Electronic Communications of the EASST
Volume 19 (2009)


# Proceedings of the Second International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2009)

## An Architecture to Support Learning-based Adaptation of Persistent Queries in Mobile Environments

Jamie Payton, Richard Souvenir, and Dingxiang Liu

6 pages

Guest Editors: Romain Rouvoy, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# An Architecture to Support Learning-based Adaptation of Persistent Queries in Mobile Environments

**Jamie Payton, Richard Souvenir, and Dingxiang Liu**

Department of Computer Science
University of North Carolina at Charlotte
{payton, souvenir, dliu}@uncc.edu

**Abstract:** Queries are frequently used by applications in dynamically formed mobile networks to discover and acquire information and services available in the surrounding environment. A number of inquiry strategies exist, each of which embodies an approach to disseminating a query and collecting results. The choice of inquiry strategy has different tradeoffs under different operating conditions. Therefore, it is beneficial to allow a query-based application to dynamically adapt its inquiry strategy to the changing environmental conditions. To promote development by non-expert domain programmers, we can automate the decision-making process associated with adapting the inquiry strategy. In this paper, we propose an architecture to support automated adaptative query processing for dynamic mobile environments. The decision-support module of our architecture relies on an instance-based learning approach to support context-aware adaptation of the inquiry strategy.

**Keywords:** adapation, context-awareness, pervasive computing, machine learning, query processing, ad hoc network

## 1 Introduction

Mobile computing devices have become more powerful, affordable, and widely available, which has led to their widespread adoption. Interest in developing new applications that take advantage of connections among devices to achieve a task continues to grow. Such applications would allow emergency responders carrying mobile devices to exchange information to coordinate triage activities; construction supervisors could connect to sensors to monitor and react to the presence of hazardous chemicals on a construction site; tourists could use mapping, transportation, and weather services available in the vicinity to generate an itinerary for sightseeing. These and other applications can be supported by mobile ad hoc networks, which are formed on-demand without any fixed network infrastructure, allowing for deployment at any time in virtually any location.

Developing applications that must acquire information and services across a highly dynamic mobile ad hoc network (MANET) can be challenging. Queries are a popular abstraction used to hide the complex network communication details associated with discovering and collecting distributed information. Often, applications must continuously monitor for changes in information or conditions in the environment. For example, a construction supervisor's application may monitor for the presence of a dangerous leak to ensure the safety of workers, or a tourist may wish to take advantage of any audio guide services she encounters as she wanders through a his-

torical area. In such cases, a *persistent query*, which provides continuous reporting of relevant state changes, can support application development.

Employing an ideal persistent query that reports all state changes in a rapidly changing environment is expensive in terms of message overhead and resource consumption. Instead, we can approximate a persistent query as a sequence of one-time queries (i.e., queries that are issued and evaluted once). The persistent query's inquiry strategy, which defines where, when, and how one-time queries are issued, has different tradeoffs and its suitability depends on the operating conditions. Therefore, the notion of an adaptive persistent query [RJPR08] has been introduced to allow the inquiry strategy to be dynamically adjusted.

In previous work, the task of determining *how* to adapt the inquiry strategy was left to the application programmer. Here, we focus on automating the process of adaptation for adaptive persistent queries to simplify the programming task. We present an architecture to support persistent query-based application development in MANETs. Our approach to persistent query adaptation is independent of application-specific details; the adaptation decision is based on maximizing the reflection of changes in available data while minimizing the overhead associated with the query's execution. We use an instance-based learning approach to estimate the quality of the persistent query's result as a function of the changing state of the environment.

## 2 Background on Adaptive Continuous Queries

Researchers have begun to develop a formal framework to support expression of adaptive persistent queries and to support reasoning about their execution [RJPR08]. The framework defines a persistent query as a sequence of one-time queries. This sequence is controlled by the specification of an *inquiry strategy*, which defines: 1) the frequency with which one-time queries are issued, 2) the *inquiry mode* (i.e., the protocol used to implement a one-time query), and 3) inquiry mode parameters. To create a persistent query result, the results from the component one-time queries can be coalesced.

