Cloud-based Query Evaluation for Energy-Efficient Mobile Sensing

Tianli Mo[‡], Sougata Sen[†], Lipyeow Lim[‡], Archan Misra[†], Rajesh Krishna Balan[†], and Youngki Lee[†]
[‡]University of Hawai'i at Mānoa and [†]Singapore Management University

Abstract

In this paper, we reduce the energy overheads of continuous mobile sensing, specifically for the case of context-aware applications that are interested in *collective context or events*, i.e., events expressed as a set of complex predicates over sensor data from multiple smartphones. We propose a cloud-based query management and optimization framework, called *CloQue*, that can support thousands of such concurrent queries, executing over thousands of individual smartphones. Our central insight is that the context of different individuals & groups often have significant correlation, and that this correlation can be learned through standard association rule mining on historical data. *CloQue's* exploits such correlation to reduce energy overheads via two key innovations: i) dynamically reordering the order of predicate processing to preferentially selects predicates with not just lower sensing cost and higher selectivity, but that maximally reduce the uncertainty about other context predicates; and ii) intelligently propagating the query evaluation results to dynamically update the confidence values of other correlated context predicates. We present techniques for probabilistic processing of context queries (to save significant energy at the cost of a query fidelity loss) and for query partitioning (to scale *CloQue* to a large number of users while meeting latency bounds). An evaluation, using real cellphone traces from two different datasets, shows significant energy savings (between 30 to 50% compared with traditional short-circuit systems) with little loss in accuracy (5% at most). In addition, we utilize parallel evaluation to address the latency issues. The experiment shows our approach saves up to 65% evaluation time.

I. Introduction

This work proposes a system for efficiently executing *multi-person, continuous* queries, expressed over context derived from smartphone-embedded sensors of a *large group* of individuals. In many context-aware computing scenarios, users are interested in context or events that are not just derived from a single individual, but instead results from the collective context of a group of individuals. For example, a university student may wish to be notified when the rest of her project mates have reached the meeting room, indicating the imminent start of a planned meeting. Similarly, there are myriad examples centered around proximity awareness–e.g., reminders about the questions I need to ask when I am next in the same room as my manager. Evaluation of such continuous, multi-person queries will often involve diverse sensing and context inference tasks across multiple users, which severely aggravate the energy overhead of smartphones beyond individual mobile sensing.

Our key premise is that it is possible to significantly reduce the energy overheads of evaluating such multi-person (or *collective*) context, while minimally sacrificing the accuracy of the derived context, by designing query optimization techniques that exploit two features: i) *Correlation Across Users:* Users often perform activities in coordinated or correlated fashion–e.g., John and Mary are colleagues working at the same company, and they usually go to lunch at the cafeteria together. If one day we know Mary is at the cafeteria at lunch time, we can infer that John is quite possibly at the cafeteria as well without actually acquiring any location data from John. Cross-user correlation implies that the context of a person B can be inferred simply by retrieving the context about another person A, thus avoiding the energy burden on B's smartphone. ii) *Sensor Diversity:* Different context attributes constituting a collective query require data from different sensors, and can thus have different sensing costs–e.g., GPS-based location context is known to be much more energy-draining that WiFi-based location context.

Both of the above strategies for query optimization have been investigated previously (e.g., context correlation in [15] and short-circuiting of queries in [7], [12]), but almost exclusively for retrieving the context of an individual user in isolation. Our intention is to utilize the principles of query short-circuiting and context correlation to make evaluation of context more energy-efficient, but for collective context queries, *at scale*–e.g., over hundreds or thousands of individuals in environments such as office buildings or college campuses. Such a setting gives rise to several unique challenges:

- *Varying levels of Cross-User Correlation:* Correlation across contexts for the same individual is fairly straightforward to ascertain and utilize-e.g., a person "driving a car" is not "at home". However, correlation in the context attributes across individuals cannot, for the most part, be expressed simply by deterministic rules. Moreover, different groups of individuals may exhibit differing levels of correlation (pairwise, as groups of three individuals and so on). Accordingly, a query optimization framework must be able to *discover* and *reason with* varying levels of context correlation, across groups of varying sizes.
- *Shared Context of Interest across Queries:* Multiple concurrent queries are likely to require the same context from the same individual. For example, consider one query looking for whether *"User A & B are in the cafeteria"*, while another looks for whether *"A & C are in the meeting room"*: user A's location context is now utilized by two distinct queries. In this case, the query optimization logic must not only consider the common metrics for single-query evaluation, e.g., *acquisition cost* and *selectivity*, but also *coverage* (how many queries depend on a particular context predicate).
- *Variable Processing Latencies:* Applications may need to be notified of collectively derived context within a specified time limit. For many sensors, determining a specific context requires

a certain minimum duration (e.g., to determine if an individual is 'walking' or 'sitting', accelerometer data must be acquired for at least 5 seconds). Thus, besides saving energy, the query optimization framework must adhere to application query latency bounds, especially when the number of participating phones or query predicates is large.

To support energy-efficient evaluation of such multi-person, continuous queries on a large (at least *campus-level*) scale, we propose *CloQue*[1], a cloud-based framework. Applications submit their continuous collective-context based queries to the *CloQue* cloud engine, which then retrieves the required contextual states by dynamically tasking specific sensors on individual smartphones. The *CloQue* engine activates only those sensors (at each query evaluation instant) essentially needed to answer the currently executing set of collective queries, and then notifies the subscribing applications when each overall query is completely evaluated (either deterministically or probabilistically).

We present a novel, unified query evaluation algorithm, and associated data structures, that allows *CloQue* to : *a)* balance selectivity and coverage objectives, and *b)* incorporate predicted context likelihood (whether derived individually or via correlation from other observed contexts). *CloQue* comes in two different variants: (1) *CloQue-NoRules*, that performs query short-circuiting without considering context correlation, and thus achieves 100% fidelity in query processing, and (2) *CloQue-Full*, that additionally incorporates correlation-based context prediction to support probabilistic evaluation of context queries, thus achieving greater energy savings with minimal loss in processing fidelity. We evaluate *CloQue* using two different datasets that provides traces of real-life context captured from several hundred participants. We show that *CloQue* can indeed provide significant (up to 60%) savings in energy accuracy in practical campus-like environments.

Key Contributions: Our key contributions are:

1 **Unified Specification of Query Predicates and Probabilistic Context Correlation:** To leverage the query optimization potential offered by cross-individual correlation, *CloQue* allows applications to specify a probabilistic *confidence threshold* for each collective query. We develop a novel unified representation of both the relationships among the query predicates (expressed in DNF) and the correlation among the context states (expressed as an *association rule*). A key innovation is the use of two separate *confidence values* with each context predicate, which permits both deterministic and probabilistic queries to be short-circuited in a uniform way.

2 **Energy-Efficient Unified Evaluation Algorithm of Collective Queries:** We develop a new, bottom-up query evaluation algorithm that separately computes the two different *confidence values* for each context node in the query graph, and then propagates these up in the query graph by using the association rules. We propose a novel metric, called *normalized expected change in confidence (NECC)*, based on these propagated confidence values, to dynamically determine a context evaluation sequence that balances acquisition cost, selectivity

[1]**Clo**ud-based **Que**ry Evaluation Framework, pronounced as 'cloak'

and coverage.

3 **Latency and Scalability Support:** To perform energy-efficient query optimization within specified latency bounds, we present an enhanced, *partitioned* operation of *CloQue*, which identifies and executes an appropriate partition of context queries in parallel. The result shows the parallelization strategy save up to 45% of entire evaluation time. Moreover, we show how *CloQue's* uses a shared cache that allows different partitions to intelligently re-use already evaluated context attributes, allowing *CloQue* to scale to a large participant base and to a large number of contextual predicates.

