ 1<k<n}} w_{i,k} \right\} \tag{9}$$

2) Cost Minimization: The *input*:

- Memory requirements (1) and memory capacities (2) are same as for Communication Minimization.
- Cost of machines: The monetary cost of machines.

$$B = [b_1, b_2, \cdots, b_m] \tag{10}$$

The vector w contains the cost of each machine in the system, the cost of the machine if used, otherwise 0.

$$w_m = [w_1, w_2, \cdots, w_m] \tag{11}$$

Valid allocations satisfy the same constraints just as for Communication Minimization: i) (6) ii) (7).

We include the cost of a machine if and only if at least one process is placed to the machine:

$$\forall i \in \{1, \cdots, m\} : \sum_{j=1}^{n} x_{i,j} \geq 1 \leftrightarrow w_i = b_i \tag{12}$$

As the objective, we minimize the sum of elements in the w vector:

$$\text{cost} = \min \left\{ \sum_{i=1}^{m} w_i \right\}, \tag{13}$$

B. Heuristics for Approximating Resource Consumption

The lack of exact a-priori knowledge of optimization parameters is a well-known challenge in query optimization, which can be addressed by *heuristics-based* estimations to yield reasonable approximations in an efficient way. In our context, the *memory usage of processes* is a primary target for such estimations, for both problems. Furthermore, the *communication intensity* (i.e. network traffic) between processes is necessary for the Communication Minimization problem.

1) Importance of Precise Estimation: If we overestimate the memory requirement of a process and reserve much more memory than necessary, then we can not allocate as many processes to a machine as would otherwise be possible, and thus we can not reach possible better allocations. On the other hand if we underestimate the memory consumption of processes, then we risk either that the processes fail because of insufficient amount of allocated memory or we risk the violation of allocation constraints, and thus the overusage of resources. As a consequence, accurate memory estimation is required for good resource utilization and for the stable operation of the system.

For Communication Minimization, the precise estimation of communication intensity is of key importance as it could misdirect the allocation.

For the Rete algorithm that operates with collections storing static structures (tuples), the memory usage and the communication intensity of a component can be estimated using simple linear regression methods. The independent variable of these linear functions is the estimated number of model elements, measured in *normalized tuples*, stored by the Rete nodes.

2) Calculation and Propagation of Estimations: The core idea is to estimate the amount of outbound network traffic from a Rete node by the amount of data it contains, as in the *initialization phase* of the network, the entirety of this data will be transmitted. On the other hand, data storing Rete nodes must store all the data, that is propagated by their parent nodes.

This principle can be applied and propagated through the complete Rete structure using breadth-first search (BFS). Figure 3 shows an example where we marked which iteration the different estimations can be calculated in. The white circle means the memory estimation and the black circle means the communication intensity estimation. After the level of *input nodes* (for those the amount of stored and so the propagated data can be known exactly, as those propagate all their input data), the estimations for their direct children nodes can be calculated in the second iteration and so on. It is important to note that we can not calculate estimations for a node until the estimations are calculated for all its parent nodes.

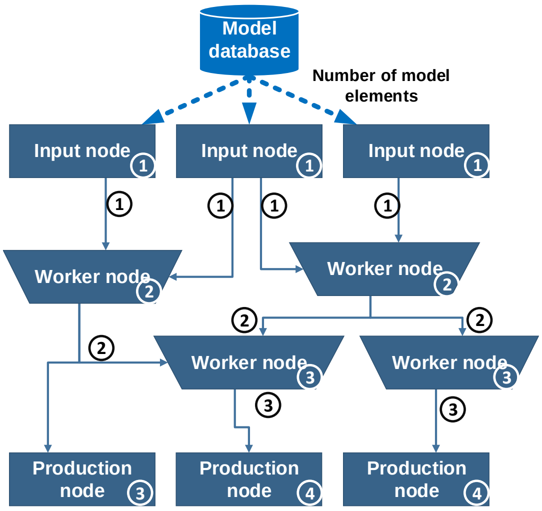


Fig. 3: Computation of heuristics for a sample Rete network.

