Adaptively brokering continuous queries for exploiting streams

José-Luis Zechinelli-Martini
Universidad de las Américas, Puebla - CENTIA
LAFMIA - CNRS UMI 3175
Exhacienda Sta. Catarina Mártir s/n
72820 San Andrés Cholula, Mexico
Email: joseluis.zechinelli@udlap.mx

Thierry Delot
LAMIH - UMR CNRS 8530
Université de Valenciennes
Le Mont Houy 59313 Valenciennes - France
Email: tdelot@univ-valenciennes.fr

Sergio Ilarri
IIS Dept, University of Zaragoza
Maria de Luna 1 Zaragoza, 50018 - Spain
Email: silarri@unizar.es

Abstract

This paper introduces an approach for adaptively brokering queries for exploiting streams produced and consumed by devices with limited computing and storage capacities. Such devices can be mobile and static and they can build networks for exploiting streams by evaluating queries in a collaborative way. Query brokering strategies are based on the notion of relevance which depends on application requirements and execution contexts used to associate priority to queries and streams. Since query brokering can be done in dynamic environments, relevance can change and thus brokering strategies can be dynamically adapted. The paper illustrates the feasibility of the approach by showing how it can be customized for three different application scenarios: a system for observing industrial environments, a system to share events (e.g., accidents, available parking spaces, emergency vehicles) in a vehicular network, and a P2P application to compare prices of products in different stores.

1. Introduction

Mobile networks allow to share data without the need of a fixed infrastructure. Yet, processing streams and queries in this context is challenging. Push approaches where data streams arriving in nodes are stored locally and processed later to obtain the answers to continuous active queries can be considered. Even in this case, important research issues remain open, such as deciding whether a data item must be stored and how to process the queries efficiently.

Current solutions exist for designing sensor networks and for implementing stream databases that can be used for building such kind of applications. In general, the approach consists in observing data (streams) and at given moments send timestamped observed data to a central server where they are processed, summarized, and historized. Given the volume of observed data, histories are refreshed according to approaches based on sliding windows or temporal based aggregation operations. The central server is able to answer queries on current histories. In order to avoid losing data, queries are continuously evaluated, as fresh data arrive.

Yet, with the evolution of technology most devices have increased their computing and storage capacities although energy saving is still an issue. So, novel query processing strategies must start exploiting devices capacities for performing as many operations as possible and adopting a just-in-time data transmission (i.e., only when absolutely necessary).

This paper introduces an approach for adaptively brokering queries for exploiting streams produced and consumed by devices with limited computing and storage capabilities. Such devices can be mobile and static and they can build networks for exploiting streams by evaluating queries in a collaborative way. Query brokering strategies are based on the notion of relevance, which depends on application requirements and execution contexts used to associate priority to queries and streams. Since query brokering can be done in

dynamic environments, relevance can change, and thus brokering strategies can be dynamically adapted.

The remainder of the paper is organized as follows. Section 2 describes the main principle of query brokering, focussing on relevance computation and query scheduling. Section 3 describes strategies for consuming streams according to their relevance. Sections 4, 5, and 6 describe respectively two representative use cases that we have proposed for evaluating our approach. Section 7 gives an overview of related work concerning query processing approaches. Finally, Section 8 concludes the paper and discusses future work.

2. Query brokering

We propose strategies for brokering queries. A query implements operations for processing streams at specific moments. A query has an associated priority and cost that can change depending on the application and its execution context. Besides, streams being pushed in devices can have different relevances depending on the priority of the query that will process them and on application requirements. In order to describe such strategies we define a query processing unit named query broker QB.

Queries will be processed by a network of query brokers. A query is received at any QB which evaluates it locally and forwards the query to its neighbours. Results are disseminated backwards and they are integrated by the QB that initially received the query.

Streams are processed in each QB and results are disseminated to other QB so that they can evaluate their own queries (see Figure 1). Streams are processed with respect to their relevance, which is determined by the state of the processing device in which the QB is deployed, the network, and application requirements. Of course, evaluation conditions and application requirements can change along time. Thus query evaluation criteria change dynamically in order to cope with such changes. In the following, we show how this is done.

2.1. Query broker

A QB is the minimal query processing unit, deployed in a processing device adding data processing functions to it. It has an associated buffer that stores data that should be aggregated or used to execute queries. A QB can compute algebra unary or binary operators: aggregation, projection, selection, union, Cartesian product, by interacting with other QBs. Of course, algebra operators implementations are modified for processing streams according to existing proposals. A query broker receives, processes, and transmits data according to queries.