4 **Real life-Trace-based Evaluation:** By testing the performance of *CloQue* on two real-life datasets: (i) *MIT Reality Mining* and (ii) *MIT Social Evolution*, we demonstrate that: a) *CloQue* can achieve 50-60% reduction in overall energy overheads, compared to traditional approaches of either push-based sensing or simple short-circuited query optimization, without sacrificing query correctness; b) When applications allow modest tolerance in the query accuracy, *CloQue's* use of cross-individual correlation leads to significant additional energy savings (e.g., applications specifying a 4% error bound can sometimes reduce energy overheads by another 20%).

II. Related Work

CloQue views context determination as a process of query evaluation over mobile-generated data streams. This view is closely related to the fields of query optimization in databases, sensor networks and mobile sensing. We thus focus closely on a) query optimization in databases and sensor networks and b) energy efficient sensing on smartphones.

Our work is inspired by work on probabilistic databases [5], [16],as we respond to queries with sufficient confidence while acquiring data from a bare minimum set of sensors. However, our work differs from such work as: *a)* instead of relational algebra, our queries are boolean combinations of predicates expressed over time-windows of sensor data streams, and *b)* we adjust the selectivity probabilities of predicates continually *during* query evaluation. Accuracy of response in a probabilistic database might be a challenge, which has been dealt with in works such as [20], [3]. We use historical data to minimize on errors to a large extend.

For sensor networks, Deshpande et al. [7] proposed a model-driven data acquisition approach that models sensor data using Gaussian probabilistic models. More recently, Raza et al. [17] proposed a simpler and more practical alternative using linear models. However, these approaches do not use inferencing rules to refine the confidence intervals of predicates during runtime, nor do they consider a cloud-based implementation designed for mobile sensing.

In the area of mobile sensing, there has been a lot of interest in understanding group based activities [18], [6]. Groups can be useful in conserving energy required for sensing and processing. Once groups are determined, energy efficiency can be achieved using collaborative sensing where sensing task can be distributed across the different users in the group to reduce individual's energy consumption. Alternately, energy efficient can be achieved by performing on-phone query optimization, where if a certain criteria has been met, then sensing on the phone is terminated. Since both collaborative sensing and query optimization is similar to *CloQue's* approach, we discuss them here.

Query Optimization - To optimize query processing on a smartphone, the ACQUA framework [12] performs query short-circuiting by using a metric that normalizes sensor energy costs by the predicate selectivity. In contrast, the ACE framework [15] applies simple Association Rule Mining (ARM) techniques to discover context correlations from historical data. It uses probabilistic correlation to short-circuit the evaluation of mutually-exclusive contextual predicates, providing a 4-fold reduction in energy overheads. *CloQue* unifies these principles via a unified query short-circuiting framework, that jointly combines query selectivity, sensing cost as well as cross-context correlation to support multiple collective queries (both deterministic and probabilistic) executing over sensor data from hundreds of smartphones. *CloQue's* approach of using a centralized cloud-based query engine is based on the suggestion of a centralized sensing coordinator in [19], and seeks to shift the bulk of the processing and coordination load to the infrastructure, while requiring only light-weight and intermittent sensor processing on the individual mobile devices.

Collaborative sensing - the collaborative sensing approaches seek to share the burden of context computation across multiple phones. For example, Darwin [14] and SocioPhone [10] utilizes the microphones of multiple phones to accurately perform speaker identification, whereas CoMon [9] seeks to save energy by sharing the sensing and stream processing functions among a group of proximate smartphones. All of these approaches, however, relate primarily to the optimized estimation of a *shared context* (typically ambient parameters such as noise or temperature) among nearby devices. They do not use the notion of probabilistically short circuiting queries, which we use in our approach. A major concern with collaborative sensing can be privacy which can keep users away from collaborating. However, providing adequate incentives might help in larger participation [11]. However providing incentives or privacy are beyond the scope of this paper so we do not put effort on address them.

III. The CLOQUE System Architecture

CloQue employs a client-server architecture, with a centralized query processing engine responsible for coordinating the sensing and context collection tasks across a large set of mobile devices. Figure 1 describe's *CloQue's* functional architecture. We first describe each of the blocks in the architecture and then talk about how query is represented.

A. Functional Blocks

The *Smartphone Access Layer* handles all communications with the smartphones, such as issuing commands to an individual smartphone to evaluate and return a specific context attribute (using phone-embedded sensors) and receiving the results of such evaluated context.

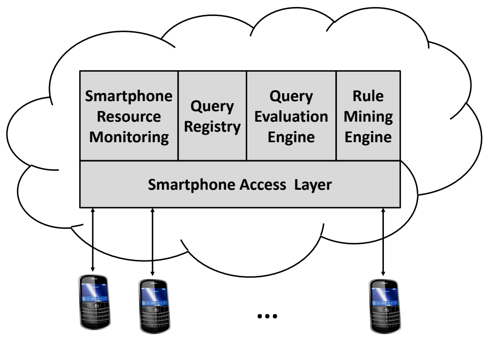

Fig. 1. The Overall Functional Architecture of *CloQue* .

The *Query Registry* allows different context-aware applications to issue continuous queries to the *CloQue* engine, and remove queries when they are no longer needed. Each query is a boolean combination of *context predicates*, with each predicate defined over one or more sensor streams (e.g., accelerometer, compass or microphone) of a single smartphone. When queried by the Smartphone Access Layer for a specific context, an individual phone evaluates and returns a boolean value, denoting whether the particular context is true or not (e.g, "user A is walking"= TRUE).

The *Resource Monitor* tracks the resource levels at each smartphone (e.g., its battery level and the energy costs of an individual sensor) by periodically querying each individual device, thereby maintaining an up-to-date value of the cost of evaluating each query predicate.

The *Rule Mining Engine* collects historical data about each individual and infers association rules from them, using standard ARM techniques, such as the *a priori* algorithm [4].

Finally, the *CloQue Query Evaluation Engine* (**QEE**) is the central coordinator that evaluates the continuous queries in the registry and sends the results to subscribing smartphones (see Fig. 1). In the rest of this paper, we focus on describing the QEE.

B. Context Query Representation

A query is a boolean combination of predicates in disjunctive normal form (DNF), modeled as a three-level tree, with the root node being the logical OR operator, the second level nodes representing the logical AND operators, and the leaf nodes representing the predicates. Figure 2 illustrates an example of a query. Each predicate p_j must be associated with at least one sensor; however, two different predicates can operate on data from the same sensor. In the probabilistic setting of *CloQue*, each query is also associated with a user-specific *confidence threshold*, and the

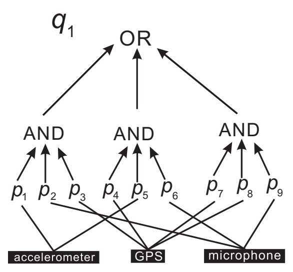

Fig. 2. An example of a query in DNF operating on sensors from a single smartphone. Each predicate p_i is a distinct predicate. The predicates p_1 and p_5 are both associated with sensor the accelerometer sensor. Similarly, the predicates p_3, p_4, p_7, and p_8 are associated with the *GPS* sensor. the predicates p_2, p_6, and p_9 are associated with the microphone sensor.

query is considered to be successfully evaluated to be true when the probability or confidence that the query evaluates to true exceeds this threshold. In the other hand, the query is considered to be successfully evaluated to be false when the probability or confidence that the query evaluates to false exceeds this threshold.

IV. THE *CloQue* QUERY EVALUATION ENGINE

The goal of the *CloQue* QEE is to evaluate the set of queries in the Query Registry, while *minimizing the energy consumption of the set of smartphones*.

A. Probabilistic Query Evaluation

CloQue's query evaluation uses several key ideas.

