6-2013

Energy-Efficient Collaborative Query Processing Framework for Mobile Sensing Services

Jin YANG
Technische Universitat Ilmenau

Tianli MO
University of Hawaii at Manoa

Lipyeow LIM
University of Hawaii at Manoa

Kai Uwe SATTLER
Technische Universitat Ilmenau

Archan MISRA
Singapore Management University, archanm@smu.edu.sg

DOI: https://doi.org/10.1109/MDM.2013.25

Citation

YANG, Jin; MO, Tianli; LIM, Lipyeow; SATTLER, Kai Uwe; and MISRA, Archan. Energy-Efficient Collaborative Query Processing Framework for Mobile Sensing Services. (2013). *2013 IEEE 14th International Conference on Mobile Data Management: 3-6 June 2013, Milan, Italy: Proceedings*. 147-156. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/1952

Published in 2013 IEEE 14th International Conference on Mobile Data Management: 3-6 June 2013, Milan, Italy: Proceedings, pp. 147-156.
https://doi.org/10.1109/MDM.2013.25

Energy-Efficient Collaborative Query Processing Framework for Mobile Sensing Services

Jin Yang*, Tianli Mo†, Lipyeow Lim†, Kai-Uwe Sattler*, Archan Misra‡
* Technische Universität Ilmenau † University of Hawai‘i at Mānoa ‡ Singapore Management University
E-mail: {jin.yang, kus}@tu-ilmenau.de, {tianli, lipyeow}@hawaii.edu, archanm@smu.edu.sg

***Abstract*—Many emerging context-aware mobile applications involve the execution of continuous queries over sensor data streams generated by a variety of on-board sensors on multiple personal mobile devices (aka smartphones). To reduce the energy-overheads of such large-scale, continuous mobile sensing and query processing, this paper introduces CQP, a collaborative query processing framework that exploits the overlap (in both the sensor sources and the query predicates) across multiple smartphones. The framework automatically identifies the shareable parts of multiple executing queries, and then reduces the overheads of repetitive execution and data transmissions, by having a set of ‘leader’ mobile nodes execute and disseminate these shareable partial results. To further reduce energy, CQP utilizes lower-energy short-range wireless links (such as Bluetooth) to disseminate such results directly among proximate smartphones. We describe algorithms to support our server-assisted distributed query sharing and optimization strategy. Simulation experiments indicate that this approach can result in 60% reduction in the energy overhead of continuous query processing; when ‘leader’ selection is dynamically rotated to equitably share the burden, we observe an increase of up to 65% in operational lifetime.**

I. Introduction

An increasing number of context-aware mobile computing applications involves the processing of continuous queries over data streams generated by a variety of smartphone-embedded sensors (such as GPS, accelerometer, microphone and gyroscope). A majority of the initial applications, in areas such as activity recognition [1], health monitoring [2] or indoor location tracking, apply such query processing on an *individualized* basis, using data solely from an individual's personal smartphone-embedded or wearable sensors. Recently a more sophisticated mobile computing paradigm is emerging that revolves around *collaborative sensing of shared context*.

In this pervasive mobile sensing paradigm, a potentially large number of physically-proximate smartphones (e.g., a group of mobile devices associated with customers in a food court or attendees in a lecture theater) simultaneously execute multiple continuous queries, each operating on a collection of individual, shared and remote (cloud-based) sensor data sources. In particular, a single complex query, executing on a smartphone, can involve three distinct types of sensor sources:

a) *Mobile and Personal*: Here, the data streams are associated with sensors embedded on the local smartphone and represent context that is *unique* to each specific individual-e.g., accelerometer data used to infer an individual's locomotive state.
b) *Mobile and Non-Personal*: While the data streams are still associated with smartphone-embedded sensors, unlike a), the context represented is now non-personal and can thus be inferred by utilizing sensor data from one or more alternate mobile devices–e.g., microphone sensors used to infer the ambient sound levels.
c) *Cloud-based, Remote*: Here, the data sources are associated with infrastructure-based "sensors", and must first be retrieved from the 'backend' infrastructure–e.g., a Web service that provides up-to-date temperature and pollutant concentrations in different urban areas.

Unfortunately, energy overheads continue to be the major bottleneck in the large-scale deployment of such continuous mobile sensing-based applications (e.g., continuous use of GPS data streams is known [3] to drain a smartphone's battery in 4-5 hours or less). In this paper, we propose and investigate a new form of *hybrid* **c**ollaborative **q**uery **p**rocessing framework (called **CQP**) for energy-efficient concurrent execution of such mobile sensing applications on multiple smartphones, that lies in between the two extremes of *a)* cloud-based centralized processing, where each mobile device merely forwards its relevant data to a centralized, infrastructure-based query processing engine and then merely receives the end result, and *b)* completely decentralized processing, where a centralized, infrastructure-based data collector collects and forwards all non-local data to each individual mobile device, allowing each smartphone to execute its complex queries completely locally and independently. Our hybrid framework is based on the fundamental observations that:

1) The energy overheads of wireless communication typically dominate the energy costs of on-board sensing (with GPS being a notable exception)–as reported in [4], wireless data transmission and reception typically consumes more than 21.5% of the energy budget on current smartphones (with displays being the other dominant energy consumer).
2) Smartphones today are equipped with multiple wireless communication interfaces (e.g., 3G, Wi-Fi and Bluetooth). Moreover, short-range radio interfaces (e.g., Bluetooth) consume significantly lower energy (per bit transmitted) than long-range 3G/4G interfaces.

To achieve energy efficiency, our hybrid framework thus

‡ This author is supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 under the research grant MOE2011-T2-1-001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the granting agency or Singapore Management University.

aims to both reduce the total volume of sensor data streams transmitted and preferentially employ shorter-range (e.g., Bluetooth-based) links for disseminating the sensor data and query states. To reduce the total volume of sensor data, we opportunistically identify and execute the shareable parts of multiple continuous queries on a single "leader" mobile device, which then distributes the intermediate query execution results to the other devices executing their queries, rather than have each smartphone retrieve its data streams independently. In practice, there are multiple mobile devices selected for executing different shareable portions of the queries, and the role of the "leader" is rotated among the group members for fairness. Moreover, CQP also transfers the intermediate query state directly between mobile devices using shorter-range links, thereby significantly reducing the intensity of 3G/4G-based data transfers between individual mobile devices and a cloud-based server.