Using the formal framework as a foundation, a set of inquiry modes have been identified and formalized [RJPR09]. *Flooding* queries are commonly used in mobile settings [IGE+03, JMB01, PR99] to acquire information from all network nodes. In a flooding query, the sending node broadcasts the query to all of its one-hop neighbors; every recipient of the query will rebroadcast the query to its one-hop neighbors until the network boundary is reached. This approach can be very expensive in terms of message overhead and resource consumption [NTCS99], but yields the most information about the state of the environment. In a static network (no value or connectivity changes), successive one-time flooding queries should give exactly the same results. *Location-based* queries operate in a similar fashion, but can reduce overhead costs by targeting a specified region of the network. *Probabilistic* queries can also reduce overhead; a parameter is provided for query propagation to randomly select the subset of neighbors that will receive (and probabilistically propagate) the query. Similarly, *random sampling* queries propagate the query to all one-hop neighbors but randomly select nodes to execute and reply to the query.

Different inquiry strategies are appropriate in different situations. For example, a construction site worker may prolong the lifetime of a network when conditions are normal by randomly sampling for safety information; if the query returns a dangerous chemical reading, a flooding

query is needed to acquire as much information as possible about a possible leak regardless of the cost. In the formal framework, a persistent query's intermediate result (i.e., the coalesced history of one-time query results) can be assessed to determine the suitability of the inquiry strategy. In the framework, a persistent query's *introspection strategy* is a function that defines the "quality" of the query result. Adaptation of the inquiry strategy is based on the evaluation of the introspection strategy's metric. The goal of this paper is to provide automated support for the processes involved in deciding *when* and *how* to adapt persistent queries. In Section 3, we present an architecture that allows for *learning* a general introspection metric that minimizes the overhead associated with the query while maximizing the quality of the query result.

# 3 Architecture Description

Figure 1 illustrates our architecture for supporting adaptation in persistent query processing. Although omitted from the diagram, we assume every network node has low-level functions which support exchange of messages, interaction with local sensors, and maintenance of an up-to-date list of its own one-hop neighbors. The One-time Query module supports an inquiry mode through the use of the appropriate conjunction of query propagation schemes implemented in the Query Propagator (QP) and reply processing schemes implemented in the Reply Processor (RP). Both the QP and the RP units make use of the discovery module to send query-related messages over network links to the appropriate one-hop neighbors. Applications that require knowledge about the changing state of the environment can be developed using the Adaptive Persistent Query (APQ) module; a persistent query is implemented as a series of one-time queries, which are executed by the One-time Query module. Applications that wish to have tight control on the conditions that trigger adaptation of a persistent query and the manner in which the inquiry strategy is adapted can specify these conditions within a code fragment that is given to the architecture and automatically deployed. The AppAdapt module is responsible for managing the user-specified adaptation, but is not the focus of this paper. Instead, we focus on the further development of the AutoAdapt module, which uses a learned function $F$ (shown as an oval) that captures the "quality" of the query result under a set of environmental conditions to automatically adapt the inquiry strategy. Below, we describe how we learn the function $F$ and how the AutoAdapt module uses this function to adapt a persistent query's execution.

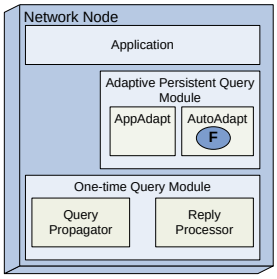


Figure 1: Adaptive Persistent Query Architecture

## 3.1 Learning Problem Overview

It is desirable for our persistent query (comprised of a sequence of one-time queries) to closely approximate the ideal persistent query. In other words, we want our persistent query to capture as much information as possible about the changes that occur in the environment. We could simply use an inquiry strategy with a high frequency of issue and an inquiry mode that collected infor-

mation from all nodes (e.g., a flooding query), but message overhead and resource consumption is a concern in networks of mobile devices. Therefore, we want to learn when and how to adapt the inquiry strategy to balance the tradeoff between the "quality" of result (i.e., how well the approximate persistent query reflects the ideal query) and the cost of the query.

We frame this as a numerical optimization problem where the goal is to maximize the difference, $F - \alpha C$, where $F$ is a function that defines this "quality" value, $C$ is the cost of execution using a particular inquiry mode, and $\alpha$ is a scaling constant. $F(R, R^*, s)$, where $R$ is the set of results (i.e., representations of responding hosts) returned by a one-time query with inquiry strategy $s$ and $R^*$ is the ideal set of reachable results reachable using strategy $s$, is defined as $|R|/|R^*|$ if $s$ is a flooding strategy and $|R|/(|R^*| * p)$ if $s$ is a probabilistic strategy with probability $p$. That is, $F$ represents the percent of hosts that, ideally, should have responded.