1 Each (context predicate) node in a query tree has two dynamically changing *confidence* values (both always varying in the range [0,1]): the *true-confidence* denotes the current probability that the predicate is true, and *false-confidence* denotes the current probability of the predicate being false.

2 Besides updating query node confidence values deterministically (by actually evaluating predicates on the smartphones), we use association rules mined from historical data to update the confidence values of query nodes, thereby exploiting historically-observed correlation across contexts.

3 The predicate nodes of all the queries are maintained in an ordered queue, with the evaluation order being recomputed, during *query evaluation* to increase the likelihood of short-circuiting of the DNF formulas of multiple queries.

CloQue uses *association rules* to capture the inter-dependencies and correlation among multiple context. An association rule consists of a head and a body. The head of a rule is a single predicate, while the body is a list of other predicates, such that head is true only if all

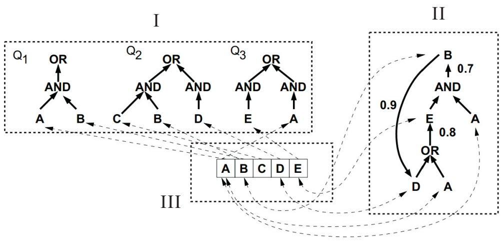

Fig. 3. Example of the 3 data structures used in query evaluation. The query set is I, rule set is labeled II, and distinct predicates list is III.

the predicates in body are true. Different rules with the same head are treated as a logical OR relationship—the head predicate is true if at least one of the bodies of the rules is true. Similar to conventional approaches, a rule is associated with a *support* (the fraction of historical data where the rule holds) and a $confidence$ (the fraction of historical data where the head of the rule is true, given that the body is true).

Query Data Structure The QEE maintains three data structures: the set of queries, the list of distinct predicates, and the set of rules. Note that a predicate can appear in multiple query leaf nodes and in multiple rules. Figure 3 illustrates an example of the three data structures where all predicates are associated with distinct sensors. For conciseness, we will use the sensor identifiers as the predicate identifiers.

The set of queries is represented as a forest of query trees. The two confidence values (*true-confidence* and *false-confidence*) for each context node are both initialized to zero. After evaluating a predicate at the associated smartphone, the true-confidence and the false-confidence of the related predicates will be set to $(1,0)$ or $(0,1)$ respectively depending on whether the predicate is true or false. The set of association rules is represented as a directed graph with two types of vertices: *Logical* vertices represent the logical AND/OR operators, while *predicate* vertices are identical to the predicate nodes in the query tree. Outgoing links from predicate vertices indicate query predicates (e.g., the *and* relationship in a rule such as "Meeting is over" if ("A is walking" and "B is stationary")), whereas edges leading to predicates denote implication relationships (e.g., "Meeting is over" $\Rightarrow$ "A is walking" (with $conf = 0.6$).

B. Predicate Ordering for Query Evaluation

The predicate list data structure specifies an order to evaluate the predicates in order to minimize energy consumption via short-circuiting. We first describe the processing logic assuming a hypothetical *baseline* approach, where the predicates are evaluated in a static or fixed order. The QEE evaluates the forest of queries in a bottom-up fashion, starting with the leaf nodes which are linked to the predicate list. These predicates are evaluated sequentially: the first predicate in the list is evaluated by querying the corresponding smartphone and retrieving the result, followed by *propagation of confidence values* (to be described shortly, in Section IV-C) via the set of available association rules. As a result of such confidence propagation, if any query has been satisfied (the query confidence threshold met), QEE generates an alert for the application. It then proceeds to evaluate the next predicate.

The *CloQue* QEE does not use a static predicate list, but instead uses a dynamically re-ordered predicate list (that is re-ordered after *each predicate is evaluated*), to reflect that confidence propagation can change the true-confidence and false-confidence values of predicates yet to be evaluated. *CloQue's* re-ordering algorithm is outlined in Alg. 1, and uses a new metric called *NECC* to balance several competing desires, preferring predicates that: *a)* have a high probability of short-circuiting; *b)* incur less energy cost to evaluate, *c)* affect a larger number of queries (higher *coverage*); and *d)* will resolve the maximum amount of uncertainty about other un-evaluated predicates. While the first two goals have been part of prior query short-circuiting frameworks, objectives *c)* and *d)* are unique to our scenario. Constraint *c)* arises from our consideration of a potentially large number of overlapping group queries, whereas constraint *d)* is similar to the concept of *maximal information gain*, and arises from the fact that the true and false confidences of predicates are correlated.

To capture objective d), we simulate the update propagation for the two hypothetical cases when a predicate z is true (t) and when the predicate is false (f), and sum the change in true- and false-confidence values over all the internal nodes (AND and OR nodes) in the query forest. Suppose there are m internal nodes $\{q_1, q_2, ..., q_m\}$. The change in confidence assuming predicate $z = t$ is,

$$\Delta C\mid_{z=t} = \sum_{i=1}^{m} \Delta C_t(q_i)\mid_{z=t} + \Delta C_f(q_i)\mid_{z=t} \tag{1}$$

where $\Delta C_t(q_i)\mid_{z=t}$ is the change of an internal node's $true\text{-}confidence$ and $\Delta C_f(q_i)\mid_{z=t}$ is the change of an internal node's $false\text{-}confidence$. The change in confidence assuming that the predicate is false, $\Delta C\mid_{z=f}$, is computed similarly.

We now defined the *normalized expected change in confidence* (NECC) metric as:

$$NECC(z) = \frac{P(z)\Delta C\mid_{z=t} + P(\neg z)\Delta C\mid_{z=f}}{cost(z)} \tag{2}$$

where $P(z)$ (similarly $P(\neg z)$) denotes the probability that predicate z evaluates to be true (or false). After computing this value for all the un-evaluated predicates, the QEE next picks the one with the highest NECC value.

C. Confidence Propagation Using Rules.

After the evaluation of a predicate at the smartphone, the *CloQue* query engine updates the confidence values in the query forest using the association rules. The query engine first propagates

Algorithm 1 QUERY EVALUATION LOOP

Input: A set of queries $Q = \{q_1, q_2, ...q_m\}$, a set of rules R, a set of energy cost $Cost$, evaluation period ω
Output: Generate alerts for each query that is satisfied

```
1: Let $H$ be the priority heap for the predicate list by using Eqn. 2
2: for every $\omega$ seconds do
3:     for all predicate $h \in H$ do
4:         calculate the NECC for predicate $h$
5:     end for
6:     heapify($H$)
7:     while empty($H$) is false do
8:         $z \leftarrow$ extractMax($H$)
9:         $val(z) \leftarrow$ evaluate $z$ at phone
10:        UPDATE RULE CONFIDENCE($R, val(z)$)
11:        UPDATE QUERY CONFIDENCE($Q, val(z)$)
12:        for all $q_i \in Q$ that satisfied do
13:            generate alert for $q_i$
14:        end for
15:        for all predicate $h \in H$ do
16:            calculate the NECC for predicate $h$
17:        end for
18:        heapify(H)
19:    end while
20: end for
```

the updated confidence values through the rule graph (note that these updates can change the confidence values of other predicates as well), and then propagates the updated confidence values up the query trees.