Research Questions: To develop a viable cooperative and semi-distributed query sharing framework, we need to address several research questions:

- How do we identify the common (shareable) parts of multiple complex queries? We tackle this problem, in Section V, by proposing query matching and clustering algorithms.
- What parts (shareable or non-shareable) parts of multiple queries and remote data should be executed/retrieved on a single mobile device vs. locally executed/retrieved by individual smartphones? We explore this problem, in Section IV and propose 3 variants of the CQP framework.
- How do multiple phones coordinate to establish a shared and distributed query plan? We describe, in Section IV, how CQP uses a central server-assisted to derive and disseminate a consistent, shared query plan across multiple participating smartphones.

Key Contributions: We believe the following to be our key contributions in this paper:

- We propose the CQP collaborative query processing framework, where the continuous queries of multiple smartphones are processed in a semi-distributed fashion among the phones, and which relies on the use of more energy-efficient short-range wireless links to transfer data & intermediate query state among mobile devices. This framework is more energy efficient than the current model of independent query execution by individual smartphones and is more practicable than purely distributed, P2P query processing alternatives.
- We study 3 different variants of CQP (that differ in what parts of the data and intermediate query results are transmitted over short-range wireless interfaces) and establish the cases for which each of these variants outperforms the others.
- We perform simulation-based studies to quantify the energy savings achieved by CQP for realistic application scenarios, taking special care to incorporate the detailed energy consumption characteristics of different wireless technologies. Our results demonstrate significant ($\approx 60\%$ in our studies) reduction in energy overheads.

The rest of the paper is organized as follows. Section II provides a brief survey of the related and prior work. Section III then uses a couple of representative applications to explain the key concepts and insights behind the design of CQP. Section IV describes the component-level functional architecture of the CQP framework and describes the three models of semi-distributed, collaborative query processing. Subsequently, Section V presents algorithms for automatically identifying the common overlapping sub-queries that may be executed on a common node. Section VI presents simulation-based performance studies. Finally, Section VII concludes the paper with a discussion of open issues that we are working to address.

II. Related Work

There has recently been a wave of research on "people-sensing", centered on the use of sensor data collected from individual smartphones for applications such as traffic management, automated reminders and dynamic activity status updates [5], [6], [7]. To enable continuous energy-efficient execution of such mobile sensing applications, the Jigsaw platform [1] uses a pipelined stream processing architecture that adaptively triggers different sensors at different sampling rates while meeting the application's context accuracy requirements, while the SociableSense framework [8] adaptively shifts the individual components of the query processing functionality between a mobile device and the cloud. Our work is also motivated by the ACQUA framework [9] which performs query optimization for a single mobile device (retrieving data from multiple on-body sensors) and saves transmission energy by preferentially executing predicates with higher selectivity and lower acquisition cost. All of these approaches, however, target the optimization of streaming queries for a single smartphone in isolation, and do not consider the overlap between multiple queries executing on multiple mobile devices. The theoretical foundation of our use of low-power short-range wireless links for inter-smartphone communication is provided by [10], which empirically studied the energy consumption in mobile phones and analyzed the implications for network applications.

There has been relatively scant research on the collaborative processing paradigm, where multiple mobile devices share or coordinate their sensing activity. The Darwin mobile sensing framework [11] focuses on harnessing the sensing capability of multiple proximate phones to improve the accuracy of a single common query, by building and exploiting distributed classifiers that overcome the limitations of a single device. The ErDos framework [12] exploits collaboration among multiple nearby devices to loadshare the sensing burden (e.g., round-robin sharing of GPS sensors) of shared *ambient context*. Similarly, the use of GPS-based location sharing, via the use of low-cost Bluetooth communication among multiple nearby phones, for energy-efficient location tracking has been demonstrated in [13], [14]. However, these approaches focus either exclusively on sharing the raw sensor data (using direct short-range communication among the peer devices) or on sharing only location context. These approaches do not consider the problem of sharing (either wholly or partially) the *execution of arbitrarily complex streaming queries*; we believe that our work is among the first to suggest a distributed framework for energy-efficient sharing of query execution among multiple proximate mobile devices.

Our techniques of query sharing are inspired by recent work on multiple query optimization on data streams (albeit done without focusing on the specific challenges of mobile sensing). Recent work [15] employs pattern matching queries, state merging and indexing techniques in sharing work among queries reading the same input streams, while [16] focuses on sharing work among queries reading different streams. These techniques lay the foundation of our distributed query processing approach, where the common (shareable) part is executed on a single node and the device-specific residual portion of individual queries is executed on each individual smartphone. However, unlike such past work, which did not consider the specific challenges of mobile sensing, our work explicitly considers the different energy overheads associated with the transmission of data & query results between smartphones, and seeks to optimize the overall energy consumption. Our proposed CQP framework also considers the joint processing of both mobile sensor-generated data streams and cloud-resident structured data sets.

To identify a group of smartphones that may benefit from sharing their query processing tasks, our CQP framework also borrows from recent work on grouping techniques and location management for mobile phones. For example, to group phones based on their physical distance or proximity, we can utilize windowing and clustering [17] techniques. Alternatively, we can group based on the shared interest in specific contexts. Recent work (e.g.,[18][19]) provides solutions at both the network layer and application-level middleware support to achieve such context-dependent grouping. These solutions are efficient for the grouping requirement of our framework, thus we adopt these techniques by using the similarity among queries as the context information to form groups. While alternative forms of explicit or implicit grouping may be possible (e.g., in femtocellular architectures, where each cell size is $\approx$ 20meter in diameter, all phones associated with a single access point may be part of a common group), further research on grouping strategies for mobile phones is outside the scope of this paper.

III. Motivating Examples and High-Level Vision

CQP is motivated by our belief that many emerging mobile computing applications, both in consumer and enterprise domains, require not just an individual's sensor data, but *collective context* obtained by processing sensor streams from a *group of mobile devices*, as well as cloud-based data sources. We first describe two such representative use cases, extracting relevant insight that informs our design of the CQP framework. We then illustrate the alternative centralized or fully distributed alternatives for query processing, and explain the key principles by which the CQP framework can lower the energy overheads of continuous mobile query processing.