The estimations for different types of Rete nodes are calculated following different rules that were refined in empirical experiments carried out with typical model-driven engineering workloads and models (sparse graphs). We present a few examples for these rules, for more examples see [3].

When i) joining nodes of the model through edges, we expect to have 1% of the input set size (Cartesian product of the two input sets) as result. For ii) selection nodes (which filters elements according to criteria), we expect to have 10% of input elements as result, as this is a good upper-bound estimation for usual queries.

C. Solution by Constraint Satisfaction

We proved both the Communication and Cost Minimization problems to be $\mathcal{NP}$-hard [3], therefore we propose to solve the problems by constraint satisfaction programming (CSP) [4] using the IBM CPLEX Optimizer[2] as the platform of implementation. We show how the Communication Optimization problem can be formalized with CPLEX and the OPL language[3]. We also illustrate the constraints by an example.

1) Input representation: Figure 4 shows the available machines with their memory capacities and with the *overhead multipliers* between each pair of machines.

Figure 5 shows the sample Rete network with the estimations given and already assigned to processes.

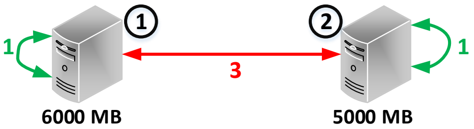


Fig. 4: Available infrastructure.

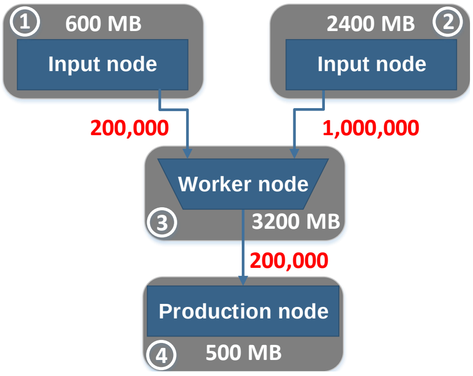


Fig. 5: Rete node and Processes with estimations.

We have an auxiliary matrix X to describe the allocation constraints: (6) and (7). If $x_{i,j} = 1$ then the jth process will be allocated to the ith machine and 0 means the opposite.

2) Memory Capacities: Constraint (6) can be stated in OPL with the following code where the *processes* array contains the memory requirement of each process and *capacities* array contains the memory capacities of machines.

```
forall (i in 1..m)
    sum(j in 1..n)
        processes[j]*x[i][j] <= capacities[i];
```

The constraint is the following for machine ①:

$$600 \cdot x_{1,1} + 2400 \cdot x_{1,2} + 3200 \cdot x_{1,3} + 500 \cdot x_{1,4} \leq 6000 \tag{14}$$

3) 1-to-1 Allocation: Constraint (7) in OPL:

```
forall (j in 1..n)
    sum(i in 1..m)
        x[i][j] == 1;
```

The constraint for process ① is:

$$x_{1,1} + x_{2,1} = 1 \tag{15}$$

4) Objective Function: The OPL code for (8) and (9) constraints, where the *edges* matrix ($n \times n$) contains the communication intensity values between every two processes and the *overheads* matrix ($m \times m$) contains the *overhead multipliers* between machines.

```
forall (i,j in 1..m)
    forall (k,l in 1..n)
        (x[i][k]+x[j][l] >= 2) => w[k][l] == edges[k][l]
            * overheads[i][j];

minimize sum(i,j in 1..n) w[i][j];
```

Constraints for the communication cost between processes ① and ③ with the possibility to be placed to any pair of

[2] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer
[3] http://www-01.ibm.com/software/commerce/optimization/modeling

machines:

$$\begin{aligned} x_{1,1} + x_{1,3} &\geq 2 \rightarrow w_{1,3} = 200000 \cdot 1 \\ x_{1,1} + x_{2,3} &\geq 2 \rightarrow w_{1,3} = 200000 \cdot 3 \\ x_{2,1} + x_{1,3} &\geq 2 \rightarrow w_{1,3} = 200000 \cdot 3 \\ x_{2,1} + x_{2,3} &\geq 2 \rightarrow w_{1,3} = 200000 \cdot 1 \end{aligned} \tag{16}$$

5) Solution: Given this input, the CPLEX Optimizer determines the overall communication intensity of the optimal solution to be 2,200,000 that is shown in Figure 6.