Confidence propagation is performed independently for the true-confidence and the false-confidence values. Let $C_t(u)$ and $C_f(u)$ denote the true-confidence and the false-confidence of a node u in either of the three data structures. The update logic is based on the intuition that the true-confidence of an OR-node is the maximum confidence of the true-confidence of its predecessors and the true-confidence of an AND-node is the minimum confidence of the true-confidence of its predecessors. (Conversely, the false-confidence of an OR-node is the minimum confidence of the false-confidence of its predecessors and the false-confidence of an AND-node is the maximum confidence of the false-confidence of its predecessors. For the rule graph where a predicate node can have incoming edges associated with a rule-confidence, the true-confidence of a predicate node given that its predecessor's true-confidence has been updated is the rule-confidence multiplied by the predecessor's true-confidence. Similar to false-confidence. The following update equation summarizes the bottom-up update logic for the true-confidence value of node v given

Algorithm 2 UPDATE QUERY CONFIDENCE(Q, z)

```
Input: A set of queries $Q = \{q_1, ..., q_m\}$
Output: Confidence of query nodes affected by $z$ will be updated sequentially
1: for all leaf nodes $l \in Q$ associated with $z$ do
2:     perform DFS traversal from $l$ updating the $t-confidence$ of nodes using Eqn. 3 (and its variant for
       upating $f-confidence$)
3: end for
```

Algorithm 3 UPDATE RULE CONFIDENCE(Q, z)

```
Input: A set of queries $Q = \{q_1, ..., q_m\}$
Output: Confidence of query nodes affected by $z$ will be updated sequentially
1: for all rule nodes $r \in R$ associated with $z$ do
2:     perform DFS traversal from $r$ updating the $t-confidence$ of nodes using Eqn. 3 (and its variant for
       upating $f-confidence$)
3: end for
4: for all predicate nodes $w$ updated by rules do
5:     UPDATE QUERY CONFIDENCE($Q, \omega$)
6: end for
```

each successor node u of node,

$$\mathcal{C}_t(u)^{(n+1)} = \begin{cases} \max\{\mathcal{C}_t(u)^{(n)}, \mathcal{C}_t(v)^{(n)}\} & \text{if } u \text{ is an OR} \\ \min_{\omega \in Pred(u)} \mathcal{C}_t(\omega)^{(n)} & \text{if } u \text{ is an AND} \\ \max\{\mathcal{C}_t(u)^{(n)}, \mathcal{C}_t(v,u) \cdot \mathcal{C}_t(v)^{(n)}\} & \text{if } u \text{ is a predicate} \end{cases} \tag{3}$$

where the superscript n and $n+1$ denote the time before and after one application of the update equation. (Similar equations for updating the *false-confidence* values are omitted due to space constraints.) The term $\mathcal{C}_t(v,u)$ denotes the confidence of an association rule. Note that for the rule graph, the update propagation only updates the true-confidence, as association rules only apply when its body is true. Alg. 2 outlines the algorithm for updating the confidence values of query nodes and Alg. 3 outlines the algorithm for updating the confidence values of the rule graph.

Example *Update propagation example.* Consider a simple example of the confidence update propagation. In the Figure. 4, there are three queries Q_1, Q_2 and Q_3 (with 0.8 confidence) to be evaluated and their query trees show in the forest respectively. Assume that we have the 4 association rules as follows,

$$\begin{array}{llll} (i) & E \Leftarrow A & (0.8) \\ (ii) & E \Leftarrow D & (0.7) \\ (iii) & D \Leftarrow B & (0.9) \\ (iv) & B \Leftarrow A \quad and \quad E & (0.7) \end{array} \tag{4}$$

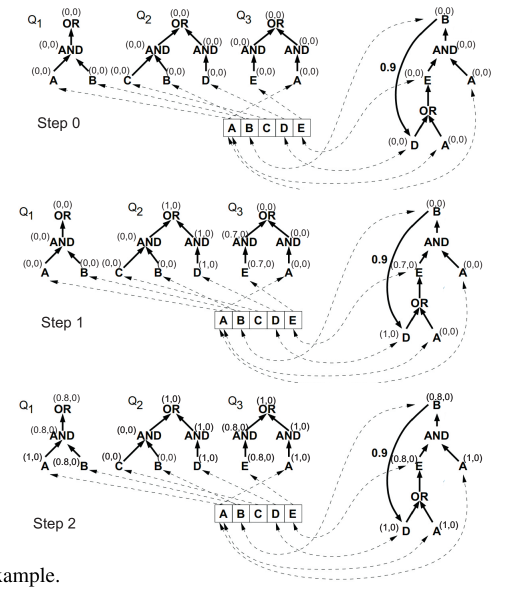

Fig. 4. Update propagation example.

For each rule, left-hand-side predicate and right-hand-side predicates are consequent and antecedent respectively. The number is the confidence of the rule. Assume that after using Eqn. 2 to sort the predicates, we get a predicate list: D⇒A⇒E⇒C⇒B. D is the top of the predicate list and B is the last. Step 0 is the beginning phase, all the $true\text{-}confidence$ and $false\text{-}confidence$ in queries and rules are initialed to 0. Since D is the top of the predicate list we send D to the associated phone for evaluation. If D evaluates to be true, then we apply Eqn. 3 to the queries and rules. In Step 1, for the rules, update $\mathcal{C}_t(D)$, the $true\text{-}confidence$ of the D from 0 to 1. Because there is a OR from D to E, so that update E's $true\text{-}confidence$ $\mathcal{C}_t(E)$ with $\mathcal{C}_t(D) \cdot 0.7 = 1 \cdot 0.7 = 0.7$, where the first 0.7 is the confidence of rule (ii). Then update the D in the Q_2, the D's parent is a AND so that its $true\text{-}confidence$ is the minimum $true\text{-}confidence$ of its child which is 1. Since E got updated in rules, so we continue to update E in Q_3. Similar to the propagation of updating D in Q_2, the $true\text{-}confidence$ of E's parent AND becomes 0.7. Further bottom-up update the $true\text{-}confidence$ of OR. From Eqn. 3, the $true\text{-}confidence$ of a OR is the maximum $true\text{-}confidence$ from its children, in this case, the $true\text{-}confidence$ of Q_2's root OR is 1 (> 0.8). Therefore, Q_2 evaluates to be true and output an alert. Afterwards send A to the associated phone for evaluation. If A is true(Step 2), in the rules first update A's $true\text{-}confidence$ $\mathcal{C}_t(A)$ to 1. From rule (i), the $\mathcal{C}_t(E)$ changes from 0.7 to 0.8. Then also update the E's $true\text{-}confidence$ in Q_3 from 0.7 to 0.8, as well as the E's parent AND. For B in rules, $\mathcal{C}_t(B)$ is the minimum between $\mathcal{C}_t(E)$ and $\mathcal{C}_t(A)$ so that it is 0.8. Then from rule (iii), D can be updated by B. However, it's not necessary

Algorithm 4 GREEDY PARTITION THE QUERIES

```
Input: A set of queries $Q = \{q_1, q_2, ...q_m\}$, a latency threshold $l$
Output: $n$ query partitions $P = \{p_1, ..., p_n\}$
1: for all $q \in Q$ do
2:     estimate the evaluation time $t_q$ of $q$
3: end for
4: sort $Q$ in descending order by evaluation time
5: create a new empty partition $p_1$ in $P$
6: set the evaluation time $t_{p_1}$ of partition $p_1 \leftarrow 0$
7: while $Q$ is not empty do
8:     while pop a query $q$ from $Q$ do
9:         estimate the evaluation time $t_q$ of the query $q$
10:        get the evaluation time $t_{p_i}$ of the latest created partition $p_i$
11:        if $t_{p_i} + t_q < l$ then
12:            add query $q$ to partition $p_i$
13:            $t_{p_i} \leftarrow t_q + t_{p_i}$
14:        else
15:            create a new partition $p_{i+1}$ with $q$ in it
16:            set the evaluation time $t_{p_{i+1}} \leftarrow t_q$
17:        end if
18:    end while
19: end while
20: sleep until being awaken
```

because $C_t(D)$ is larger than $C_t(B)$ already. Go ahead to update *true-confidence* of B in Q_1 (Q_2 is finish). Then update A in Q_1 and Q_3, so that the *true-confidence* of Q_1 becomes 0.8 and the *true-confidence* of Q_3 becomes 1. Therefore the Q_1 and Q_3 evaluate to be true and output alerts. The E, C, B in the predicate list we do not need to send to phones for evaluation so the energy saving is gained.