Example 1: **Meeting Status Indicator:** In enterprises, participants in a meeting may be running an application on their phones that continually tracks the meeting's progress and automatically detects when the meeting has ended (vs. when a single participant steps out briefly to take a phone call). The 'meeting ended' query may need to monitor: *a)* the location and current physical activity (sitting, standing) (using Wi-Fi and accelerometer data) of all the participants,*b)* the light and sound levels of the room and *c)* the status of the projector and conferencing phone in the room, and combine multiple predicates into a single complex query, such as:

"ALERT when' majority of participants stand up' AND ('room lights go on' OR 'room sound levels decrase') AND 'projector is switched off' ".

In this case, we can identify the following characteristics: *a)* each phone needs access to the 'activity' state of the other participants (obtained from their phone's accelerometer sensor), *b)* the light and sound levels are ambient environmental state that may be sensed by one smartphone and shared with the other smartphones, and *c)* the projector/phone status may need to be obtained from an infrastructure-based source (e.g., an enterprise Presence Server).

Example 2: **Emergency Triage Management:** As a much more data-intensive application, consider an emergency management scenario of the future where individual victims in a disaster area (e.g., a building) have their location and other medical parameters, such as Sp02 (Blood Oxygen Saturation) and heart rate, being monitored by their individual smartphones (which may be locally connected to body-worn sensor devices). As rescue professionals move through the disaster area, each of them may choose to execute slightly different continuous queries, corresponding to their specific field of expertise. For example, we simply consider two rescuers, located near each other, who have the following two queries:

Q1 *(Rescuer 1): "Track the nearest 3 victims' locations, such that each victim's SpO2 $< 95\%$ AND heart rate $> 120/min$"*

Q2 *(Rescuer 2): "Track the nearest 3 victims' locations, such that heart rate$> 95/min$ and blood pressure $> 150mmHg$. "*

Given the rescuers' proximity to each other, it is easy to identify the following characteristics: *a)* Each phone may need to monitor data from a non-identical, but *overlapping* set of victims' smartphones; *b)* If a particular victim is relevant to both Q1 and Q2, then, while the query predicates related to 'heart rate' values are not identical, they may be effectively combined within a *common processing logic*–i.e., if the heart rate is $< 95/min$, it is automatically $< 120/min$ as well. Thus, unlike the case of 'ambient sound/light levels' in Example 1 (where the predicate is identical across multiple smartphones), Example 2 demonstrates the possibility of having a query predicate be *partially* evaluated on a mobile device, while the residual predicate evaluation is performed on another smartphone.

Motivation for CQP. Having identified the salient characteristics of shared mobile sensing applications, we first present the two relatively straightforward (or naive) frameworks for implementing these queries. To quantify their relative energy impact, we focus on the 'Emergency Triage

Management' scenario described above, and assume an underlying set of 1000 vicitims, with each victim's sensor data consisting of five attributes: $\{GPS-lat, GPS-long, Severity, HeartRate, SPO2\}$. Each sensor data tuple is 20 bytes long; thus, the total sensor data volume is 20KB during each sampling interval of 20 seconds. We can identify two obvious alternatives for query processing:

- **Independent Query Processing:** In this framework, we assume the existence of a cloud-based centralized Relay Server, such that each of the 1000 hypothetical victims transmit their relevant sensor data to the Relay Server, which then transmits the entire set of such sensor data (using a wide-area 3G cellular interface) to each of the 2 rescuer phones separately. In this case, based on the energy characteristics of 3G radios, we calculate the energy consumed (for each 20 second interval) by the 2 receiver phones to be ≈ 23.4Joules.
- **Centralized Server Processing:** In this approach, the 2 rescuers upload their queries and the 1000 victims upload their relevant sensor data to a cloud-based Query Server using a wide-area 3G cellular interface. The Query Server executes the queries centrally and returns the query results to the mobile devices of the 2 rescuers. This approach is particularly inefficient when queries involve mostly local sensor streams, as the smartphones then waste energy unnecessarily in transmitting *all of* their sensor data to a cloud server.

To understand the motivation behind the CQP framework, we now explore possible optimizations that may lower the energy overheads. As a possible alternative to the Independent Query Processing framework, we could have one of the 2 rescuer phones receive the sensor data streams of the 1000 victims from the Relay Server over the 3G interface, and then relay this data in turn to the other rescuer's mobile device *using the cheaper shorter-range Bluetooth interface*, effectively replacing the 3G data reception by the 2nd phone with a lower energy Bluetooth-based transmission. Using the operating assumptions mentioned before, we compute that this approach will reduce the total enery consumed (for each 20 second interval) by the 2 rescuer phones to ≈ 10.54Joules.

As a further optimization, the 2 rescuer phones may now perform an additional step of *query sharing*. In this approach, the phone receiving the data streams from the Relay Server could then evaluate the common portions of the queries (e.g., the predicate related to 'HR >95/min') first, and then relay only the intermediate query result (instead of the raw data streams) to the second phone. Depending on the selectivity properties of the predicates, this should further reduce the transmission energy overheads.

The three variants of our proposed CQP framework implement various combinations of the above-proposed optimizations. At a high-level, CQP envisages that, given a group of mobile devices simultaneously executing multiple queries, energy savings will be achieved by having a single node (a designated 'leader' mobile device) compute the common, shareable sub-queries and return the intermediate results to the individual group members (for final query processing).

IV. The CQP Framework

A. Overview

Our collaborative query processing (CQP) framework conceptualizes the eco-system of mobile devices, base stations, cloud computing servers as a massive distributed database system where the data sources can be either sensor data streams from individual mobile devices or relational tables residing in cloud-based database servers. Mobile applications on each mobile device are viewed as client applications that pose (possibly continuous) queries on this conceptual distributed database system. This view of mobile applications enables us to collaboratively process queries among groups of mobile devices, so as to to reduce the total energy consumption across all mobile devices and increase their operational lifetime.

Roles. The CQP framework groups mobile devices into *location-based groups* and *query-based groups*. Intuitively, mobile devices within the same location-based group are physically close together, and thus experience the same 'ambient context' and can communicate over lower-powered links, such as Bluetooth or Wi-Fi. Query-based groups are a further clustering of phones within a location-based group, such that the queries of the phones within a query-based group are "overlapping" and thus able to share data sources and processing. The CQP framework abstracts the function that each device and/or system performs into three distinct roles.

