An Architecture for Query Optimization in Sensor Networks

Ixent Galpin, Christian Y.A. Brenninkmeijer, Farhana Jabeen, Alvaro A.A. Fernandes, Norman W. Paton

School of Computer Science, University of Manchester
Manchester M13 9PL, United Kingdom
{ixent,brenninkmeijer,jabeen,alvaro,norm}@cs.man.ac.uk

***Abstract*— We present a novel sensor network query processing architecture that (a) covers all the query optimization phases that are required to map a declarative query to executable code; and (b) does so for a more expressive query language than has heretofore been supported over sensor networks. The architecture is founded on the view that a sensor network truly is a distributed computing infrastructure, albeit a very constrained one. As such, we address the problem of how to develop a comprehensive optimizer for an expressive declarative continuous query language over acquisitional streams as one of finding extensions to a classical distributed query processing architecture that contend with the peculiarities of sensor networks as an environment for distributed computing.**

I. Introduction

This paper addresses the problem of optimizing the evaluation of declarative queries over sensor networks (SNs) [1]. Viewed as a particular type of distributed computing infrastructure, SNs are constrained to an unprecedented extent, and it is from such constraints that the challenges we have addressed arise. We present the SNEEql (for Sensor NEtwork Engine query language) optimizer architecture, shown in Fig. 1, an extension of the classical *two-phase optimization* approach [2] from distributed query processing (DQP), well-established in the case of robust networks, and demonstrate that it can be adapted to be effective and efficient over SNs.

Like the two-phase optimization approach, SNEEql query optimization is decomposed into a *single-site* phase (comprising Steps 1-3, in gray boxes), and a subsequent *multi-site* phase (comprising Steps 4-7, in white, solid boxes), each of which is further decomposed into finer-grained decision-making steps. We make no specific claims regarding the novelty of the single-site phase, which generates a query execution plan (QEP) in physical-algebraic form (PAF), since the techniques used to implement these steps are well-established. As with classical DQP, for distributed execution, in the multi-site phase the PAF is broken up into fragments for evaluation on specific nodes in the network. In a SN, consideration must also be given to (i) routing, which determines the paths by which tuples travel between sites within the network, which may have a huge impact on the cost of a QEP, given that in SNs, *communication events are overly expensive*: they have energy unit costs that are typically an order of magnitude larger than the comparable cost for computing and sensing events; and (ii) the timing of QEP fragments to facilitate duty cycling (when nodes transition from being active, and in power-saving modes) and the co-ordination required between sites for wireless communications. These challenges are addressed by the introduction of Steps 4 and 7 respectively. Finally, the *Code Generation* phase grounds the execution on the concrete software and hardware platforms available in the network/computing fabric and is performed in a single step, Step 8 (in a white, dashed box). Note that optimizer decisions are informed by metadata such as the SN topology, resources on SN sites such as memory available, and also predictive cost models, which compute the worst-case upper-bounds for the output size and time taken for operations.

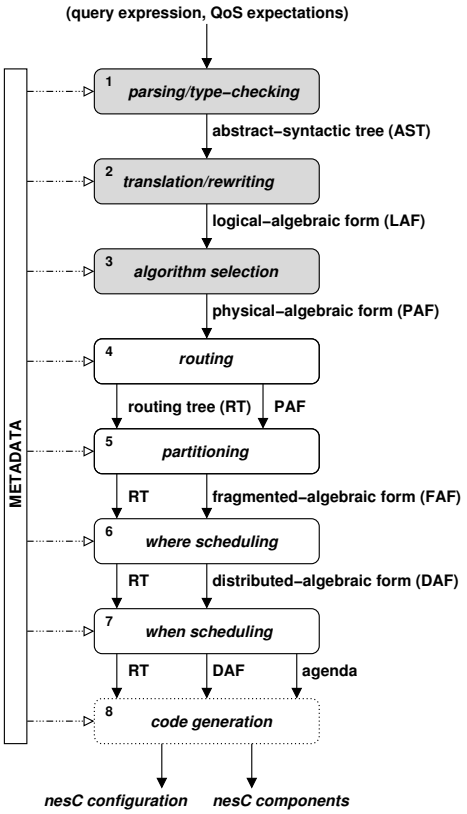

Fig. 1. SNEEql Compiler/Optimizer Stack

```
Schema metadata:
   river (id, time, rain, depth)
    Sites (5,6,7,9)
   hilltop (id, time, rain)
    Sites (4)

Query:
  SELECT  RSTREAM
          river.time, hilltop.rain, river.depth
  FROM    river[NOW],
          hilltop[AT NOW - 15 MINUTES]
  WHERE   hilltop.rain > 500
  AND     river.rain < hilltop.rain

Quality of Service Expectations:
  ACQUISITION RATE = EVERY 15 MINUTES
  MAX DELIVERY TIME = 24 HOURS
```