V. Parallel Evaluation in CloQue

The algorithms in Section IV evaluate the entire set of predicates (across all users) sequentially. The evaluation of the next predicate needs to wait for the previous predicate evaluation to finish. Consequently the overall latency of the query processing may become really high. The latency issue can get worse when processing thousands of predicates, expressed over hundreds to thousands of smartphones. The sequential processing is inadequate to deal with the cases if users require shorter response time. In order to alleviate this problem, *CloQue* partitions the given set of queries into m partitions, such that each partition is evaluated in the cloud independently in its own process. These processes can be executed in parallel and each process will handle one sequential evaluations. This strategy avoids single sequential processing that the subsequent predicates need to wait the current predicate evaluation to be finished.

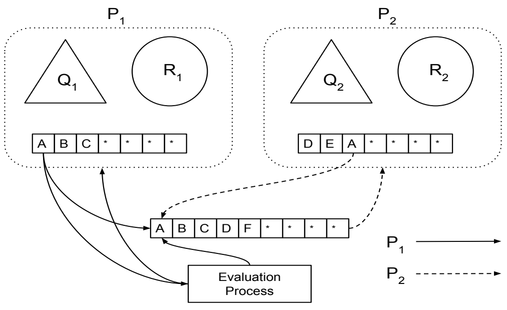

Fig. 5. Parallel evaluation in CloQue with a shared predicate result cache.

Algorithm 5 GREEDY PARTITION THE QUERIES WITH CLUSTERING

Input: A set of queries $Q = \{q_1, q_2, ...q_m\}$, a latency threshold l, a cluster parameter k
Output: n query partitions $P = \{p_1, ..., p_n\}$

```
1: use $k$-means to cluster all query $q$ in $Q$ into $k$ clusters $C = \{c_1, ..., c_k\}$
2: sort $C$ in descending order by estimated evaluation time $t_c$
3: create a new empty partition $p_1$ in $P$
4: set the evaluation time $t_{p_1}$ of partition $p_1 \leftarrow 0$
5: for all $c \in C$ do
6:     while pop $q$ from $c$ do
7:         estimate the evaluation time $t_q$ of the query $q$
8:         get the evaluation time $t_{p_i}$ of the latest created partion $p_i$
9:         if $t_{p_i} + t_q < l$ then
10:            add query $q$ to partition $p_i$
11:            $t_{p_i} \leftarrow t_q + t_{p_i}$
12:        else
13:            create a new partition $p_{i+1}$ with $q$ in it
14:            set the evaluation time $t_{p_{i+1}} \leftarrow t_q$
15:        end if
16:    end while
17: end for
18: sleep until being awaken
```

A. Context Caching across Partitions

Independent evaluation of each query partition runs the risk of redundant evaluations (and unnecessary energy overhead), especially if the same context predicate is common across multiple partitions. *CloQue* tackles this problem by caching predicate evaluation results in a cache that is accessible to *all* partitions across the concurrent execution environment. Figure 5 illustrates this caching approach, where P_1 and P_2 are two independent and concurrent processes evaluating the two query partitions Q_1 and Q_2 respectively. Each process then first checks to see if a shared context has already been evaluated (i.e., if the evaluation result is available in the cache)–if so, it simply reuses the cached context without contacting the smartphone. However, if the context is yet to be evaluated, the process (say P_1) then contacts the smartphone, and marks the context state in the cache as *"evaluating"*. In this case, another process (say P_2) attempting to evaluate the same context temporarily blocks, until the evaluation has been completed. If the evaluation is completed, the predicate data will be sending to the waiting process P_2. Each cached context will expire after a certain time. After expiration the next process asking this context will need to contact the smartphone to get newer data of the context. In addition, the cache is flushed after one run is finished, the next loop starts with an empty cache.

For user experience point of view, a query result is expected with in a certain time after user submits the query. Hence, we need to consider the response time or latency for evaluating the queries. The partitioning strategy assumes the existence of a latency threshold l, such that the total latency of evaluating *all* the queries, in any given partition of *CloQue*, cannot exceed l. To partition the queries, we currently adopt a *worst-case* partitioning strategy, by assuming that different queries have no predicates in common. In this case, the worst-case outcome requires the evaluation of *all* predicates of all queries in a partition, and incurs a latency equal to the sum of the latency of evaluating each individual predicate. Accordingly, we use a greedy bin packing approach, that packs queries into partitions such that the sum of the latency bounds of all the predicates within a partition is below l.

B. Greedy Partition the Queries

Algorithm 4 outlines this current approach, which does not factor in the reality that many predicates are strongly correlated and that the average evaluation latency of a partition may thus be significantly smaller than the worst-case bound. At line 1 - 3, it calculates the estimated evaluation time t_q for each query q in the query set Q. The estimated evaluation time t_q is calculated by adding up the evaluation time of all predicates in query q. Thus, t_q is the worse case of the evaluation time that evaluates all predicates in query q.

The evaluation time of a predicate we can empirically get from experiments. At line 4, sort each query q in descending order according to its estimated evaluation time t_q. Line 5 initializes the first partition p_1 in P. At line 6 - 12, keep adding query q from the sorted Q into the current partition p_i while the estimated evaluation time t_{p_i} of p_i does not exceed the threshold l. The evaluation time t_{p_i} is calculated by summing up the evaluation time of all queries in p_i. At line 16, the partition algorithm is put to sleep until next turn period.

C. Greedy Partition the Queries with Clustering

Algorithm 4 partitioning strategy assumes no common predicates in queries. However, in practical scenarios and especially when large numbers of queries involved, many common predicates exist over the queries. Another strategy is to cluster the queries with common predicates into the same process. Thus one process can execute more queries, and fewer types of predicates so that the total number of processes needed is smaller. In addition, clustering queries with common predicates into the same process also increases the chance that the queries being updated by rules. Because the most of the rules in that process are also associated with the common predicates.

Algorithm 5 outlines the clustering partitioning approach. At line 1 - 2, use k-means to cluster every query q into k clusters based on their common predicates.

Each query q can be presented as a vector $v_q = \{d^q_{s_1}, d^q_{s_2}, ..., d^q_{s_g}\}$, where $d^q_{s_i} = 1$ if query q has predicate(s) that associated with the sensor s_i otherwise $d^q_{s_i} = 0$. Sensors set $S = \{s_1, s_2, ..., s_h\}$ has all the sensors exist in Q. The k-means distance function using in our algorithm is $d(q_a, q_b) = \sum |d^{q_a}_{s_i} - d^{q_b}_{s_i}|$. Line 3 initializes the first partition p_1 in P. At line 4 - 13, keep adding query q from c in C into the current partition p_i while the estimated evaluation time t_{p_i} of partition p_i does not exceed l. At line 14, the partition component is put to sleep until next turn period.

In the experiment Section VII we demonstrate Algorithm 5 gains more energy saving and requires fewer processes to be created.

In order to optimize the usage of rules, it is always better to group “close” queries into the same partition. The “close” queries are those queries that more likely to share common predicates. If we can group “close” queries into the same process we can gain more efficiency in using the rules and contacting with the phones. However, the line 2 in Algorithm 5 sorts the clustering by their estimated evaluation time, which is not able to sort the clusters by their similarities. Hence, it is possible that some queries from two or more “not close” clusters that are grouped into one partition. That means this clustering sorting strategy has room to be improved.