- **A group leader** receives data, sends data, or process data on behalf of the member devices or phones in the group.
- **A member device/phone** belongs to a group and poses a set of queries on local sensor data or remote data on a server.
- **A server** manages a collection of groups, optimizes and co-ordinates the collaborative processing of each member device's queries. Without loss of generality, we assume that the server is also a proxy to any relational data that the queries may require.

We envision that in most cases, one mobile device within a query-based group will take on the role of group leader (besides being a member device). The group leader role can be suitably rotated among the member devices in the group to balance the energy-consumption. The edge server at or near the base station will play the role of the server. However, the CQP framework is quite general and will support other cases as well. For example, in femtocellular networks, all mobile devices connected to a single access point (AP) can be implicitly part of the same *location-based group*, and the server and group leader role may be performed by the AP itself.

Query Model. We assume that the queries posed by member phones operate over both relational data and sensor streaming data. Query languages for the specification of such queries are beyond the scope of this paper, although streamSQL can certainly be used. We assume that a query is modeled by a tree of standard relational algebra operators and the continuous execution semantics are implemented as periodic execution (with period T) of the query tree. Sensor data streams are incorporated into this model by considering finite windows (win) of the data streams at each periodic execution time; i.e., the evaluation at time instant t utilizes sensor data within the time interval $(t - win, t)$. In the most general form, a query tree would operate on both local sensor data and remote (relational) data.

Grouping by Location and Queries. To collaboratively share resources for query processing, mobile devices or smartphones need to be clustered into groups. The CQP framework uses a two-level grouping of phones. The first level of grouping clusters phones using the GPS location of the phones. The resultant groups are called *location-based groups*. The exact distance threshold depends on the type of applications to be supported and on the communication link used–e.g., for Bluetooth-based intra-group communications, all members of a group can be no more than 10 meters apart. The second level of grouping clusters the phones within each location-based group using the amount of commonality of the queries in the group. The resultant groups are called *query-based groups*. A location-based group can contain multiple, overlapping query-based groups.

Both location-based and query-based groups are managed by the server. All phones periodically send their location data to the server, which clusters the phones into location-based groups and manages the group membership when phones leave or new phones join the system. Similarly, all phones periodically send their queries to the server and the server will cluster the phone-query pairs into groups and manage the group membership.

Query Processing Strategies. To illustrate how queries are processed using the CQP framework, we first consider how queries would be executed in the *absence* of CQP. The queries are still assumed to be executed periodically, but there is no concept of groups and group leaders. Two possible strategies are possible depending on whether most of the query processing is performed at the phone or at the server.

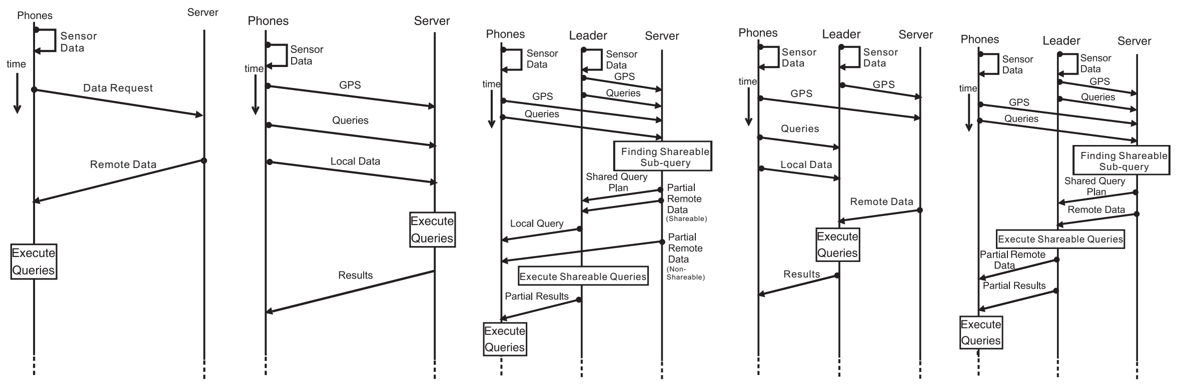
- **Naive-P** Fig. 1(a). At each query execution, each phone sends requests for remote data to server. The server transmits the required remote data to each phone. Each phone acquires the local sensor data (e.g. accelerometer). After all required data have been received, the phone executes the query and obtains the result.
- **Naive-S** Fig. 1(b). At each query execution, each phone acquires the local sensor data (e.g. GPS) and sends the local sensor data and the query to the server. The server receives the sensor data and query from the phone, executes the queries using both local sensor data and remote data, and sends the results back to the phone.

Now consider query processing using the CQP framework. We outline three possible strategies of collaborative query processing, CQ-S, CQ-L and CQ-LS, that differ mainly in the amount of processing handled by the group leader. In all three strategies, groups are formed and managed by the server as described previously. Using the query-based grouping information, the server then co-ordinates the collaborative query execution via collaborative query execution plans (CQEP). A query-based group must be contained within a location-based group. A phone-query pair is grouped into a query-based group if the query can share resources (such as local sensor data or common query fragments) with the other phone-query pairs in the query-based group. A collaborative query execution plan consists of group membership information, as well as query plan fragments for each device/system, including the group leader and the server. We outline how the collaborative query processing is performed for the three strategies within a query-based group of phones.

- **CQ-S** Fig. 1(c). Each member phone receives from the server a fragment of the CQEP for execution. The group leader also receives from the server a fragment of the CQEP called the *shared query plan*. The results of the shared query plan needs to be sent to the member phones in order for the member phones to complete execution of their CQEP fragment. In this strategy, each member phone acquires the remote data required by their CQEP fragment from the server independently. The leader phone and each member phone also acquires local sensor data from their own sensors. The group leader then executes the shared query plan and sends the results to the member phones, which then individually execute their residual CQEP fragments.
- **CQ-L** Fig. 1(d). The CQ-L strategy is exemplified by the group leader performing most of the query processing for the member phones. Each member phone sends its query and local sensor data to the leader. The leader acquires all the required remote data from the server. Upon receiving the required sensor data and remote data, the leader executes all the queries for each member phone and sends the results to each member phone.
- **CQ-LS** Fig. 1(e). The CQ-LS strategy differs from the CQ-S strategy by having the group leader act as a proxy for all member phones and acquire *all required* remote data from the server, instead of the CQ-S approach where each member phone acquires the remote data independently from the server. CQ-LS aims to exploit the low-powered Bluetooth link among group members for data dissemination. The group leader and each member phone also acquire local sensor data from their own sensors. The group leader then executes the shared query plan and sends the results and the required remote data to each individual member phone, which then individually executes its residual CQEP fragment.