Figure 2 shows the general architecture of a QB that consists in five main modules devoted to data and query processing tasks: filtering and relevance computing, stream processing, memory management.

- The Filtering Module receives and timestamps streams from a processing device. It interacts with the relevance computing module for determining whether the stream is interesting for the QB, and when and how it will be processed and stored.
- The Relevance Computing Module implements functions for determining whether a stream is relevant for the QB. The relevance function depends on the application context. It takes into consideration at least the query priority in which a tuple will participate and other QoS criteria. Sections 5 and 6 explain this component for specific application scenarios.
- The Stream Processing Module implements a scheduling strategy for evaluating the queries of a QB and performing other operations such as transmitting the current history stored in the buffer, receiving the history of another QB, and updating the history. This module also interacts with the Memory Manager for performing updates of the history.
- The Memory Manager manages the buffer of the QB that stores the history. It offers functions for inserting and flushing the history.

2.2. QB functions

The main functions of a QB are: processing streams (time stamping streams is done by its associated processing device), storing data in a history, transmitting, receiving and updating a history (H), and evaluating queries.

- The QB can store results in its history H.
- It can transmit H and receive an H' from another QB according to a dissemination protocol.
- A QB can also update its history by computing H = H $\cup$ H' where H' is received from another QB.
- Finally, a QB evaluates queries. This function is triggered at given moments of time defined by the application, for instance at given time points (at 12:00) or periodically (every hour). This possibility implements a continuous query evaluation.

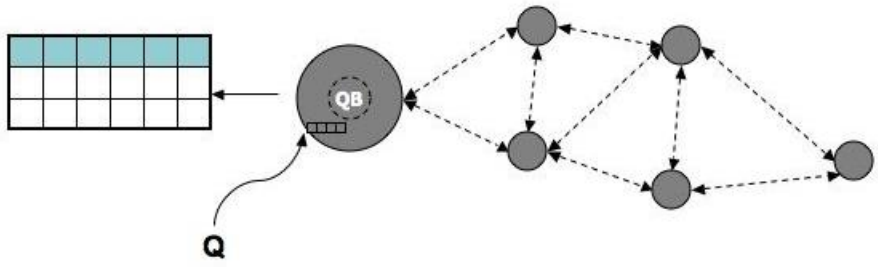

Figure 1. Query brokering

2.3. Query scheduling

Given a set of queries to be evaluated, a QB schedules or delegates them according to their priority and cost. Given the priorities of the queries P1 ... Pn: The evaluation schedule of a QB is defined as follows:

- P1: Once a new stream is detected generate a tuple and store it results in H.
- P2: If it is time to receive histories from neighbours then receive and update H (note that the task receive is triggered by an event, for example 12:00).
- P3: Otherwise compute the queries according to their priority P_i storing results according to the relevance of the resulting data.

2.4. Query brokers network

A QB interacts with other QB for sending and receiving queries and histories. Therefore, it implements an interaction protocol for receiving and sending signals and for aggregating them in a distributed manner. The communication protocol depends on the topology of the network, the capability of processing devices, their role and location.

3. Relevance based stream consumption

A QB implements strategies for consuming streams. The objective is to balance reception, processing and transmission cost within a processing device. Therefore, it combines filtering and aggregation operations. Such operations are done with respect to the notion of relevance.

3.1. Filtering

The objective of filtering is to avoid processing irrelevant data streams within a device. In a QB queries evaluation is based on sliding windows. So, filtering is sued for processing windows by managing stream histories received and computed in the QB. The idea is to build histories under a just-in-time approach by storing and exploiting only the data which are relevant for given queries' evaluations to be done by a QB. Furthermore, filtering is based on the notion of relevance computed according to a function that can use the geographic location, time and other attributes such as the identifier of the producer.

3.2. Aggregation

Each QB reads and timestamps continuously. A QB aggregates periodically monitored values using a function that is defined according to application requirements. Such values are stored in a buffer; after the aggregation the buffer is flushed and the aggregated stamped value is stored instead in order to be able to receive more data.