The problem now is how to sort the clusters $C = \{c_1, ..., c_k\}$ as that adjacent clusters are “close” to each other. This sorting problem can be described similar to traveling salesman problem. Which each cluster is the cities, the order is the route go through all the cities. The problem is to minimize the sum of the distance among the order of the clusters. Given a list of clustering and the distances between each pair of clusters, what is the shortest possible route that go through each cluster exactly once? Travelling salesman problem is an NP-hard problem so we here use the greedy heuristic solution such as nearest neighbour heuristic to find out the orders of clusters. We random pick the first cluster, and then pick the next cluster which is nearest to the first cluster. Continue using this strategy and find out the order of all the clusters.

This improvement shows a positive result in experiment both in latency and energy consumption.

After the partition, each process evaluates its queries sequentially. Only one predicate is being evaluated at the time, while other predicates are waiting for the current predicate evaluation to finish. In order to further reduce the evaluation time, we introduce a parameter w that specifies how many predicates are being evaluated at the same time. If $w = 2$ then every process will evaluate 2 predicates from the sorted predicate list at the phones.

VI. CLOQUE: IMPLEMENTATION AND EVALUATION

In this section, we discuss the implementation of *CloQue* and evaluate its effectiveness. The evaluation goals were:

1 **Base Evaluation**: We demonstrate the effectiveness of *CloQue* in saving energy while preserving high levels of accuracy (i.e., in terms that our result matches reality). For energy, we measure the total energy consumption across all phones used in the query. We show, in Section VI-E, that *CloQue* reduces the overall energy consumption significantly (up to 70%, relative to a baseline push-based approach) with only a tiny (up to 4.7%) loss in accuracy.

2 **Sensitivity Analysis**: We understand the effect of various parameters on *CloQue's* performance. In particular, we changed the support and confidence thresholds used by *CloQue* and the length of the evaluation period. We show, in Section VI-F, that changing these parameters had significant impact on the energy consumption and accuracy of *CloQue*.

A. Implementation details

We have implemented a working prototype of *CloQue*, with the Query Evaluation Engine implemented in a perl-based engine, hosted on a Tomcat server with multiple threads being spawned to support concurrent partitions. The queries themselves were expressed as XML fragments and converted into a normalized form that allowed them to be stored as multiple rows in a MySQL database maintained by the Query Repository. The Smartphone access layer used sockets to interface with the specific phones, calling each phone to evaluate a specific context on demand. Besides this working prototype, we implemented a simple emulator that interfaces with *CloQue's* Smartphone access layer. This emulator allows us to evaluate *CloQue* using existing large-scale datasets, which were then replayed appropriately back to the Query Evaluation Engine, when it requested for a specific context from a specific phone.

B. Datasets Used – Reality Mining & Social Evolution

To perform this evaluation, we used two reallife datasets collected by researchers at MIT: the Reality Mining dataset [8] and the Social Evolution dataset [13].

The Reality Mining dataset contains the daily activities of more than 100 students, staff and faculties at MIT and was collected over a period of 2 academic years — 2004 and 2005. The data was collected from Nokia 6600 or similar smartphones and contains data such as the location of the user (celltower IDs), proximity to others (via Bluetooth), and activities performed (e.g., using any apps or making a call). In addition, the dataset granularity was between 10 to 120 seconds between each user update. We used up to 3 weeks of data from this dataset to test *CloQue*, while the underlying correlation rules and likelihood estimates were built from several months of (training) data.

The Social Evolution dataset contains sociometric information, which allows richer queries, as well as daily activities of 80 university dormitory residents from October 2008 to May 2009.

The daily activities were captured by cell phones every six minutes and includes proximity (via Bluetooth), location (via WiFi Scan), and call logs. The sociometric information was collected through surveys and included flu status, CCA activities, political opinions and so on. We used 14 weeks of data from this dataset during which the participants provided details about flu like symptoms. We used 4 weeks during this period for our testing.

C. Queries and Energy Profiles Used

To test all four variants, we used the same set of queries. These queries were designed to mimic three different scenarios of every day events of interest in workplace settings, corresponding to certain categories/types: *a)* **Interruptibility**: –both individual (e.g., *"Bob is at work and is not using his phone"*) and group-level (e.g., *"Bob and Jack are both at work and are not using their phones"*; *b)* **Group Semantics:** *"Bob, Jack, and Ross are together at the Cafeteria", "The Prism research group is having their group meeting"* ; and *c)* **Proximity Alerts:** e.g., *"Bob and Jack are near each other in any building"*; All queries used in the experiments are strictly in disjunctive normal form (DNF).

We created 3 different query sets (one for each scenario listed above) for both our datasets (Reality Mining and Social Evolution). Each query set used trace data from at least 20 different smartphone users who were chosen such that (1) they had interaction with other users, (2) their associated trace data covered a sufficiently long time period, and (3) chosen to best reflect the primary motivation behind the collection of each dataset. For example, for proximity monitoring in the Social Evolution dataset, we selected individuals who had exhibited flu-like symptoms in the recent past, or were known to be part of the same social club. Overall, our queries involved 30 distinct users for Reality Mining and 55 distinct users for the case of Social Evolution. We had a total of 63 unique predicates in the Reality Mining dataset and 99 in the Social Evolution dataset. These were divided amongst the six query sets with each set having atleast 20 unique predicates. Our experiments were run on a single Intel Core 2 machine with 8G of RAM running Debian Linux 7.0.

Both datasets did not have the energy consumption values of the sensor collection and computation performed by the phones in the dataset. Thus, we used the Monsoon Power Monitor [1] to measure the power consumption of a Samsung Galaxy S3 phone [2] (Exynos 4412 Quad, Quad-core 1.4 GHz Cortex-A9, 4.8 inches Super AMOLED capacitive touchscreen display) running on Android version 4.0.3 when performing the following operations; (a) Android's getCellLocation API() to determine the energy consumption of cell tower monitoring; (b) Bluetooth activation and scanning to determine the power for proximity monitoring; (c) Android's getSystemService() and getCallState() APIs to obtain the phone activity and call state, respectively. These operations match nicely with the data available in the two datasets.

D. Four Implementations Used for Evaluation

- ***Naive*** is a naive implementation of collaborative sensing where every sensor specified in a

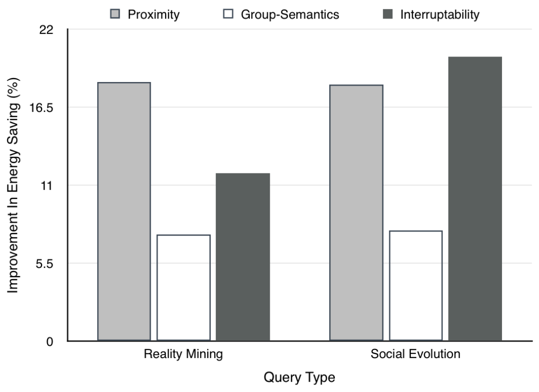

Fig. 6. Improvement in Energy Savings (%) Between *CloQue*$_{Full}$ and *CloQue*$_{NoRules}$

query is evaluated— i.e., the evaluation of a query set is not complete until all the predicates in each of the queries have been evaluated. We used a server-side cache to store the result of an evaluated predicate to avoid repetitive on-phone evaluations of the same predicate. The choice of *Naive* is similar to the baseline is chosen by [9] to show energy consumption when there is no collaboration between phones. This implementation has 100% query fidelity.

- ***Short-Circuit*** is an improved implementation of collaborative sensing where queries are evaluated in-order until a result is deterministically known — i.e, query processing will be short-circuited once a result is known. However, the order of query processing is fixed in a FIFO order. This is similar to the approach described in prior work, such as [16]. This implementation has 100% query fidelity as well.
- *CloQue*$_{NoRules}$ is a variant of *CloQue* that intelligently reorders queries but does not use the association rules and confidence propagation mechanisms described in Section IV-A. We used this variant as it also has 100% query fidelity and thus can be compared directly with the previous two implementations.
- *CloQue*$_{Full}$ is the full implementation of *CloQue* as described in Section IV. The main difference from *CloQue*$_{NoRules}$ is that the full version of *CloQue* uses the association rules and confidence propagation mechanisms to tradeoff a little accuracy for extra energy savings.

