Operator Allocation in Borealis with Integrated Sensor Network Query Processors

Wolfgang Lindner
MIT, CSAIL
The Stata Center, 32 Vassar ST, Cambridge, MA 02139, USA
wolfgang@csail.mit.edu

Holger Velke
FAU Erlangen-Nürnberg, Informatik 6
Martensstr. 3, 91058 Erlangen, Germany
sihovelk@stud.uni-erlangen.de

Klaus Meyer-Wegener
FAU Erlangen-Nürnberg, Informatik 6
Martensstr. 3, 91058 Erlangen, Germany
kmw@acm.org

Abstract

Today, Sensor Network Query Processors (SNQPs) are being integrated with Data Stream Management Systems (DSMSs). Due to the query capabilities of SNQPs, query processing can be extended from the DSMS into the sensor network. The problem then is to decide which operators should be allocated to the SNQP rather than being processed in the DSMS. This paper presents a first solution to this problem. Operators are classified as movable or not movable. Next, a QoS-based rating model, a two-stage neighborhood function browsing through the set of possible operator allocations, and an optimization algorithm using heuristics are introduced. Finally, it is shown how the operator allocation process fits into the Borealis optimization.

1. Introduction

Over the recent years Sensor Network Query Processors (SNQPs) like TinyDB [8], Cougar [9], and Directed Diffusion [6] have been developed to simplify the access to data produced by a sensor network (SN). The SNs addressed in this paper consist of battery-powered motes. The acquired data are sent to a base station via wireless communication which can be lossy. In general, the goal of such an SN is to reduce power consumption to obtain maximum lifetime.

Sensor networks are a natural source for data stream management systems (DSMSs) like STREAM [4], TelegraphCQ [5], and Borealis [2]. At MIT we developed an integration framework [3] meeting the challenges of integrating DSMSs and SNQPs. Until now the integration framework uses only a very simple optimization for distributing operators. In this paper we present a more advanced optimization approach for Borealis with integrated SNQPs. Still, it will only be the second step, since many other approaches are possible.

The goal of the Borealis optimization [2] is to satisfy the quality-of-service (QoS) requirements of the users for each query in the system. The tasks of the Borealis optimization are twofold. First, the operator order of the query diagram (all queries in the system) is optimized. Second, the distribution of operators over the sites of the distributed Borealis system is optimized. The result is a running Borealis system meeting the optimization goal.

Integrating an SNQP with Borealis adds another degree of flexibility, namely the versatility of operator allocation between Borealis and the SNQP. The Borealis query system as well as its operators descend from Aurora [1]. For the examples in this paper the following subset ($\Omega = \{F, M, U, A\}$) of Borealis operators is considered:

- **F**ilter: extended relational selection, splits an input stream into one or more output streams depending on predicates;
- **M**ap: generalized projection, performs computations on attributes of the input tuples;
- **U**nion: merges two or more streams;
- **A**ggregate: applies aggregate functions to a sliding window of input-stream tuples.

It is assumed that the SNQP supports all these operators except the union.

The Borealis system uses the box-and-arrow paradigm to specify data flow. A *box* represents a query operator while

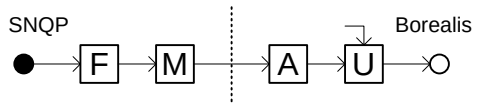

Figure 1. Ordered Query Diagram

an *arrow* represents the data flow between boxes. The following examples use this query notation.
Figure 1 shows an example of a query diagram with operators allocated in an SNQP. Optimization can be illustrated using the aggregate operator. Aggregate operators reduce the number of tuples. SNs use wireless communication for transmitting tuples from the motes to the base station. As already mentioned this communication can be lossy so the SN may lose tuples. Processing the aggregate operator in the SNQP reduces the number of transmissions and increases the lifetime. On the other hand losing an aggregate tuple has a significant impact on the quality of data since the information of all tuples contributing to the aggregate is lost. The decision of optimization is therefore a trade-off between the goal of the SNQP to increase lifetime and the goal of Borealis to satisfy user requirements (quality of data in this example).