The objective of aggregation is to: optimize resources while processing the maximum of data flows. Indeed the problem when stream processing is embedded within a processing device with limited memory capability is that huge amount of data need to be stored in order to be able to answer queries. So, in our approach, stream aggregation is used for flushing data from the processing device buffer but avoiding the loss of information. Moreover, instead of transmitting data every time it is observed and investing energy, the processing device keeps data as long as possible until a query arrives and data must be transmitted.

In order to define our aggregation strategies, we chose a temporal model that provides granularities (i.e., seconds, minutes, hours) and data types for representing timestamps. For example, time points (10.07.2008:10:00), and intervals ([10.07.2008:10:00, 10.07.2008:12:00], 10.07.2008:10:00+2). The model also provides associated functions for transforming

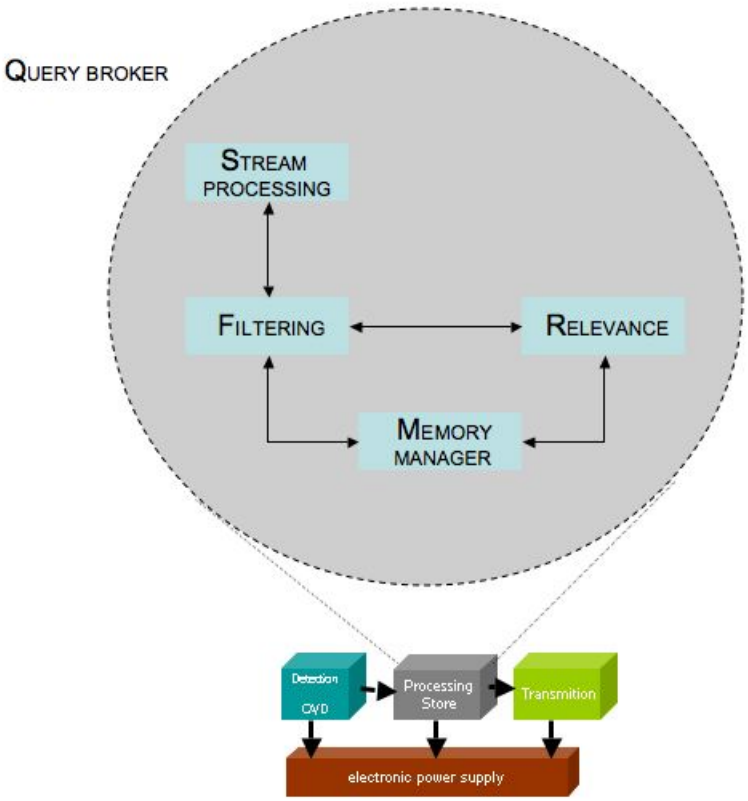

Figure 2. Query broker general architecture

data expressed under different granularities and for operating on data types. Such techniques were used in historical databases a few years ago and we adopt them in this work.

Aggregation can be configured in a QB by specifying:

- Frequency: every hour, once available, once memory/energy reaches a threshold.
- Scope: the whole buffer or a sample.
- Function: type (monotonic, strict, and strict monotonic) versus processing and energy cost.

The choice depends on and changes with respect to the characteristics and state of the processing device, and application requirements.

Classic aggregation functions have been studied in existing stream databases solutions. In general, such functions are classified as monotonic, strict monotonic, and strict, and they can have of course, different computing costs. Based on these results, we propose a very naive algorithm for managing aggregation within a processing device with energy and memory limitations. The algorithm defines an aggregation cycle guided by:

- Energy: low, medium, high.
- Function cost: energy, computational cost.
- Available memory.

The following lines give the general elements of the aggregation cycle. Given a window that specifies a history of tuples, if there is available memory then according to the available level of energy (i.e., high, medium, low), one of three aggregations are computed. Computing one or another aggregation depends on the computing cost (e.g., average is the most costly operation). Once a new aggregation has been computed, memory is flushed and the new aggregated value is stored.

Given a Window w
IF memory $\leq \varepsilon$ THEN
IF energy = high THEN value = AVG(w)
ELSE IF energy = medium THEN value = MAX(w)
ELSE IF energy = low THEN value = SUM(w)
flush(w) and store(value)

4. Use Case I: tracking freight elevators

Consider an application willing to analyze the routes of freight elevators around a warehouse for determining whether drivers follow predefined routes, their position, and to determine the current occupation state of the warehouse [1]. Each vehicle is assigned to specific drivers and, according to the occupation of the warehouse each driver has to follow a route in order to organize boxes. Several vehicles go around the warehouse but we do not consider collisions, assuming that routes have been planed in order to avoid that problem.