E. Results: Base Evaluation

Figure 7 and Figure 8 shows the total energy consumption, for both the datasets, for all the different implementations. The results show that, relative to *Naive* and *Short-Circuit*, the 100%

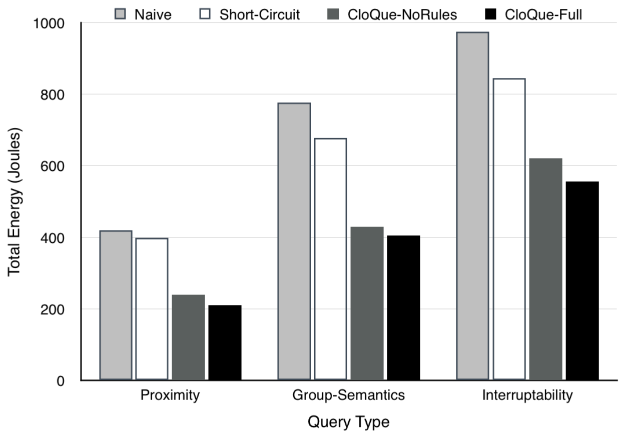

Fig. 7. Total Energy Consumption of the Four Variants on Reality Mining Dataset.

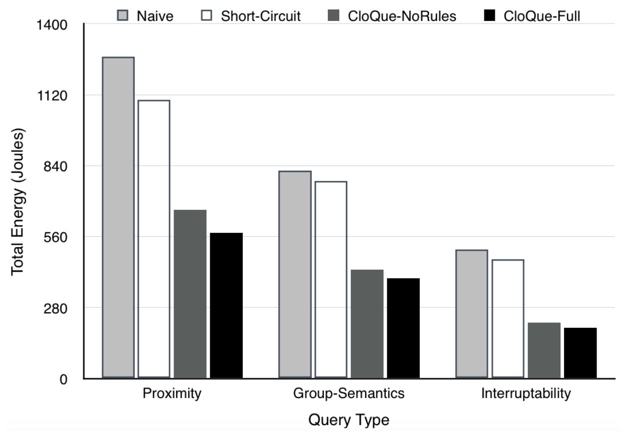

Fig. 8. Total Energy Consumption of the Four Variants on Social Evolution Dataset.

accurate version of *CloQue* ($CloQue_{NoRules}$) reduces the total energy consumption by about 50% with the full version of *CloQue* ($CloQue_{Full}$) doing even better than $CloQue_{NoRules}$.

Figure 6 shows, in more detail, the benefits of turning on the association rule engine in *CloQue*. In particular, we can save 10 to 20% more energy for proximity and interuptability type queries while saving about 8% more energy for group-semantics based queries. The accuracy obtained by $CloQue_{Full}$ is also between 95 to 96%. Thus, the full version of *CloQue* provides up

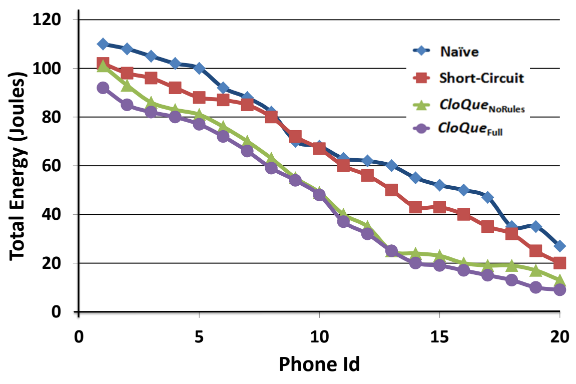

Fig. 9. Distribution of energy consumption over different phones/users. The data has been sorted in descending order of energy consumption.

to a 20% energy savings (depending on the type of query) for a modest 4 to 5% accuracy loss.

However, the energy improvements are not consistent across all the smartphone users. Figure **??** shows the energy consumption distribution for 20 smartphones for the four implementations. The results show that implementation has a similar energy consumption spread across the smartphones. However, the energy consumed at each phone by $CloQue_{NoRules}$ and $CloQue_{Full}$ is significantly lower than the other two implementations – with $CloQue_{Full}$ consuming about 12.08% less energy per phone, on average, than $CloQue_{NoRules}$. We plan to address this energy skew, in future work, by adjusting *CloQue's NECC* metric to account for residual battery capacity.

F. Results: Sensitivity Analysis

In this section, we investigate the effect of changing a) the support and confidence thresholds of the association rule engine, and b) the query evaluation period.

Support and Confidence Thresholds: Table I shows the effect of changing the confidence values of *CloQue's* (using the $CloQue_{Full}$ variant) association rule engine. In the Social Evolution dataset, we observe that reducing the confidence from 95% to 50% results in a 33.3% reduction in energy consumption but at the cost of an almost14.5% reduction in accuracy! On the other hand, increasing the support (values not shown) from 10% to 20% resulted in up to a 15% increase in energy consumption but with only a 2% improvement in accuracy. Hence, for these two datasets, *CloQue* was far more sensitive to the condfidence values than the support values. Overall, we

Conf. (%)	50	60	70	85	90	95
Reality Mining						
Accu. (%)	75.2	85.2	90.8	93.7	95.2	96.0
∑Energy (J)	413.3	455.3	485.9	506.8	515.0	523.5
Social Evolution						
Accu. (%)	83.3	87	90.8	92.0	95.2	95.3
∑Energy (J)	465.2	480.0	498.7	510.8	533.2	620.1

TABLE I
EFFECT OF CHANGING CONFIDENCE LEVELS

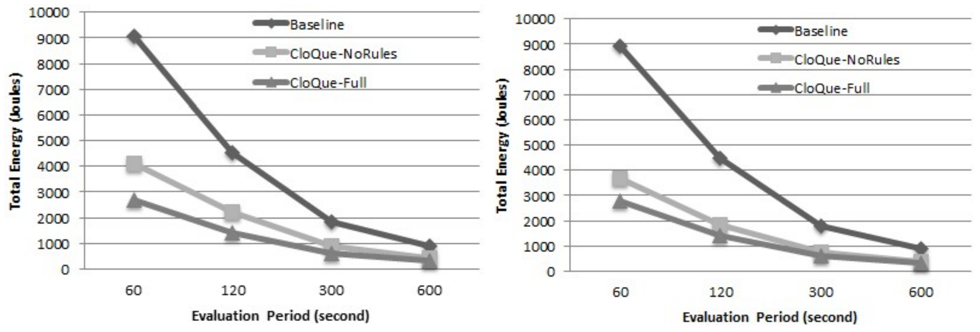

Fig. 10. Energy consumption for different evaluation periods on query set Q_A. The results for Q_B are similar and are omitted for brevity.

found, that for these two datasets, confidence and support values of 90% and 10% respectively gave the best tradeoff between energy consumption and accuracy.

Query Evaluation Period: As mentioned earlier, the query evaluation period is the frequency at which *CloQue* evaluates the continuous queries. If this value is too large, *CloQue* may take too long (or never) to detect that a query has been satisfied. However, setting this value to a low value can result in excessive energy consumption as *CloQue* re-queries devices excessively. To understand the impact of the query evaluation period on energy consumption, we measured the total energy consumption for evaluation periods of 60, 120, 300, and 600 seconds respectively. In Figure 10, we found that the energy consumption was proportional to the evaluation period — i.e., a 60 second evaluation period consumed ≈5x more energy than a 300 second evaluation period. However, in the best case, the latency of the 60 second evaluation period (i.e., the time to detect that a previously unsatisfied query has been satisfied) is also ≈5x lower than the 300 second evaluation. In our current implementation, we use a 300 second evaluation period as that seems to provide the best tradeoff between energy consumption and query latency for our current use cases.