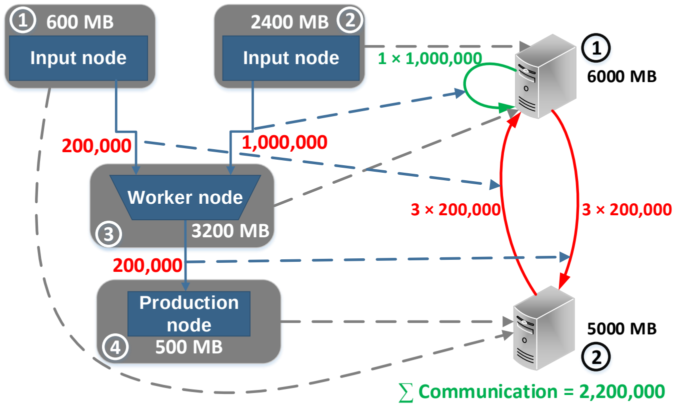


Fig. 6: Optimal allocation for the example input.

IV. Performance Impact of Optimization

To justify the beneficial effects of Communication Minimization on query performance and stability, we conducted measurements using a model validation benchmark, the Train Benchmark [5], a suite designed for evaluating the performance of graph pattern matching tools.

A. Experiments

We designed two experiments. In the *first experiment*, the impact of respecting allocation constraints is assessed by comparing an optimized configuration to a naïve setup that a) uses the default heap size for the Java Virtual Machines, and b) assigns close to the maximum RAM available to the JVMs. In the *second experiment*, we assess the impact of network traffic optimization by comparing the optimized configuration to a *non-optimized* setup. To emphasize the effect of network communication on overall performance, we configured the cloud environment to use connections with lower bandwith (10 Mbit/s) that simulate a system under load. In both experiments, we run a complex query of the Train Benchmark on instance models of increasing sizes, up to 2 million nodes and 11 million edges.

1) Hardware and software setup: The benchmark software configuration consisted of an extended Train Benchmark setup using the 4store (version 1.1.5) database in a clustered environment. We ran three virtual machines (VMs) on a private cloud powered by Apache VCL, each VM was running 64-bit Ubuntu Linux 14.04 on Intel Xeon 2.5 GHz CPUs and 8 GBs of RAM. We used Oracle 64-bit Java Virtual Machines (version 1.7.0_72). We integrated our monitoring system into the benchmark environment in order to record telemetry data at each execution stage of the benchmark. The benchmark results were automatically processed by the R scripts provided by the Train Benchmark framework.

B. Results and Analysis

1) Experiment 1: Memory optimization: As it can be seen in Figure 7, both the "default heap size" variant and the "maximum heap size" variants failed to execute successfully for the largest instance model sizes, both reporting out of heap space errors in the JVMs running one of the Rete nodes.

As the "default heap size" variant uses 1 GB heap limits, the JVM may get into a thrashing state under such loads and cause a timeout during measurement which the Train Benchmark framework registers as a failed run. Similarly, in the case of the "maximum heap size" variant, due to the lack of a reasonable upper limit, the JVMs may interfere with each other's memory allocations resulting in a runtime error.