The manager of the warehouse wants to have a detailed control of the routes followed by each freight elevator for determining the time invested in achieving the route and to detect possible diverts that can lead to a waste of time. Specifically, s/he wants to know

- Q_1: The current position of a freight elevator (priority 1).
- Q_2: The route followed by a freight elevator until a given point in time (priority 2).
- Q_3: How many times did a freight elevator passed over a given location within a time interval (priority 3).
- Q_4: The number of boxes delivered until a certain instant (priority 2).

Therefore, an observation system has been deployed in order to observe the freight elevators, the boxes and the routes followed by drivers.

- Each fright elevator has an electronic device that identifies it.
- Each driver has an electronic ID card.
- Sensors have been installed around the warehouse in order to detect freight elevators' positions.
- Boxes have RFID stickers with an ID and the description of the content.

Additionally, given a set of freight elevators that go through routes and a set of valid routes for a freight elevator, analyze them for determining:

- If in all instants t of an interval of time the freight elevator visited all the points belonging to a valid route.
- If the trajectory within an interval [t1, t2] belongs to a valid route.
- If the freight elevator did segments of routes that do not belong to a valid one.

1. This is a simplified version of an undergoing project UDLA-Volkswagen Mexico.

4.1. Observing streams

When a freight elevator passes in front of the sensor, it identifies that fact that the *freight elevator F_1 driven by D_1 passed at point (X,Y) at time t*. Every time a sensor detects a freight elevetor, it generates the tuple $<$ FT:fi, D:dj, X:x, Y:y, T:t, B: {Bm }$>$ meaning that the freight elevator fi driven by the driver dj passed at point (x,y) at t hours with the boxes B. Composing tuples implies evaluating the first query type (Q_1). Such query must be evaluated by every QB.

Once generated, a tuple is processed by the filtering module for determining its relevance and then storing it in the history.

4.2. Evaluating queries for determining freight elevators' routes

For evaluating queries, a QB must manage its history. For example, consider Q_2 (*the route followed by a freight elevator until a given point in time*) is evaluated as follows:

1) Given a history H, let Ho be the result of selecting, all the tuples of each freight elevator F_i and ordered by their time stamp. Let the process of computing Ho be Q'_2.
2) Then, using Ho, compute a set of tuples R = {$<$P1:(x1,y1), P2:(x2,y2), T:(t1,t2), F:fi, D:dj, B:B-B2$>$} where t1 $\leq$ t2, meaning that freight elevator fi driven by dj delivered B boxes and went through the segment P1,P2. Let the process of computing R be Q'_2.
3) Finally, $Q_2 = \pi_{P1,P2,T,F,D}$ (R)

Query Q_3 computes *the times that the same freight elevator has passed in front of a QB*. Its evaluation is done using H according to the following SQL-like expression:

```
select FT, count(*) from H group by X, Y, FT
```

Similarly, Q_4 is computed by projecting the freight elevators' and the drivers' identifiers, and the boxes on R (see 1 in the previous paragraph). Finally, Q_5 (*the number of boxes delivered until a point in time*) is computed as follows:

1) Given R, let T1 be the oldest tuple in R then, total = | T1.boxes |
2) For each Ti in R where Ti $\neq$ T1, total = total - | boxes|.

4.3. Scheduling queries

Given the following priorities associated to queries:

- Priority 1: Q_1,
- Priority 2: Q_2 and Q_4,
- Priority 3: Q_3

The evaluation schedule of a QB is defined as follows:

- P1: Once a new observation is detected generate a tuple and store it results in H (Q_1).
- P2: Assume that histories are exchanged by QBs every hour. So, if it is time to receive histories from neighbours then receive and compute H'.
- P3: Otherwise compute the following queries Q_2', Q_2" [2], Q_4, Q_2, Q_3 in that order, then store or transmit results according to the relevance of the resulting data. The relevance is computed according to data required by queries and according to their timestamps. Only data computed in the last hour are considered for evaluating queries.

4.4. Brokering queries

In the freight elevator example, the observation system is based on a sensor network installed in a warehouse. It enables the detection of the freight elevators passing in strategic points of the warehouse. Since a QB is embedded in each sensor, the network can be seen as a network of QBs. Groups of QB installed in different zones of the warehouse are coordinated by a QB. Coordinators can then collaborate by exchanging data; they are, in general, used as interaction points that interact with applications using the QBs network.