B. The CQP Architecture

Fiigure 2 shows the functional architecture of the CQP framework. To enable query sharing, some key components of the framework should either be implemented on the server (either embedded at the wireless base stations or implemented as cloud services). These key components coordinate the sharing of processing among different mobile phones that are within Bluetooth-based communication range. On the mobile device side, the framework is implemented as middleware



(a) Naive-P (b) Naive-S (c) CQ-S (d) CQ-L (e) CQ-LS

Fig. 1. 2 Naive methods and 3 methods of Collaborative Query Processing

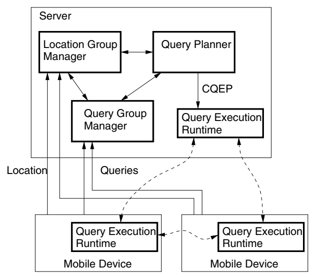


Fig. 2. System architecture for the CQP framework.

that interconnects the applications which generate the queries, the query execution engine and the communication control components. The Location Group Manager (LGM) determines the location-based groups for the phones and manages the group membership. The Query Group Manager (QGM) builds the query-based groups on top of the location-based groups and manages the query-based group membership. The Query Planner (QP) optimizes and produces a collaborative query execution plan (CQEP) that orchestrates the entire collaborative query execution among member phones, group leaders and server. The Query Execution Runtime (QER) receives a CQEP or fragments thereof and executes the CQEP. The CQEP contains group membership information and the job specification for each device including where to acquire data, what queries to execute, and where to send the results.

Location Group Manager (LGM). The LGM clusters mobile devices using their location or GPS data and maintains the group membership information as devices move into range and out of range. This is a well studied problem in mobile communication systems and we rely on current state-of-the-art techniques (e.g. [20]) to find and manage location-based groups. Whenever changes in smartphone locations causes a change in location-based groups, the Query Planner may have to potentially regenerated a new CQEP.

Query Group Manager (QGM). The QGM builds on the location-based group information to determine and manage query-based groups. For a single location-based group, each member phone periodically sends their queries to the QGM. The QGM clusters the set of phone-query pairs according to the similarity or degree of commonality of the queries. Details on the clustering algorithm is discussed in Section V. The output of the clustering algorithm consists of query-based groups (of the phone-query pairs) and the maximal common sub-query associated with each group. The QGM periodically recomputes the query-based groups, due to dynamics in both node mobility and changes in specified queries.

Collaborative Query Planner (QP). The QP is responsible for constructing a collaborative query execution plan (CQEP), using query optimization in to find the best CQEP that achieves objectives, such as reducing the SUM of energy consumption or maximizing the system lifetime of all the mobile phones in the group. The CQEP is a distributed query execution plan that consists of jobs for each member phone, group leader and server. The jobs are specified by relational algebra operator trees augmented with operators that acquire local sensor data and remote data and with operators that send the results of the job to other devices or servers. Note that the QP is stateful in the sense that each invocation is not independent of the CQEP produced in the previous invocation. The statefulness allows the QP to generate CQEPs that rotate the role of the group leader among phones in a group or assign resource-intensive jobs to phones with the longest battery life.

Query Execution Runtime (QER). The QER coordinates the collaborative query execution among different devices to generate coherent query results for each query. The QER resides on all devices and servers including member phones, group leaders and servers. Conceptually, it receives a CQEP and executes the jobs (query trees) in the CQEP that are assigned to the device. The server QER and group leader QER may also forward jobs to other member phones and co-ordinate the forwarding of relevant results.

Handling Change. To avoid incurring a prohibitive cost due to frequent replanning, the LGM may restrict location-membership to *relatively stable* nodes, borrowing from recent mobile computing techniques that use phone-usage and activity patterns (e.g., whether a user is actively using an email App on their phone or is currently seated) to predict the *likely movement pattern* of individual devices. Moreover, the QP may choose stability over optimality, leaving query plans unmodified if the performance gains are predicted to be incremental. In general, if the "leader" node assigned to a location based group moves outside the Bluetooth communication range, the LGM triggers a re-plan procedure to generate new location based group information, which is then used to generate the modified CQEP. Developing practically effective strategies for managing group dynamics is an open problem that we defer to our future work. In this paper we focus on the algorithms for collaborative query processing.

V. Finding Query-based Groups

To automatically generate a CQEP, a central challenge is to identify the membership of a query-based group, i.e., identify the shareable, common parts of multiple queries. We now describe the algorithm to address this challenge. The input of the algorithm is a collection of queries posed by the collection of phones in a given location-based group. The j-th query posed by phone i is denoted by $q_{i,j}$. The output of the algorithm is a collection of query-based groups and the common sub-queries associated with each of the query-based group. Note that, to avoid redundant query execution and energy overheads, the query-based groups form a partitioning over the set of queries $\{q_{i,j}\}$, i.e., each query should not be in more than one query-based group.

Query Tree. A query $q_{i,j}$ is a tree of relational operators. More formally, a query is a tree (V, E) where V is a set of nodes and E is a set of edges. A leaf node represents a data source and a non-leaf node represents a relational operator. A directed edge between two nodes indicates the direction of data transfer. Each node $v \in V$ is associated with a name and a parameter: $name(v)$ denotes the name of the operator or the name of the data source, and $param(v)$ the parameter that the operator requires. For data source nodes, the $param(v)$ consists of the information required to retrieve data from that data source. For the relational selection operator σ, the $param(v)$ would be the selection predicate. For the projection operator π, the parameter would be the set of columns to project. Some relational operators such as $\times, \cup, \cap$ do not take any additional parameters and hence their $param(v)$ would be empty.

Two nodes u, v are **name matching** if $name(u)$=$name(v)$. Two query subtrees rooted at u and v are name matching if the two trees are isomorphic (there exists a bijection between the two trees) and each pair of corresponding nodes are name matching. Two nodes u, v are **exact matching** if $name(u)$=$name(v)$ and $param(u)$=$param(v)$. Two query subtrees rooted at u and v are exact matching if the two trees are isomorphic and each pair of corresponding nodes are exact matching. The reason for distinguishing the weaker name matching is that it is often possible to obtain a common subquery that contains nodes that are only name matching with a relaxation on the parameter associated with the node. An example of such relaxation would be two queries p and q containing a selection operator on the same data source. The selection parameter in p is HR>95 and in q is HR>120. A common subquery would contain the selection operator with parameter HR>95.