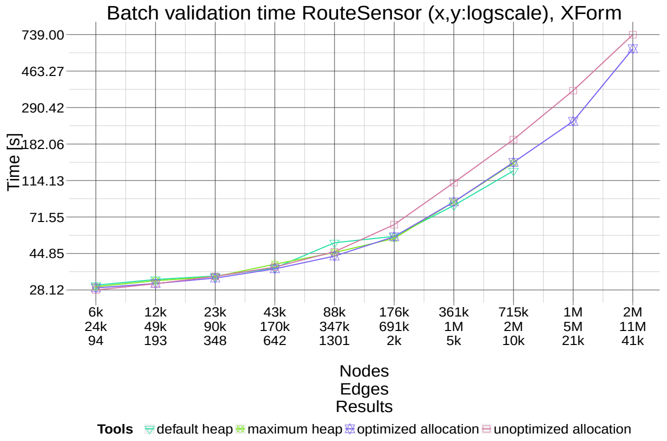


Fig. 7: Runtime of first evaluation of a query on a slow (10 Mbit/s) network.

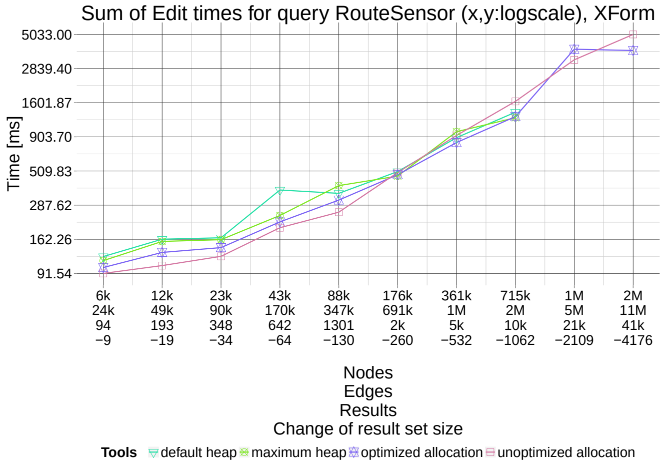
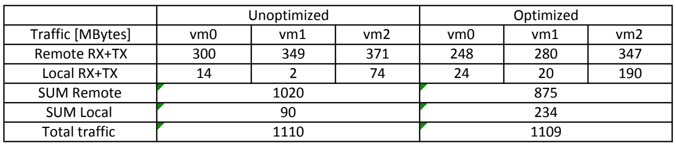


Fig. 8: Runtime of the transformation phase on a slow (10 Mbit/s) network.

2) *Experiment 2: Network optimization:* Figure 7 compares the *optimized* variant to a "non-optimized" (shown as *unoptimized*). We may observe that while the overall characteristics are similar, the *optimized* variant shows a constant-multiplier advantage, running approximately 15–20% faster than the *unoptimized* variant. Figure 8 shows the reevaluation time of a query after changes were applied to the model. In the reevaluation phase, the amount of data transmitted is significantly less than in the first evaluation phase. As the numbers are comparatively small (which is consistent with the Train Benchmark specification), the advantage of the *optimized* variant is within the measurement error range.

These observations are explained by telemetry data recorded by a monitoring system during the measurement. The network traffic measurements are summarized in Figure 9, which compares the network traffic volume recorded for the "optimized" and "unoptimized" variants. It can be seen that while the overall volume is practically equivalent, its distribution between local and remote links is characteristically different (i.e. in the "optimized" case, the overall remote volume is approximately 15% lower than in the "unoptimized" case).



	Unoptimized			Optimized		
Traffic [MBytes]	vm0	vm1	vm2	vm0	vm1	vm2
Remote RX+TX	300	349	371	248	280	347
Local RX+TX	14	2	74	24	20	190
SUM Remote	1020			875		
SUM Local	90			234		
Total traffic	1110			1109		

Fig. 9: Network traffic statistics.