5. Use Case II: sharing events in a vehicular network

In this section, we present an application scenario where vehicles exchange interesting information. First, we introduce the context and then we explain how the different query processing modules can be customized for this context.

5.1. The VESPA system

VESPA [1], [2] (Vehicular Event Sharing with a mobile P2P Architecture) is a system developed to share information about events in inter-vehicle ad hoc networks. As opposed to classical navigation systems and other existing applications, VESPA aims at sharing much more dynamic data between vehicles, both temporary and mobile events (emergency brakings, available parking spaces, etc.).

In VESPA, vehicle-to-vehicle communications (V2V) are considered. This has an important advantage, as it leads to a fully distributed mobile peer-to-peer system where no support infrastructure is required. However, it also introduces important challenges: The amount of time during which vehicles can exchange information is very limited, due to the high mobility of the vehicles and the short range of the wireless communications involved (about 200 meters). Therefore, prioritizing the data that must be exchanged, based on their relevance is a key issue in this context.

The originality of VESPA is that it takes into account any type of event (e.g., available parking spaces, obstacles in the road, information relative to the coordination of vehicles in emergency situations, etc.) in the network. VESPA is complementary to existing navigation systems supporting only static information such as points of interest (POI), since it allows drivers to be informed of ephemeral events occurring on the roads. The proposal is based on the concept of encounter probability, which is computed both to estimate the relevance of the events and to disseminate the events in the vehicular network. The encounter probability between a vehicle and an event is computed using the following formula:

$$EP = \frac{100}{\alpha \times \Delta d + \beta \times \Delta t + \gamma \times \Delta g + \zeta \times c + 1}$$

In the previous formula, the following elements are considered to compute the relevance of an event (based on temporal and spatial criteria, among other factors):

- The minimal geographical distance between the vehicle and the event over time (Δd).
- The difference between the current time and the time when the vehicle will be closest to the event (Δt).
- The difference between the time when the event is generated and the moment when the vehicle will be closest to the event (Δg).
- The angle between the vehicle's direction vector and the event's direction vector (represented by a colinearity coefficient c).

The previous factors are weighted in the formula of the encounter probability by using the penalty coefficients α, β, γ and ζ, with values ≥ 0, and where

2. Note that these queries select data from the history that are required for computing Q_2.

Δt and Δg are expressed for convenience in seconds. They are used to balance the relative importance of the Δd, Δt, Δg, and c values. The bigger the coefficient is, the more penalized the associated valued is when computing the encounter probability. For example, the greater the α value, the shorter the spatial range where the event is relevant. β and γ are used so that only the most recent information and the information about events that will be encountered very rapidly is considered. Finally, ζ is used to weigh the importance of the colinearity coefficient.

5.2. Customization of QB modules for VESPA

In the following, we indicate how the different modules proposed in our general architecture are adapted to VESPA:

- The Relevance Computing Module decides whether a data item is relevant or not. In VESPA, an event is relevant if the encounter probability is greater than a predefined relevance threshold and, at the same time, the event is retrieved by some query.
- The Filtering Module, in a periodic reevaluation of a query, is able to check quickly if it is possible that a previously-irrelevant data item stored is now relevant. For this, it examines some parameters that were set by the Relevance Computing Module for that data item and query. If so, then the relevance of the data item would need to be checked by the Relevance Computing Module again.
  Thus, when an event is classified as non-relevant by the Relevance Computing Module, information about the main cause of the non-relevance is recorded. For example, an event could be marked as not relevant because the angle between the vehicle's direction vector and the event's direction vector is greater than 45. This implies that the event will be not relevant, independently of other factors, as long as the colinearity between the vectors exceed 45. Therefore, the Filtering Module can check very quickly if this holds or not, in order to decide if the encounter probability must be recomputed or not.
- The Stream Processing Module is in charge of periodically computing the answers to the active continuous queries. Upon a new reevaluation period, for each active query, it first applies a prefiltering step by using the Filtering Module . Then, depending on the type of query evaluated, an internal Index-Based Filter is able to select potential relevant events quickly by using an index structure (on attributes that are not computed continuously). There could be several index structures used for the same query (e.g., a spatio-temporal index or both a spatial and a temporal index if time and spatial are equally important, or the type of the data item).
- The Memory Manager is in charge of managing the storage space where the information about events is stored on the cars. This module, which interacts with the previous modules, decides whether an incoming data item should be stored or not. Similarly, it decides when an old stored data item becomes irrelevant and, therefore, must be removed from the buffer. In VESPA, an event is stored if the encounter probability is greater than a predefined storage threshold. The storage threshold indicates the minimum value of the EP to consider an event received as potentially relevant for the vehicle. Additionally, stored events will be relevant for the driver if they are the target of some active continuous queries.