Algorithm 1 Strategy 1 for Finding Query-based Groups

```
Input: A set of queries $Q = \{q_{i,j} : j$-th query of phone $i\}$
Output: A set of query-based groups and the associated shared subqueries
1: Find collection of shared subqueries using [21]
2: Find query-based groups using a greedy set cover algorithm
3: Remove overlaps between groups
```

Algorithm 2 Strategy 2 for Finding Query-based Groups

```
Input: A set of queries $Q = \{q_{i,j} : j$-th query of phone $i\}$
Output: A set of query-based groups and the associated shared subqueries
1: Find query-based groups using a hierarchical agglomerative clustering algorithm
2: for all query-based groups do
3:     Find the shared subquery forest
```

We propose two strategies for finding query-based groups. The first strategy is outlined in Algorithm 1. We first find a collection of shared subquery trees given all the queries in the location-based group. Those shared subqueries are maximal subquery trees or forests that are shared by two or more queries. The grouping of queries induced by the shared subqueries may not cover all queries, and may overlap. Since the query-based groups need to cover all the given queries, we find a minimal set cover using a greedy algorithm. Since a set cover yields query-based groups that may still overlap, we remove overlaps to ensure that the query-based groups forms a partition.

Finding a collection of shared subquery trees is done using a bottom-up algorithm similar to [21]. An inverted index is constructed that maps leaf nodes (name and parameter) to a list of queries. Starting from the largest list, each pair of queries in the list is examined to find the shared subquery tree. If that particular shared subquery tree has been found before, the pair of queries is merged with the existing list for that shared subquery tree.

The second strategy is outlined in Algorithm 2. We first apply a hierarchical agglomerative clustering algorithm to cluster the queries using a distance function based on the number of shared data sources (exact matching on leaf nodes) between two queries. The stopping criterion can be based on the number of clusters or the size of clusters (more sophisticated conditions can be used too). Those clusters yield the query-based groups; a tree isomorphism based algorithm is then used to find the shared subquery tree or forest for each query-based group.

VI. Experimental Evaluation

This section describes our simulation-based experiments to quantify the performance gains (in terms of the reduction in energy overheads) of the proposed CQP framework. Our experiments are conducted using a Python-based simulator which accepts as input groups of phones, phones' queries and the data streams required by the queries. Each query may require some amount of remote data from the server and some amount of data from the local sensors characterized by a parameter called the local-remote ratio of local data size to remote data size. The queries at each phone is evaluated periodically every 20 seconds. The total duration of simulation is 3600 seconds (one hour). We perform simulations for the five query processing strategies introduced in Section IV-B: Naive-P, Naive-S, CQ-S, CQ-L and CQ-LS. The results are averaged over all the 20-second execution intervals within the 3600 second simulation time.

Data. We generate synthetic relational data of varying sizes. For the remote server data required by a given query at each evaluation interval, we vary the size uniformly randomly from 10KB to 100KB in steps of 10KB. For the streaming local sensor data required by the query at each evaluation interval, we generated the value of each attribute in the streams using the normal distribution $N(\mu, \sigma)$ (with appropriate truncation to avoid underflow below 0 or overflow above 100%). The data size of the streaming local sensor data for a given query is determined using the local-remote ratio for that query. Unless otherwise specified the local-remote ratio for a query is uniformly randomly selected between 0.25 and 4.

Performance metrics. We use three metrics in our experiments. *Energy consumption* is the energy consumed spent on data transmission on each phone and is calculated according to the models in Fig. 3. *Data transmission size* is the sum of the quantity of data transmitted between phones (using Bluetooth) as well as between server and phones(using 3G). *System operational lifetime* is the duration between the beginning of the simulation to the first device running out of energy.

A. Energy Consumption

Our first experiment answers the question: To what extent can CQP help reduce energy consumption in the system? The experimental setting consists of 18 smart phones in one group sending out queries to be answered. The queries are evaluated every 20 seconds (we will call this a query-reply round). The amount of server data required by each query is fixed at 10KB and the ratio of local data is randomly generated as described previously. We measure the energy consumed by each phone in the system as they process the queries using each of the five query processing methods. For the CQP methods (CQ-S, CQ-L and CQ-LS), without loss of generality since the groups are independent, the 18 phones are grouped into one location-based group and one query-based group.

Fig. 4 shows the total energy consumed (over the 18 phones) for each of the five methods averaged over the query-reply rounds. Observe that the CQP methods have a significant advantage in energy efficiency. In each query-reply interval, Naive-P and Naive-S consumed over 215 Joules to answer the queries. CQ-L and CQ-LS both prevent their member phones from acquiring remote data using 3G, effectively halving their energy consumption (cf. naive methods).

To better understand the data transmission characteristics of each method, we measured the average total data transmitted in one query-reply round and these are plotted in Fig. 5. Naive-S uses the centralized server to process all the queries and thus does not require the server data to be transmitted to each phone. Naive-P requires each phone to download the data required by the query from the server resulting from more data transmission than Naive-S. CQ-S adopts a distributed processing strategy and thus its data transmission characteristics is inbetween Naive-P and Naive-S. CQ-L and CQ-LS both requires large amount of data to be transmitted between the leader and the group members albeit using short range communication technology that is more energy efficient. One important lesson here is that minimizing data transmission may not necessarily minimize energy consumption. Fig. 4 and Fig. 5 show that exploiting more energy-efficient communication channels can result in dramatic energy savings even at the cost of data transmission.

We have looked at the sum of energy consumption over all 18 phones. Another important question is: What is the energy consumption for each of the 18 phones? Fig. 8 plots the profile of the energy consumption for each of the 18 phones for the five methods. In general, the CQP methods decrease the energy consumption on each member phone at the cost of an increase in energy consumption on the leader phone. The additionl energy burden on the leader motivates the need to dynamically rotate the leader role among the phones in the group in order to maximize the system operational lifetime.