The purpose of this paper is to present ideas for integrating an arbitrary SNQP with Borealis. It will give an overview of some the tasks to be accomplished and will show some initial solutions. A set of notions is introduced to identify the potential as well as the limitations of the optimization. In particular it is shown how to decide whether an operator should be processed in the SNQP rather than in Borealis. This has to take the different and potentially contradicting optimization goals into account. The proposed approach can also be used to integrate location-aware SNs since it could easily be extended to optimize partitions of the sensor network separately.

2. Background and Related Work

First, a short survey of our integration framework [7, 3] is given. It is used to connect Borealis with an SNQP. The proposed architecture and the constraint model are briefly described. Second, the components collaborating in the Borealis optimization are sketched.

2.1. Integration Framework

The architecture proposed in the integration framework [3] consists of wrappers and a connection layer. Each data source is covered by a specific *wrapper* which creates a uniform way to access it. The wrapper provides information about services and constraints of the SNQP. Services of the SNQP are the data it produces and the Borealis operators it supports. The data is described by its attributes and the corresponding domains. The costs for processing an operator initially allocated to the SNQP are provided for each supported operator. At runtime the wrapper also provides information about the actual system state. The *connection layer* consists of sensor proxies. A *sensor proxy* connects one Borealis site with one or more data sources covered by wrappers.

The *constraint model* [7] defines metrics as quantitative measures of the actual system state. A *metric* M represents a single aspect of the system, e.g. lifetime or latency, by a value m from the domain $dom(M)$ of the metric. *Scores* are defined on these metrics.

$$S : dom(M_1) \times dom(M_2) \times \ldots \times dom(M_n) \rightarrow X$$

A score combines several metrics into a value that indicates the quality of processing. Several scores can be collected in the *score vector* V_S. A *vector of weights* V_W is used to indicate the relative importance of each score. Additionally, a vector of soft constraints V_{SC} and a vector of hard constraints V_{HC} are defined. The *vector of hard constraints* defines a lower bound for acceptable values of each score. The values given by the *vector of soft constraints* also define a lower bound, but here smaller values are allowed; they only indicate a loss in QoS.

2.2. Borealis Optimization

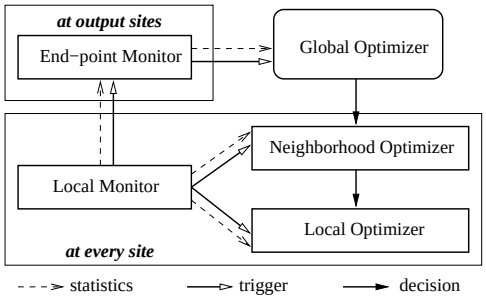

Figure 2. Borealis Optimizer Components

An overview of the Borealis optimizer components [2] is given in Figure 2. The *global optimizer* is responsible for initial query distribution and for controlling the output QoS. The *neighborhood optimizer* does load balancing between immediate neighbors. Operators can be moved between neighboring sites in the case of a communication or processing bottleneck. Each Borealis site runs a *local optimizer* dealing with e.g. load shedding, reordering of operators, and scheduling of operators and tuples. The three cooperating optimizers are triggered by monitors. *Local monitors* collect local statistics and forward them to the *endpoint monitors* which observe the QoS of output streams. They recognize QoS problems and trigger optimization.

Borealis introduces four metrics (lifetime, coverage, throughput, and latency) to measure the QoS of a stream in the system. Borealis allows QoS to be accessible at any point in the system [2].

3. Optimization

An important optimization goal is to decide which operator should be processed in the SNQP rather than in Borealis. This will be referred to in the following as *operator allocation*. It must consider the optimization goals of the participating systems, the system constraints, and the QoS requirements of the users.