Different data structures are used in VESPA to reevaluate, according to the previous description of modules, the active continuous queries. For example, a table is used to store new data items that have arrived since the last refreshment cycle.

6. Use Case III: Tracking Product Prices

In this section, we present an application scenario where mobile users exchange information about price products. First, we introduce the context and then we explain how the different query processing modules can be customized for this context.

6.1. Tracking Product Prices

Customers are frequently confused about product prices, specially when they want to buy some expensive product. In particular, a potential customer frequently wonders whether in some nearby shop the same (or a similar) product will be available at a better price. The possibility to access this information as needed, on-the-fly, would be really useful. However, most companies only publish very limited information about their products on the Internet, to protect their offers from potential competitors. Therefore, a different approach is needed to retrieve this type of information.

One possible solution is to think about a P2P system that allows the users to post information about offers and search product prices. Thus, a user looking at a product (e.g., a hard drive) can enter on its PDA a partial description of the product and search for offers introduced by other users. The retrieved information

can be used to value the convenience of buying that product or go to a nearby store instead.

6.2. Customization of QB modules for the Product Price Comparison Application

In the following, we indicate how the different modules proposed in our general architecture are adapted to the scenario presented:

- The Relevance Computing Module decides whether an offer received is relevant or not. Relevance is based on a spatio-temporal criteria. Thus, offers about products located in stores far away from the user are considered irrelevant. Similarly, offers that were introduced long time ago in the system are discarded because they are probably not valid anymore. The temporal and relevance boundaries are set by the user depending on his/her preferences. For example, if a user is not willing to walk/drive more than 3 Km to find a cheaper product, this will be specified as spatial relevance criterium.
- The Filtering Module, on a reevaluation of queries, checks if some previously-irrelevant offer is now relevant. For example, if a product was not relevant because it was available on a store located more than 3 Km away from the user's location, this product may become relevant if the user has changed his/her location since then. In this case, the relevance of the information will need to be recomputed by the Relevance Computing Module.
- The Stream Processing Module is in charge of periodically computing the answers to the active continuous queries. Upon a new reevaluation period, for each active query, it first applies a prefiltering step by using the Filtering Module. Then, an Index-Based Filter is able to select relevant offers by using a spatio-temporal relevance index. Offers located in stores within a certain distance and not older than a certain amount of time are considered relevant for the user.
- The Memory Manager decides whether an incoming offer should be stored or not. Similarly, it decides when a previously stored offer becomes irrelevant and, therefore, should be removed. The relevance computed by the Relevance Computing Module is compared against a storage threshold to decide whether the offer must be stored or not. The thresholds used to decide are usually more flexible than the spatial and temporal criteria predefined by the user on his/her queries. The reason is two-fold:
  1) in this cooperative system the user is willing to store offers that are relevant for other users (even if they are not relevant for himself/herself), and
  2) the relevance of the offer for the user can change along time (e.g., as the user moves from one place to another).

A preliminary prototype of this kind of system has been implemented and some extra functionalities have been considered, such as the possibility to chat with other users in real-time to discuss about products posted in the system.

7. Related work

Querying is one of the most important functions [3], [4] for accessing and sharing data among information sources. Several query processing mechanisms have been proposed to efficiently and adaptively evaluate queries. New classes of dynamic distributed environments (e.g., peer-to-peer where peers can connect or disconnect at any time) introduce challenges for query processing. Some works add indexing structures to P2P architectures for efficiently locating interesting data and/or improving query languages expressivity ([5], [6], [7], [8], [9], [10]). Such systems rely on a global schema and often pre-determined logical network organizations and are in general poorly adapted to hybrid query processing introduced by dynamic environments where metadata are not always available.

According to [11], mobility introduces four categories of queries:

- Location Dependent Queries (LDQ) [12], which are evaluated with respect to a specific geographical point (e.g. "Find the closest hotel of my current position").
- Location Aware Queries (LAQ) which are location-free (e.g."How is the weather in Lille¿').
- Spatiotemporal Queries (STQ) [13], which include all queries that combine space and time and generally deal with moving objects.
- Nearest Neighbours Queries (NNQ) [14] which address moving objects (e.g.. "Find all cars within 100 meters of my car").