B. System Lifetime

In this experiment we address the question: how long can the system operate using the CQP framework before one phone runs out of energy? If the system's lifetime is reduced compared to the naive methods, smartphone users would have no motivation to adopt the CQP framework. The simulation setting is similar to the previous experiment and begins with each phone having a battery level that is uniformly randomly generated between 10K to 20K Joules. In the fixed leader simulation, one phone is randomly chosen as the leader for the entire simulation. In the dynamic leader simulation, the leader is chosen from the group according to the phones' current battery level every 20 seconds. We measure the lifetime of the system as the time for the first phone to run out of battery. The results are plotted in Fig. 6 and Fig. 7. As expected, the CQP methods with the fixed leader approach have shorter system lifetime than the naive methods. However, the CQP methods using the dynamic leader approach significantly outperform the naive methods with CQ-L achieving a system lifetime 57% longer than Naive-S.

C. Influence of Group Sizes

The previous experiments have assumed a single location-based group and a single query-based group. In this section,

Wi-Fi

$$E_t = \begin{cases} P_i * (\frac{N}{f} - \frac{N*S}{B}) & \\ \quad +P_a * \frac{N*S}{B} + E_{switch} & \text{if} \frac{N}{f} - \frac{N*S}{B} > Th_{idle} \\ P_a * \frac{N}{f} & \text{otherwise} \end{cases}$$

Bluetooth

$$E_t = P_i * (\frac{N}{f} - \frac{N*S}{B} - T_{switch}) + P_a * (\frac{N*S}{B} + T_{switch})$$

3G

$$E_t = E_{bit} * (N * S) + E_{switch} + P_{tail} * T_{tail} + M * \frac{N}{f},$$

	IEEE 802.11	Bluetooth 2.0+EDR	3G
P_a	947 mW	60mW	–
P_i	231 mW	5 mW	–
P_{tail}	–	–	620 mW
B	54 Mbps	1 Mbps	1 Mbps
M	–	–	20 mW
E_{bit}	–	–	25 mJoule/bit
E_{switch}	14 μJoule	–	3.5Joule
Th_{idle}	100 ms	–	–
T_{switch}	–	6 msec	–
T_{tail}	–	–	12.5 sec

Fig. 3. Analytical energy overhead models of Wi-Fi, Bluetooth, and 3G wireless links.

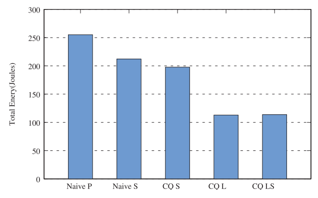


Fig. 4. Total energy cost.

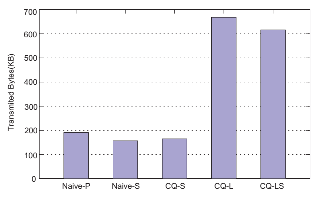


Fig. 5. Total bytes transmitted.

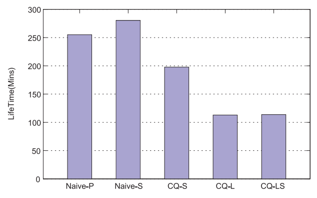


Fig. 6. System lifetime (fix leader).

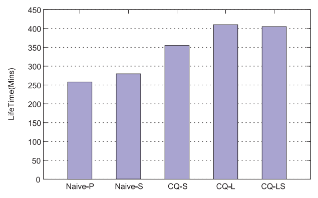


Fig. 7. System lifetime (dyn. leader).

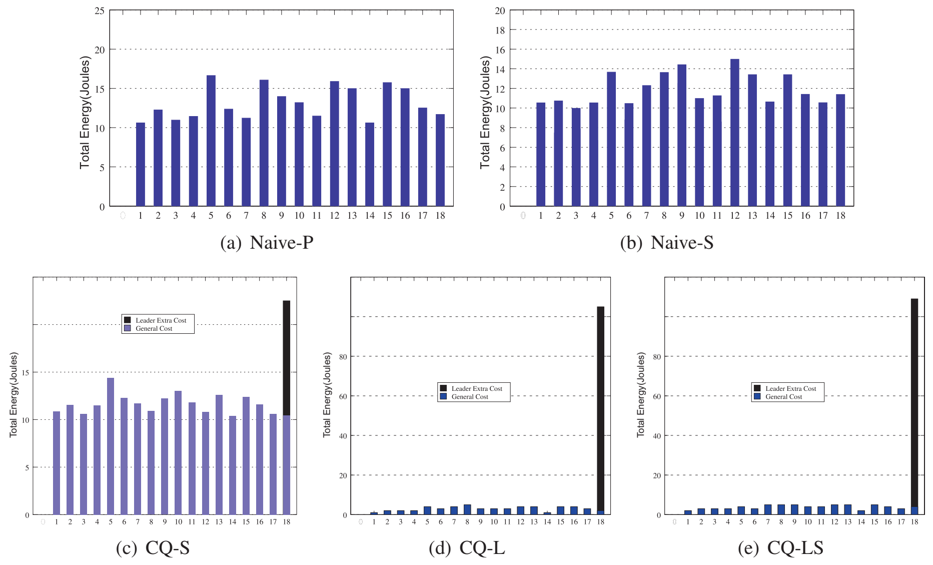


Fig. 8. Distribution of energy consumption over all the phones.

we study the effect of the sizes of these two grouping on the total energy consumption of the CQP methods.

To study the effect of the size of location-based groups, we fixed the total number of phones to 36 and performed simulations for 4 (location-based) groups of 9 phones each, 2 groups of 18 phones each, and one group of 36 phones. In each location-based group, the number of query-based group is set so that each query-based group has size 4. The ratio of local to remote data is set at 2:8 and the size of the server data required by each query is set randomly as described in the beginning of Sec. VI. Fig. 9 shows the simulation results. Observe that the larger the location-based group the greater the energy savings for the CQP methods which is consistent with our understanding that a larger group leads to more sharing of resources within the group.

To study the effect of the size of query-based groups, we used one location-based group with 16 phones and performed for each method three simulations with query group sizes 2,4, & 8 respectively. The number of query-based groups is set according to the query group size. Our results as plotted in Fig. 10 fit our intuition that bigger groups lead to greater savings.