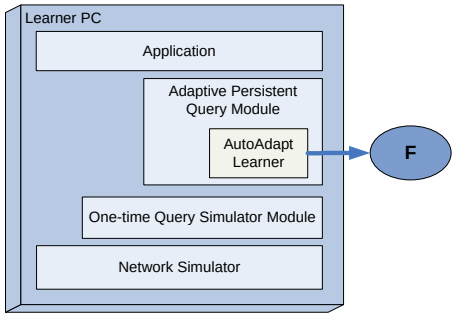


Figure 2: Architecture for the Offline Learner

In reality, however, we cannot compute $R^*$, and therefore we cannot compute $F$. So, our approach is to approximate $F$ using the history of previous query results. It is the task of the AutoAdaptLearner in Figure 2 to learn a function $\hat{F}$ that approximates $F$ to estimate how well we use one-time queries to implement a continuous query. To do so, we use an instance-based learning approach to learn $\hat{F}$ and modify our optimization function to $\hat{F} - \alpha C$; this allows us to learn a general function without requiring the overhead associated with an online learning approach.

## 3.2 Learning the Quality Function

To learn our quality function, $\hat{F}$, which is based on the history of query results, we need to define how to compute the difference between sets of results for successive one-time queries. Each query collects a node's data as well as local properties of the node at the time that the query executed. The result of the $i^{th}$ one-time query, $R_i$, is a set of hosts that responded to the query. We define a host, $h$, as a tuple, $(\iota, \nu, \lambda, \omega, \varepsilon)$ where $\iota$ is a unique node identifier, $\nu$ is a data value, $\lambda$ is the node's location, $\omega$ is the node's velocity, and $\varepsilon$ is a measure of the remaining energy. Our approximation of quality of the query's execution, then, is defined as a function of the difference between the query results for $R_i$ and $R_{i-1}$.

As a first step, we propose simple metrics for computing the difference between successive query results; we expect that more complex metrics will be developed and can be applied using our method. In our metrics, we compare values of nodes that provided results in successively issued one-time queries. We define $M$ as the set of nodes that contribute results both in $R_i$ and $R_{i-1}$: $M = \{h_x \in R_i \bigcap h_y \in R_{i-1} : h_x.\iota = h_y.\iota\}$. For each component of the host tuple (excluding the ID), we use the variance of the values in $M$ as input to $\hat{F}$. This gives us $\sigma(\nu_M)$, $\sigma(\lambda_M)$, $\sigma(\omega_M)$, and $\sigma(\varepsilon_M)$ as parameters. In addition, we calcuate a matching score $m = \frac{|M|}{|R_{i-1}|}$ which penalizes for differences between the sets of responding nodes. Other measurements, such as higher-order statistics or domain-specific metrics, may prove to be useful.

To learn $\hat{F}$ by example, we run a series of one-time queries in simulation. For each query, we compute $m$, $\sigma(v_M)$, $\sigma(\lambda_M)$, $\sigma(\omega_M)$, and $\sigma(\varepsilon_M)$. We also compute the value of $F$ using the "oracle" view of the network provided by the simulator. To compute the approximation $\hat{F}$, we use generalized radial basis functions (GRBF) [PG90], which estimate functions of the form: $\hat{F}(\vec{x},s) = \sum_{j=1}^{C} w_j \phi(||\vec{x} - \vec{z}_j||_2) + b$, where $F : \mathscr{R}^5 * \mathscr{Z} \rightarrow \mathscr{R}$, $\vec{x}$ is a vector whose components are $m$, $\sigma(v_M)$, $\sigma(\lambda_M)$, $\sigma(\omega_M)$, and $\sigma(\varepsilon_M)$, $s$ is an inquiry strategy, $b$ is a bias vector, $w_j$ is the real-valued weight of kernel center $\vec{z}_j$, for $j \in 1,2,...,C$, and $\phi$ is a real-valued *basis function*. In our algorithm, we choose the Gaussian function for $\phi$: $\phi(r) = e^{-r^2/\sigma_w^2}$, where $\sigma_w$ is the average intra-center distance.

The learned function $\hat{F}$ is incorporated into the AutoAdapt module that is deployed on the network nodes. The AutoAdapt module makes decisions regarding how to adapt the inquiry strategy to best suit the conditions of the environment by finding an inquiry mode and frequency to maximize $\hat{F} - \alpha C$, where $C$ is the cost of issuing a query using a particular inquiry strategy.