3.1. Operator Order

It is important to distinguish the actual and the logical operator order. The order in which the operators are actually processed in the SNQP is not known because the SNQP does its own optimization and uses a different query representation. The logical order refers to a consistent order with respect to Borealis. It is needed to create different operator allocations, e.g. with a neighborhood function (see 3.4 below).

The operator order carries the semantics of the query diagram. Reordering can only take place between commuting operators. Without knowledge of the logical operator order in the SNQP, operators could be moved back to Borealis in an order possibly violating commutation rules.

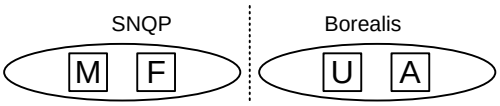

Figure 3. Sets of Allocated Operators

Figure 1 has already shown a query diagram with operators allocated to the SNQP. The sets of allocated operators resulting from this query diagram can be found in Figure 3. A particular *operator allocation* is defined by the set of operators allocated to the SNQP. Obviously, reordering operators will not create a new operator allocation if the set of operators allocated to the SNQP does not change. For example by reordering the operators F and M in the query diagram a new diagram is created but the sets of allocated operators remain the same.

3.2. Operator Categories

For optimization purposes the following operator categories are distinguished[1]:

- Operators supported by Borealis and the SNQP. These operators are categorized as *supported operator* (SO).
- Operators supported by Borealis but not by the SNQP. These are categorized as *not supported operator* (NSO).

The class SO is further sub-categorized into *movable operators* (MO) and *not movable operators* (NMO).

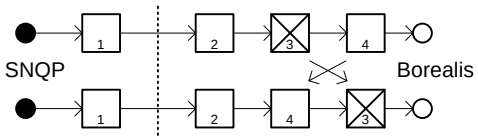

Figure 4. Scenario with MO

Figure 4 depicts a query diagram with three MOs (1, 2, 4) and an NSO (3). Clearly an NSO cannot be moved into the SNQP. Without reordering, operator 3 is a blocking operator for operator 4. But operator 4 is an MO since it commutes with operator 3. The two operators can be swapped and then it is possible to move operator 4 into the SNQP.

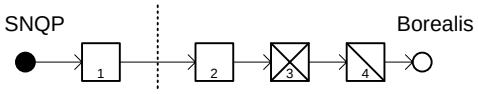

Figure 5. Scenario with NMO

Figure 5 shows a similar query diagram. The difference is that operator 3 and 4 do not commute. So operator 4, in spite of being an SO, cannot be moved into the SNQP because it is blocked by operator 3. Therefore, operator 4 is categorized as an NMO. To summarize, there are three categories of operators: NSOs, NMOs, and MOs. Only MOs must be considered for optimization purposes.

3.3. Search Space

Based on the operator categories two different search spaces are defined as operator allocations. If no operator is categorized as MO, no optimization is possible and the search spaces are empty.

The first search space consists of the operator allocations that can be produced without the overhead of reordering. The size depends on the position of the first NSO downstream of the SNQP. If the first operator is an NSO, the search space is empty and no optimization is possible. For the following example the query shown in Figure 1 is used. It is assumed that filter and map operators as well as filter

[1] In fact there are also operators which are supported by the SNQP but not by Borealis. They need not be considered here because users can only use Borealis operators to specify queries.

and aggregate operators commute. Then the search space 1 is:

$$S_1 = \{(), (F), (FM), (FMA)\}$$

The second search space considers reordering and therefore contains all operator allocations that can be produced. Its size depends on the number of MOs and their commutativity: $|S_2| \leq |MO|$. The resulting search space 2 is:

$$S_2 = \{(), (F), (M), (FM), (MA), (FMA)\}$$

Naturally, search space 1 is a subset of search space 2. It should be first choice unless no suitable operator allocation can be found, in which case search space 2 will be used. Note: Since there is no actual operator order in SNs, orders such as (MF) or (MFA) are implicitly contained in the spaces.

3.4. Neighborhood Function

A neighborhood function is defined to traverse a search space without spanning it. It derives a neighboring operator allocation from the search space based on a given operator allocation.