Given particular aspects of such queries, many mobile query processing techniques are based on query dissemination, that implies disseminating a query to evaluate to neighbouring devices, either directly connected or using multi-hop relaying mechanisms in order to reach more distant devices. A query can so be evaluated on those remote terminals and the possible results transmitted back to the one that issued the query. Query

dissemination techniques are well adapted to hybrid P2P environments, since they provide a mean to mobile devices to contact fixed servers, that generally manage more information, thus increasing the probability to retrieve interesting results.

8. Conclusions and future work

This paper presented an approach for brokering queries in order to exploit streams produced by devices with physical limitations and interconnected in dynamic networks. In contrast to existing approaches where costly query processing tasks are delegated to servers, given the limitations of stream producers, in our approach, queries are collaboratively processed by the devices. This allows to deal with queries in situations where no servers are available, but this implies also the need of developing strategies for processing queries under a best effort approach. Such strategies are guided by the notion of relevance, which combines dimensions such as the computing cost of tasks, the query priority, and the geographical proximity. Since these dimensions can have different importance according to the application, we propose a general query broker architecture that can be configured according to application requirements and the characteristics of the devices and network characteristics.

The use cases presented in the paper illustrate how query brokers are configured for mobile (VESPA and Price tracking) and static (freight elevators control) contexts. Configuring a query broker is not an easy task. We are willing to develop *patterns* for guiding programmers. We are also developing testbeds in order to quantitatively evaluating query brokering performance.

Acknowledgments

The authors would like to thank Ilian Elias Morales, master student of the UDLA who implemented a first version of aggregation functions on sensors. We also thank the support of a grant by the CAI-Europa XXI program.

References

[1] N. Cenerario, T. Delot, and S. Ilarri, "Dissemination of information in inter-vehicle ad hoc networks," in *Intelligent Vehicles Symposium (IV'08)*. IEEE Computer Society, June 2008, pp. 763–768.

[2] B. Defude, T. Delot, S. Ilarri, J.-L. Zechinelli, and N. Cenerario, "Data aggregation in VANETs: the VESPA approach," in *First International Workshop on Computational Transportation Science (IWCTS'08), in conjunction with the 5th Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS'08), Dublin (Ireland)*. ACM Press, July 2008.

[3] G. Wiederhold, "Mediator in the architecture of future information systems," *The IEEE Computer Magazine*, 1992.

[4] R. Domenig and K. R. Dittrich, "An overview and classification of mediated query systems," in *ACM SIGMOD Record*, 1999.

[5] S. Abiteboul, I. Manolescu, O. Benjelloun, T. Milo, B. Cautis, and N. Preda, "Lazy query evaluation for active xml," in *ACM SIGMOD International Conference*, 2004.

[6] M. Abdallah and H. Le, "Scalable range query processing for large-scale distributed database applications," in *International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 2005.

[7] M. Abdallah and E. Buyukkaya, "Efficient routing in non-uniform dhts for range query support," in *International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 2006.

[8] C. Labbe, C. Roncancio, and M. P. Villamil, "Pins: Peer-to-peer interrogation and indexing system," in *IEEE International Database Engineering and Application Symposium (IDEAS)*, 2004.

[9] M. Karnstedt, K. Sattler, M. Hauswirth, and R. Schmidt, "Similarity queries on structured data in structured overlays," in *IEEE International Workshop on Networking Meets Databases (NetDB)*, 2006.

[10] V. Papadimos, D. Maier, and K. Tufte, "Distributed query processing and catalogs for peer-to-peer systems," in *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[11] N. Marsit, A. Hameurlain, Z. Mammeri, and F. Morvan, "Query processing in mobile environments: A survey and open problems," in *International Conference on Distributed Frameworks for Multimedia Applications (DFMA)*, 2005.

[12] A. Y. Seydim, M. H. Dunham, and V. Kumar, "Location dependent query processing," in *nternational Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, 2001.

[13] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modelling and querying moving objects," in *International Conference on Data Engineering (ICDE)*, 1997.

[14] M. F. Mokbel, X. Xiong, and W. G. Aref, "Sina: Scalable incremental processing of continuous queries in spatio-temporal databases," in *ACM SIGMOD International Conference*, 2004.