## 4 Related Work

Query processing for streaming data has been an active area of research in the sensor networks community [IGE+03, MFHH03, MFHH02], and several query processing systems provide some version of persistent queries, typically implemented as a sequence of one-time queries. Previous work noted the need for adaptive persistent queries and strategies for deciding when to adapt inquiry strategies [RJPR09, RJPR08]. Other work has also studied the problem of *how* to adapt a probabilistic persistent query's execution [RJ08] by using application-specified data quality metrics and thresholds to adapt the value of the probability parameter that influences propagation of query-related messages, while we propose the use of a machine learning algorithm to determine how to adapt the inquiry strategy. Our work uses a simple set of metrics to capture the quality of a persistent query's execution and to determine adaptation; more complex metrics, like measures of consistency [PJR07], may be appropriate.

## 5 Conclusions and Future Work

In this paper, we have taken a first step towards developing an architecture to support applications that use adaptive persistent queries in MANETs. Here, we focused on automating the process of deciding *when* and *how* to adapt a persistent query to deliver results with the most suitable degree of quality given the conditions of the environment. Currently, an implementation of the offline learner is in progress using the Omnet++ network simulator [Var], its mobility framework [LWK], and battery module extension [For] to implement and collect data about the execution of persistent queries; we apply the radial basis function implementation in the Weka machine learning workbench [HDW94] to the collected data. To determine if this approach is truly useful, we plan to perform a thorough evaluation of a full implementation and will compare performance measurements to those of traditional persistent queries. Future work also includes further investigation of properties of the environment that can impact the quality and cost of a persistent query's execution, as well as more careful study of metrics that define meaningful differences between sets of query results.

## Bibliography

[For] A. Forster. Downloads Web Page. http://www.inf.unisi.ch/phd/foerster/downloads.html.

[HDW94] G. Holmes, A. Donkin, I. Witten. Weka: A machine learning workbench. In *Proc. of the 2nd Austrailia and New Zealand Conference on Intelligent Info. Systems*. 1994.

[IGE+03] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heideman, F. Silva. Directed Diffusion for Wireless Sensor Networking. *IEEE/ACM Trans. on Networking* 11(1):2–16, February 2003.

[JMB01] D. B. Johnson, D. A. Maltz, J. Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. *Ad Hoc Networking* 1:139–172, 2001.

[LWK] M. Loebbers, D. Willkomm, A. Koepke. The Mobility Framework for OMNeT++ Web Page. http://mobility-fw.sourceforge.net.

[MFHH02] S. Madden, M. Franklin, J. Hellerstein, W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. *ACM SIGOPS* 36(SI):131–146, 2002.

[MFHH03] S. Madden, M. Franklin, J. Hellerstein, W. Hong. The Design of an Acquisitional Query Processor For Sensor Networks. In *Proc. of the 2003 ACM SIGMOD Int'l. Conf. on Management of Data*. Pp. 491–502. 2003.

[NTCS99] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, J.-P. Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proc. of the 5$^{th}$ Int'l. Conf. on Mobile Computing and Networking*. Pp. 151–162. 1999.

[PG90] T. Poggio, F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE* 78:113–125, 1990.

[PJR07] J. Payton, C. Julien, G.-C. Roman. Automatic Consistency Assessment for Query Results in Dynamic Environments. In *Proc. of ESEC/FSE*. Pp. 245–254. 2007.

[PR99] C. Perkins, E. Royer. Ad Hoc On-Demand Distance Vector Routing. In *Proc. of the IEEE Wkshp. on Mobile Computing Systems and Applications*. February 1999.

[RJ08] V. Rajamani, C. Julien. Adaptive Data Quality for Persistent Queries in Sensor Networks. Technical report TR-UTEDGE-2008-001, 2008.

[RJPR08] V. Rajamani, C. Julien, J. Payton, G.-C. Roman. Supporting Adaptive Persistent Queries in Dynamic Environments. Technical report TR-UTEDGE-2008-017, 2008.

[RJPR09] V. Rajamani, C. Julien, J. Payton, G.-C. Roman. Inquiry and Introspection for Non-Deterministic Queries in Mobile Networks. In *Proc. of FASE*. May 2009.

[Var] A. Vargas. OMNeT++ Web Page. http://www.omnetpp.org.