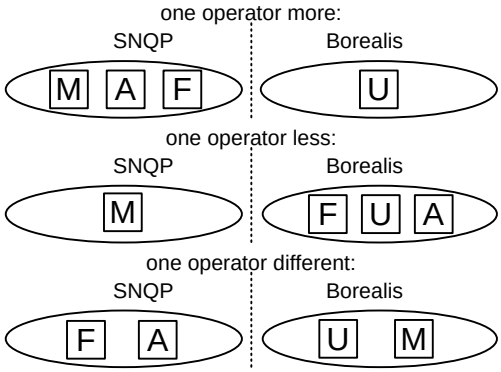

Figure 6. Neighbors of the Operator Allocation in Figure 3

Examples for neighboring operator allocations are shown in Figure 6. They are defined as operator allocations differing in just one operator, i.e. *one operator more*, *one operator less*, or *one operator different*. All neighbors of an operator allocation can be reached with one transition. Therefore, the neighborhood functions can be illustrated using a graph (see Figures 7, 8). In this graph the nodes represent the operator allocations and the edges represent the transitions. Since there are two different types of transitions the edges are labelled with **m** for moving an operator and with **r** for reordering two operators. The moving transition reaches the one-operator-more and one-operator-less neighbors whereas the reordering transition reaches the one-operator-different neighbors.

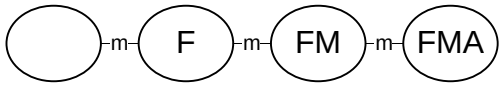

Figure 7. Neighborhood for Search Space 1

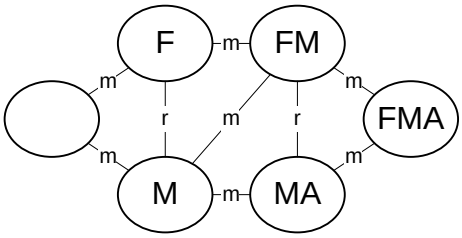

Figure 8. Neighborhood for Search Space 2

The neighborhood functions for the two search spaces are different. Figure 7 shows the neighborhood function of search space 1. In it a new operator allocation is created by moving an operator between Borealis and the SNQP. In contrast to that Figure 8 shows the neighborhood function of search space 2. Here a new operator allocation is created by first reordering and then moving an operator between Borealis and the SNQP.

3.5. Rating Model

A rating model is defined to lead the optimization through the elements of the search spaces. It is based on the QoS of the tuples crossing the border between the SNQP and Borealis. The goal is then to maximize the output QoS of the SNQP. The QoS of a tuple is monotonically decreasing on its way through Borealis. Thus the final Borealis output QoS to the user can only be less than or equal to the output QoS of the SNQP (which is also the input QoS of Borealis). The rating is only based on the operator allocation; the part of the query diagram remaining in Borealis will not be considered.

Queries are distinguished as either running or new. In case of a running query statistics on the operators and the data are available. They can be used to predict system behavior during optimization. In case of a new query however, statistics are usually not available. Hence, estimated values or operator-dependent standard values must be used. Obviously, optimization based on these values is not likely to achieve best results.

The scores introduced in Section 2 can be used to represent the considered optimization goals which can be classified as follows. First, the optimization goals of the SNQP have to be taken into account. They are the same for all queries in the SNQP. Second, the optimization goals of Borealis have to be considered. They are defined by the users who specify QoS requirements for each query in Borealis.

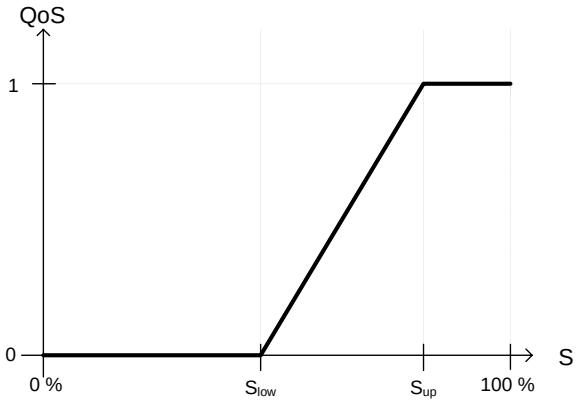

Figure 9. QoS Function for Score S

These QoS requirements refer to the output QoS of Borealis. They must be propagated upstream.