Fig. 2. Example Application Requirements

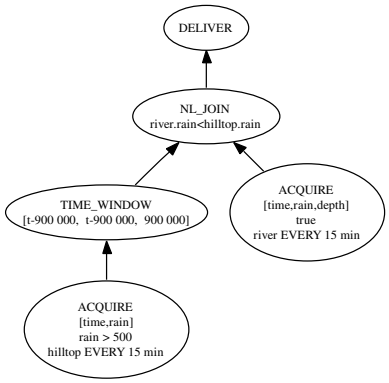

Fig. 4. Physical-Algebraic Form of query in Fig. 2

II. Architecture Components

We illustrate the step-by-step progression of the multi-site compilation of the SNEEql query with associated QoS expectations and schema metadata presented in Fig. 2, for a SN with topology depicted in Fig. 3. The syntax is inspired by CQL [3], an expressive stream querying language, but has been extended significantly for the SN context. These application requirements are inspired by (but not actually occurring in) the Crowden Brook SN deployment, an environmental monitoring system "to assess the hydro-dynamics of surface water drainage [by observing] soil moisture, temperature and rainfall on a number of vertical slope transects" [4]. The goal of this query is to obtain, every 15 minutes, timestamped readings of the current rainfall and river depth, and the rainfall at the hilltop 15 minutes previously, in cases where the rain at the hilltop is above a certain threshold, and it is currently raining less at the river than it was at the hilltop 15 minutes ago (to reduce the likelihood that any increase in river depth was caused by rain on the river itself). Note that the query, being continuous, is repeatedly evaluated at each acquisition, and that the user considers a delay of up to 24 hours to be acceptable in receiving query results. The compilation of the single-site phase results in the PAF depicted in Fig. 4.

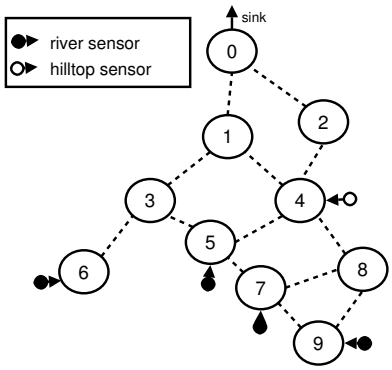

Fig. 3. Connectivities and Modalities

Routing Step 4 determines a routing tree (RT) for communication links that the data flows in the PAF can then rely on. This is achieved by computing a *steiner tree*, i.e., a tree of minimal cost derived from the network topology graph with a required set of nodes, using any additional nodes which are necessary. The resulting RT is shown in Fig. 5; it consists of the source nodes for the `river` and `hilltop` extents, the sink node where the data is to be delivered, and nodes 2 and 3 used solely to relay results.

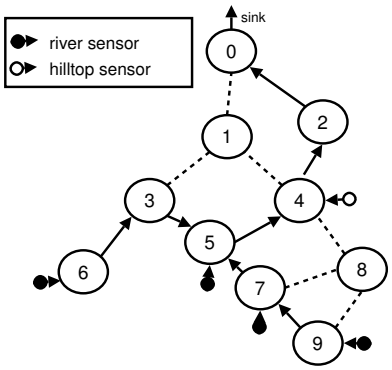

Fig. 5. Routing Tree for the Query in Fig. 4

Partitioning This step breaks up the QEP into fragments by inserting `exchange` operators [5], using semantic criteria such as operator sensitivity, and also identifying edges in the PAF with lower output sizes. Fig. 6 shows that the PAF has been partitioned into four fragments, denoted `F1`...`F4`.

Where-scheduling Step 6 decides which QEP fragments are to run on which RT nodes. This step is carried out using a heuristic algorithm that places fragments with the aim to reduce the amount of data transmitted. This step results in the distributed-algebraic form (DAF) of the query shown in Fig. 6, in which each fragment is allocated to a set of sites, listed under the corresponding fragment identifier.

When Scheduling Step 7 stipulates execution times for each fragment (seldom a specific concern in classical DQP). The approach adopted is to build an agenda that, insofar as permitted by the memory available at the site, and given the acquisition rate and the maximum expected delivery time for the query, buffers as many tuples as possible before transmitting. This