D. Local Data vs Remote Data

In this section we investigate the effect of the sizes of the remote server data and the sizes of the local sensor data required by the query on the energy consumption. For each

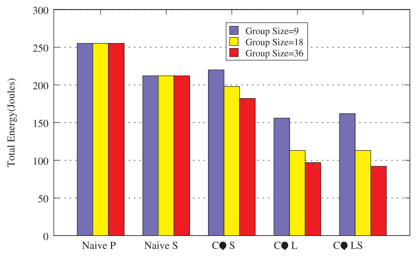


Fig. 9. Number of Group Size

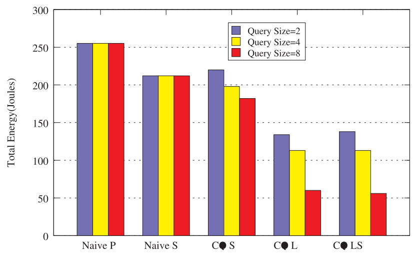


Fig. 10. Number of Query Group Size

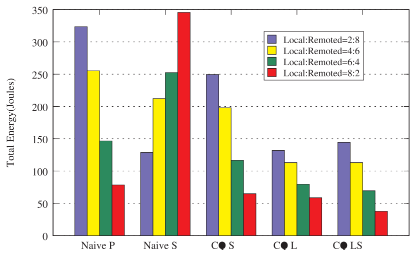


Fig. 11. Ratio of Local Data vs. Remote Data

of the naive and CQP methods, we performed 4 simulations varying the ratio of local to remote data according to the set $\{2{:}8, 4{:}6, 6{:}4, 8{:}2\}$. The setting consists of one location-based group with 16 phones partitioned into 4 query-based groups of size 4 each. Fig. 11 plots the results. Observe that Naive-P favors queries that require mostly local data, Naive-S favors queries that require mostly remote data. The CQP methods tend to favor queries that require mostly local data, even though CQ-L and CQ-LS performs comparably to Naive-S for queries that require mostly remote data. For queries that require mostly local data, the CQP methods still outperform the naive-P method. An important implication of our result is that the ratio between local and remote data required by the queries is one of the key factors to the selection the query processing strategy.

VII. Conclusion and Future Work

In this paper, we have introduced the CQP framework for energy-efficient continuous evaluation of multiple complex queries over mobile sensing data streams. The CQP framework uses two innovations for sharing query execution and sensor data streams among multiple mobile nodes. The key to the query optimization framework is the automated identification of the similarity of the queries among different mobile users, and the execution of shareable fragments of multiple queries on a common 'master' mobile node. CQP further reduces energy overheads by using low-energy wireless interfaces (such as Bluetooth) to exchange data and query state directly between nearby smartphones. We described the algorithms to detect sharable part of queries and analyzed three different optimization variants of CQP. Our results on synthetic traces indicate that, compared to existing purely centralized or decentralized solutions, the hybrid CQP framework can result in 60% reduction in the energy overheads, and 40% to 65% increase in system operational life time (if the 'leadership' role is rotated dynamically).

Our ongoing work encompasses two orthogonal threads. At a systems level, we are implementing CQP on an Android-based smartphone platform, and will then quantify CQP's energy savings via user studies with real-life sensor traces, instead of currently-used synthetically generated data. On an algorithmic level, we are refining the algorithms to include additional query semantics, such as the support of sliding window queries and to develop more *robust* query plans (that remain close-to-optimal even under dynamic node movement and query changes).

References

[1] H. Lu, J. Yang, and et al, "The Jigsaw Continuous Sensing Engine for Mobile phone Applications," in *ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*, 2010.

[2] A. Roychoudhury, B. Falchuk, and A. Misra, "MediAlly: A Provenance Aware Remote Health Monitoring Middleware," in *8th IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2010, March.

[3] S. Gaonkar, J. Li, and et al, "Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation," in *Mobisys'08*, 2008.

[4] N. Sklavos and K. Touliou, "A System-Level Analysis of Power Consumption and Optimizations in 3G Mobile Devices," in *1st International Conference on New Technologies, Mobility and Security (NTMS'07)*, 2007, pp. 225–235.

[5] J. Yoon, B. Noble, and M. Liu, "Surface street traffic estimation," in *MobiSys'07*. ACM, 2007, pp. 220–232.

[6] T. Sohn, K. Li, and et al., "Place-its: A study of location-based reminders on mobile phones," in *Ubicomp'05*. ACM, 2005, pp. 232–250.

[7] E. Miluzzo, "Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application," in *ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, 2008.

[8] K. Rachuri, C. Mascolo, M. Musolesi, and P. Rentfrow, "SociableSense: Exploring the Trade-offs of Adaptive Sampling and Computation Offloading for Social Sensing," in *ACM MobiCom*, 2011, August.

[9] A. Misra and L. Lim, "Optimizing Sensor Data Acquisition for Energy-Efficient Smartphone-based Continuous Event Processing ," in *MDM2011*, June 2011, pp. 88–97.

[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications," in *SIGCOMM*. ACM, 2009.

[11] E. Miluzzo and et al., "Darwin phones: the evolution of sensing and inference on mobile phones," in *MobiSys*, June 2010.

[12] N. Vallina-Rodriguez and J. Crowcroft, "Erdos: Achieving energy savings in mobile os," in *ACM MobiArch*, 2011.

[13] Z. Zhuang, K. H. Kim, and J. Singh, "Improving Energy Efficiency of Location Sensing on Smartphones," in *MobiSys'10*. ACM, 2010.

[14] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive GPS-based positioning for smartphones," in *MobiSys'10*. ACM, 2010, June.

[15] A. Demers, J. Gehrke, and et al, "Towards expressive publish/subscribe systems ," in *EDBT'06*, 2006.

[16] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation ," in *SIGMOD'06*, 2006.

[17] L.-A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C.-C. Hung, and W.-C. Peng, "On discovery of traveling companions from streaming trajectories," in *ICDE'12*. IEEE, 2012, pp. 186–197.

[18] R. Cabaniss and S. Madria, "Dynamic social grouping based routing in a Mobile Ad-Hoc network," in *HiPC'10*, 2010.

[19] K. Hazra and K. Nahrstedt, " Group formation and communication in mobile wireless environments ," in *Military communications conference*, 2011, November.

[20] Y. Xiao, Y. Pan, and J. Li, "Movement-based location management for 3G cellular networks," in *GLOBECOM'03*, vol. 7, Dec, 2003, pp. 4101–4105.

[21] L. L. and B. B., "Optimizing access across multiple hierarchies in data warehouses," in *HICSS'11*, 2011, pp. 1–10.