A *rating value* is calculated for each operator allocation. The optimization goal is then to *maximize* this rating value but the constraints introduced in Subsection 2.1 must also be respected. The rating-value calculation is based on the metrics provided by the wrapper. For SNQP-dependent scores these measurements can be used directly. For the Borealis-dependent scores the Borealis QoS metrics [2] must be generated based on these measurements.

If a hard constraint is violated the minimum rating value is returned for the evaluated operator allocation. This assures that such a constraint-violating operator allocation will not be chosen.

For the rating a simplified way of specifying user QoS requirements is used here. For each score two values representing the upper and lower QoS bounds are given. The lower bound S_{low} can be found in the vector of hard constraints. Scores below the lower bound are not acceptable whereas scores above the upper bound S_{up} do not improve the best QoS for the user any further. With these two values a *QoS function* is defined. The domain for it is set to $[0, 1]$ where 1 indicates best QoS and 0 indicates worst. A QoS function transforms a score into a QoS value. For example a QoS function QoS_S with the lower bound S_{low} and the upper bound S_{up} can be:

$$QoS_S = \begin{cases} 0 & \text{if } S < S_{low} \\ 1 & \text{if } S > S_{up} \\ \frac{S - S_{low}}{S_{up} - S_{low}} & \text{else} \end{cases}$$

Figure 9 shows the graph of this QoS function.
Similar to the score vector a QoS vector V_{QoS} is defined that holds all QoS values. The *normalized vector of weights* V_{WN} is the vector of weights normalized to values between 0 and 1. The QoS vector and the normalized vector of weights are used to calculate the rating value of a specific operator allocation.

For the rating the QoS values $QoS_S = V_{QoS}[i]$ can be used. So the optimization goal is defined to maximize the QoS. As rating value the simple weighted and normalized sum of the individual ratings is proposed:

$$r(V_{QoS}, V_{WN}) = \sum_{i}^{n} \frac{V_{WN}[i] V_{QoS}[i]}{n}$$

3.6. Changing Queries

If an optimal operator allocation has been found that is different from the current allocation the allocation should be changed. This means moving operators between Borealis and the SNQP. This leads to two problems. For stateful operators (e.g. aggregates) their state must migrate, too. Due to the limited space this problem will not be addressed here.

The second problem is how to change the query in the SNQP. If the SNQP does not support changing a running query the current query must be stopped and the new query must be issued. There are two approaches to do this. First, the new query can be started while the old query is still running. The old query is stopped after the new query has been fully propagated to the SNQP. This creates additional load on the SNQP. Even if optimization handles the QoS problems caused by that the additional query will worsen the situation for a certain time. Second, the old query can be stopped before the new query is started. During a certain start-up time the results of the new query will be inaccurate due to the ongoing query propagation in the SNQP. However, the additional load caused by the first approach might also lower the accuracy of results and might influence other queries in the SNQP. So it seems to be better to accept the temporary inaccuracy of results caused by the query propagation. This favors the second approach.

3.7. Allocation Optimization

For the optimization of operator allocation the algorithm of Figure 10 is proposed.

The algorithm contains two loops. In the first loop the neighborhood function of search space 1 is used. It is preferable to find an operator allocation there because no reordering is needed. If no such operator allocation is found the second loop is entered. It uses the neighborhood function of search space 2.

The purpose of the subroutines used in the algorithm is the following.