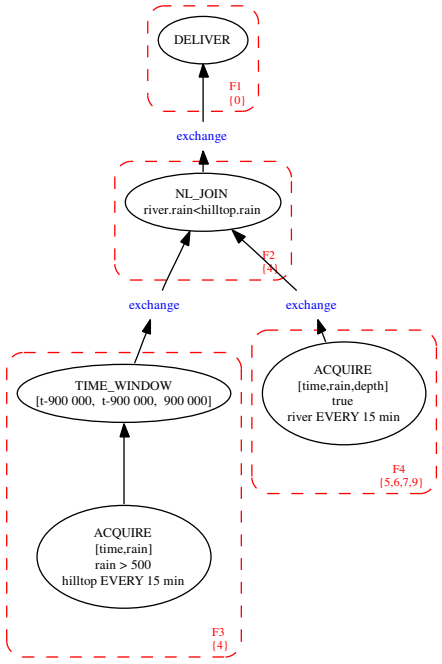

Fig. 6. Distributed-Algebraic Form of Fig. 4

Time	Sites							
	6	3	9	7	5	4	2	0
00:00:00.000	$F4_1$		$F4_1$	$F4_1$	$F4_1$	$F3_1$		
00:15:00.000	$F4_2$		$F4_2$	$F4_2$	$F4_2$	$F3_2$		
00:30:00.000	$F4_3$		$F4_3$	$F4_3$	$F4_3$	$F3_3$		
	...		...	...	...	...		
08:30:00.000	$F4_{35}$		$F4_{35}$	$F4_{35}$	$F4_{35}$	$F3_{35}$		
08:45:00.000	$F4_{36}$		$F4_{36}$	$F4_{36}$	$F4_{36}$	$F3_{36}$		
08:45:00.064	*tx3*	*rx6*	*tx7*	*rx9*				
08:45:00.658		*tx5*			*rx3*			
08:45:01.250				*tx5*	*rx7*			
08:45:02.439					*tx4*	*rx5*		
08:45:04.783						F2		
08:45:04.908						*tx2*	*rx4*	
08:45:07.095							*tx0*	*rx2*
08:45:07.158								F1

Fig. 7. Agenda for the Query Plan in Fig. 6

step is carried out using a heuristic algorithm that repeatedly schedules the acquisitional (leaf) fragments until the maximum possible level of buffering is reached, and schedules remaining tasks according to the precedence constraints implied by the DAF and RT. The aim is to be economical with respect to both the time in which a site needs to be active and the amount of radio traffic that is generated. The agenda for the example query is shown in Fig. 7. In an agenda, there is a column for each site and a row for each time when some task is started. A task is either the evaluation of a fragment (which subsumes sensing), or a communication event, denoted by *tx n* or *rx n*, i.e., respectively, tuple transmission to, or tuple reception from, site *n*. Note that in the example, leaf fragments `F3` and `F4` are repeated 36 times in each agenda evaluation, and that the agenda repeats every 9 hours. Such decisions are reached in light of the QoS specified by the user.

Code Generation Step 8 generates executable code for each site based on the distributed QEP, RT and the agenda. The current implementation of SNEEql generates nesC [6] code for execution in TinyOS [7], a component-based, event-driven runtime environment designed for wireless SNs. nesC is a C-based language for writing programs over a library of TinyOS components.

III. Conclusions

In this paper we have described the architecture of SNEEql, a SN query optimizer based on the two-phase optimization architecture prevalent in DQP. In light of the differences between SNs and robust networks, we have identified additional decision-making steps which are required. We have implemented the SNEEql query stack in 15K lines of Java. The staged decision-making approach in SNEEql offers several benefits, including: (1) The ability to pose queries using a rich, expressive language based on classical stream query languages. This is beneficial as environments in which there are less expressive languages are likely to incur greater in-network processing costs, conveying data to offline nodes for subsequent analysis; (2) The ability to schedule different workloads to different sites in the network, enabling more economical use of resources such as memory, and potentially support for heterogeneity in the SN; (3) The ability to empower the user to trade-off conflicting qualities of service such as network longevity and delivery time.

The effectiveness of the SNEEql approach of extending a DQP optimizer has been studied through an empirical evaluation, in which the performance of query execution was observed to be well-behaved under a range of circumstances, and compared well with TinyDB [7], the seminal first-generation SN database. These benefits suggest that, potentially, much can be learned from DQP optimizer architectures in the design of SN optimizer architectures.

Acknowledgment

This work is part of the DIAS-MC project funded by the UK EPSRC WINES programme under Grant EP/C014774/1. We are grateful for this support and for the insight gained from discussions with our collaborators in the project. C.Y.A. Brenninkmeijer thanks the School of Computer Science, and F. Jabeen, the government of Pakistan, for their support.

References

[1] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley and Sons, 2005.
[2] D. Kossmann, "The State of the Art in Distributed Query Processing," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 422–469, 2000.
[3] A. Arasu, S. Babu, and J. Widom, "CQL: A Language for Continuous Queries over Streams and Relations," in *DBPL*, 2003, pp. 1–19.
[4] I. W. Marshall, M. C. Price, H. Li, N. Boyd, and S. Boult, "Multi-sensor Cross Correlation for Alarm Generation in a Deployed Sensor Network," in *EuroSSC 2007*, 2007, pp. 286–299.
[5] G. Graefe, "Encapsulation of Parallelism in the Volcano Query Processing System." in *SIGMOD Conference*, 1990, pp. 102–111.
[6] D. Gay, P. Levis, J. R. von Behren, M. Welsh, E. A. Brewer, and D. E. Culler, "The nesC language: A holistic approach to networked embedded systems," in *PLDI*, 2003, pp. 1–11.
[7] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.