C. Threats to Validity

For these experiments, we considered the following internal and external threats to validity. As transient and unknown *background load* in the cloud environment can introduce noise, we performed *several execution runs* and considered their minimum value for the result plots as a countermeasure. We strived to *avoid systematic errors* in the experiments (e.g. incorrect queries or workloads) by *cross-checking all results* with the specification of the Train Benchmark. The validity of the analysis and especially the generalizability of the results to real-world workloads has been thoroughly investigated in several academic papers on the Train Benchmark (e.g. [1], [6]). We believe that our measurements are faithful extensions of the Train Benchmark and thus the results of these previous works apply to our contributions as well.

V. Related Work

Based on an idea originating in mathematical economics, [7] proposes a characteristically different performance model compared to our domain. In these cases, the cost of communication comes from the costs of reaching data from other nodes, which is not applicable for data-driven distributed systems such as IncQuery-D. Furthermore, global allocation resource constraints are not considered. There are other models [8], where the cost can include the execution costs of tasks on particular processors besides the communication cost, which is a planned future direction of our work.

Apers et al. developed a scheduling-based optimization approach [9] for distributed queries over relational databases. In their model, relational operations are scheduled on computers. The model uses the estimated time of different operations and data transmission, as it is required for scheduling. However, as relational database queries are not incremental, memory constraints for computers are not considered.

Many different mathematical models exist for the problem according to the varying requirements of systems, but up to our best knowledge the currently proposed model of allocation was never formalized and investigated for distributed systems before, especially in the context of incremental model queries.

VI. Conclusion

We presented an approach for allocation optimization in the context of distributed systems, focusing on two different optimization targets: reducing resource usage through minimizing remote network traffic and cost reduction through minimizing the amount of computation resources required to evaluate an incremental query. Our approach is based on a mapping of the allocation optimization problems to the combinatorial optimization domain, solved with the state-of-the-art CPLEX Optimizer. We applied and evaluated this concept to the optimization of distributed incremental model queries in the context of IncQuery-D.

Our primary aim for the future is a technological adaptation of our allocation techniques to the YARN resource management framework [10]. This is directly motivated by the fact that IncQuery-D itself is moving towards a Hadoop-based implementation. In this context, experience has shown that while YARN has several built-in schedulers for resource allocation, those are not well-suited to long-running, memory-intensive jobs that are inter-related by complex structural constraints (such as the Rete network). Our long-term goal is to generalize the CPLEX-based optimization approach to a wider range of Hadoop/YARN applications.

References

[1] Szárnyas, G. et al.: IncQuery-D: A Distributed Incremental Model Query Framework in the Cloud. In: ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, 2014, Valencia, Spain, Springer (2014)

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

[3] Makai, J.: Optimization of Incremental Queries in the Cloud. Report. https://tdk.bme.hu/VIK/DownloadPaper/Inkrementalis-lekerdezesek-optimalizacioja-a (2014)

[4] Van Hentenryck, P., Simonis, H., Dincbas, M.: Constraint satisfaction using constraint logic programming. Artificial intelligence **58**(1) (1992) 113–159

[5] Szárnyas, G., Semeráth, O., Ráth, I., Varró, D.: The TTC 2015 Train Benchmark Case for Incremental Model Validation. Transformation Tool Contest, 2015

[6] Izsó, B., Szatmári, Z., Bergmann, G., Horváth, Á., Ráth, I.: Towards Precise Metrics for Predicting Graph Query Performance. In *2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 412-431, Silicon Valley, CA, USA, 2013. IEEE.

[7] Ferguson, D.F. et al.: 7. In: ECONOMIC MODELS FOR ALLOCATING RESOURCES IN COMPUTER SYSTEMS. (1996) 156–183

[8] Fernández-Baca, D.: Allocating modules to processors in a distributed system. Software Engineering, IEEE Transactions on **15**(11) (1989) 1427–1436

[9] Apers, P.M.G., Hevner, A.R., Yao, S.B.: Optimization algorithms for distributed queries. Software Engineering, IEEE Transactions on (1) (1983) 57–68

[10] Vavilapalli, Vinod Kumar, et al.: ”Apache hadoop yarn: Yet another resource negotiator.” Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013