- *evaluate()* rates an operator allocation based on the rating model
- *neigh_without_reor()* retrieves a neighbor of an operator allocation from search space 1 and inserts this neighbor into the visited list

```
op_alloc := curr_op_alloc
rating := evaluate(op_alloc)

while (new_neigh_without_reor(op_alloc)) {
        neigh_alloc := neigh_without_reor(op_alloc)
        neigh_rating := evaluate neigh_alloc)
        if (neigh_rating < rating) {
                op_alloc := neigh_alloc
                rating := neigh_rating
        }
        if (stop_criteria())
                return op_alloc
}

while (new_neigh (op_alloc)) {
        neigh_alloc := neigh(op_alloc)
        neigh_rating := evaluate(neigh_alloc)
        if (neigh_rating < rating) {
                op_alloc := neigh_alloc
                rating := neigh_rating
        }
        if (stop_criteria())
                return op_alloc
}

return op_alloc
```

Figure 10. Optimization Algorithm

- *neigh()* retrieves a neighbor using search space 2 and inserts this neighbor into the visited list
- *new...()* return whether there are neighbors of an operator allocation which have not yet been visited
- *stop_criteria()* checks if a stop criterium is fulfilled

For our purposes we currently use a simple greedy algorithm as this will already improve results. More sophisticated algorithms, however, are likely to produce even better results.

4. Integration into Borealis

This Section discusses which functionality the sensor proxy and Borealis should offer. Further, it is shown how the allocation optimization fits into the Borealis optimization.

4.1. Sensor Proxy Functionality

¿From a Borealis point of view the sensor proxy hides the integrated SNQP. It acts as a virtual Borealis site and has the same interface as a regular site. In the network of Borealis sites the sensor proxy is connected with exactly one site and it runs in that site.

To fit into the Borealis optimization the sensor proxy provides Borealis with statistics and information on its capabilities. First, this means supported operators rated with initial costs which are used for initial query distribution. Second, it describes the data source provided by the SNQP which is a virtual data source for the Borealis system. The information is used by the global optimizer and by the users.

The sensor proxy further provides Borealis with information about its state. Its local monitor delivers local statistics to the Borealis endpoint monitor.

4.2. Borealis Interfaces

For interactions with the neighboring Borealis sites the sensor proxy needs an interface to:

- see the part of the query diagram running at its neighbor,
- find out whether a certain operator order can be produced,
- request reordering of the current query diagram, and
- move operators from/to the neighboring Borealis site.

For interacting with the Borealis system as a whole the sensor proxy needs:

- the ability to trigger the endpoint monitor,
- an interface to forward the monitored statistics to the endpoint monitors, and
- information on the users' QoS requirements specified for a certain point in the query diagram. Therefore, the Borealis system has to propagate the users' QoS requirements upstream.

5. Conclusion

This paper presented an initial solution to the problem of finding an optimal operator allocation in Borealis with an integrated SNQP, i.e. deciding whether to process a specific operator in the SNQP or in Borealis.

Based on the existing integration framework the collaboration of the sensor proxy with Borealis and the SNQP has been described. The proposed solution fits nicely into the Borealis optimization process. Further, an algorithm for the allocation optimization has been given. The two-staged neighborhood function and the QoS-based rating model ensure the optimization goals and the constraints of the participating systems.

For evaluation purposes we built a first prototype integrating Borealis with TinyDB.

Future work will investigate cost models in addition to the QoS-based optimization. The heuristics will be replaced by exhaustive search whenever feasible.

References

[1] D. J. Abadi et al. Aurora: A new model and architecture for data stream management. *VLDB J.*, 12(2), 2003.

[2] D. J. Abadi et al. The design of the Borealis stream processing engine. In *Proc. CIDR*, 2005.

[3] D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *Proc. VLDB*, 2004.

[4] A. Arasu et al. STREAM: The Stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1), 2003.

[5] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. CIDR*, 2003.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. MOBICOM*, 2000.

[7] W. Lindner and J. Schuler. Integrating arbitrary constraints into the query optimization process of data stream management systems. Technical report, MIT, 2006.

[8] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. ACM SIGMOD*, 2003.

[9] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proc. CIDR*, 2003.