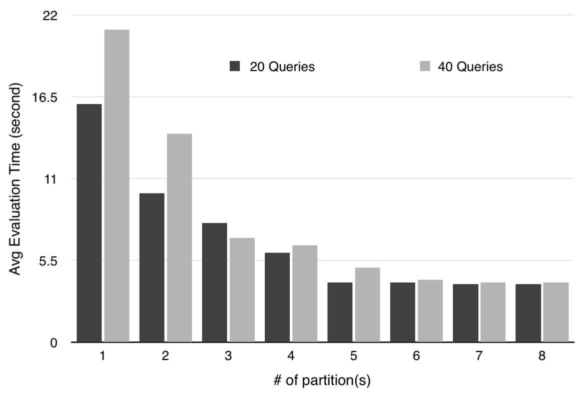

Fig. 11. Evaluation time of parallel partition

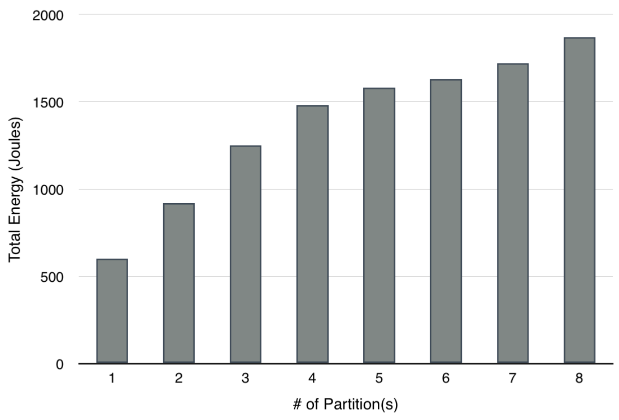

Fig. 12. Energy consumption of parallel partition

G. Parallel Evaluation and Latency

Figure 11 shows the average evaluation time on different number of partitions. We experiment two cases on Reality Mining dataset: partition 20 queries and partition 40 queries with Algorithm 4 partition approach. The queries include different query types, **Interruptibility**, **Group Semantics** and **Proximity Alerts**. We vary the latency threshold l to get the particular number of partitions.

Overall Figure 11 shows the more partitions the less totally evaluation time. For example, 2-partitions strategy costs 40% percent less time over 1-partition (no parallel evaluation). When

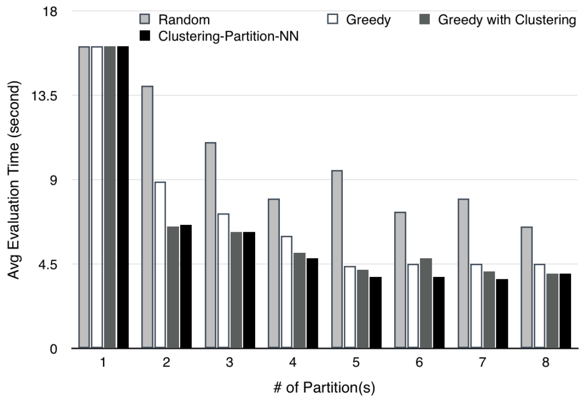

Fig. 13. Comparison of evaluation time by parallel partition

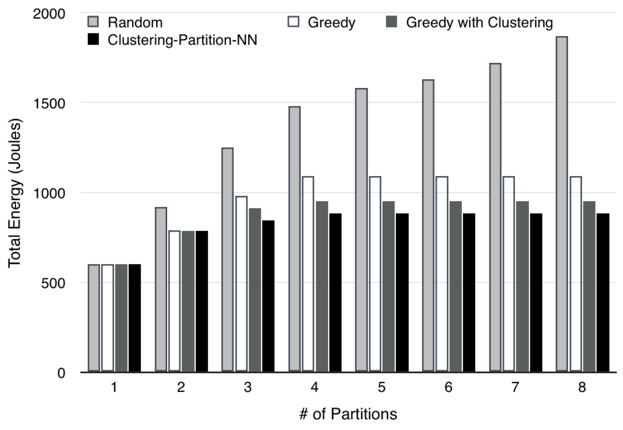

Fig. 14. Comparison of energy consumption by parallel partition

there are 6 partitions, the evaluation times drop, more than 72% compare to 1-partition(no parallel evaluation), from 16.2 second to 4.4 seconds and from 21.2 second to 4.6 seconds. However, we also notice that when the number of the partition increases to 6 - 8, there is not significant time saving generally. Also there is not significant difference in evaluation time between 20-queries and 40 queries.

Figure 12 shows the energy consumption of processing 20 queries on different number of partitions. In general, increase the number of partitions will also increase the total energy consumption. For instance, 2-partitions strategy costs nearly 150% energy consumption of 1-partition (no parallel evaluation).

Take Figure 11 and Figure 12 into account, we notice there is a trade-off in increasing the number of partitions. If the latency is critical in some cases we can have more partitions meanwhile we have to tolerate higher energy consumption. If the energy consumption is critical we should keep the number of the partitions small while tolerate higher latency.

In Section V we also proposed Algorithm 5 which is another approach to partition the queries into different processes. Figure 13 shows the differences of the average evaluation time over these four partition strategies, random partition, greedy partition, greedy partition with clustering and greedy partition with nearest neighbor chain sorting. Random partition is shown in the chart as the baseline. Overall all greedy partitions perform better than random partitions. When there are 4 partitions, greedy partition save up to 45% evaluation time compare to random partition. Greedy partition with clustering and greedy partition with nearest neighbor chain sorting are gain slightly more time saving than the regular greedy partition. In the experiment, we use 4 as the clustering parameter k.

In addition, Figure 14 shows a lot of energy savings when applying greedy partition with clustering approach over regular greedy partition, and greedy partition with nearest neighbor chain sorting perform the best. When setting the partitions number more than 4, the energy consumption of clustering partition is the same. The reason is there actually is no more than 4 partitions, all queries are already partitioned into the first 4 partitions without exceeding the latency threshold l. Clustering-Partition-NN performs better than Clustering-Partition because it finds nearer clusters and group them into the same partition.

VII. Conclusion

We presented *CloQue*, a cloud-based query evaluation system for optimizing the overall energy consumption of group-based queries across multiple smartphones. *CloQue* achieves this by using a unified query processing strategy that exploits both the variable acquisition cost of different sensors and the correlation among different phones arising from shared human activity context to change both the confidence intervals on query predicates and the confidence information in the association rules used to model correlation among phones. *CloQue* also includes a parallelization strategy to minimize query latency and to scale using cloud infrastructure. Our experiments using real traces from two different datasets shows that *CloQue* can reduce overall energy consumption by up to 60% with only a 4% loss in accuracy. In addition, we propose a greedy based parallel evaluation schema to reduce the overall evaluation time. In our future work, we plan to deploy the *CloQue* system on real phones and users, and evaluate *CloQue* in even more realistic online settings.

References

[1] Monsoon. http://www.msoon.com/LabEquipment/PowerMonitor/.

[2] Samsung galaxy s3. http://www.samsung.com/global/galaxys3/.

[3] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42, New York, NY, USA, 2013. ACM.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, Santiago, Chile, Sept. 1994.

[5] R. Cheng and S. Prabhakar. Managing uncertainty in sensor database. *SIGMOD Record*, 32(4):41–46, 2003.

[6] A. A. de Freitas and A. K. Dey. The group context framework: An extensible toolkit for opportunistic grouping and collaboration. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, CSCW '15, pages 1602–1611, New York, NY, USA, 2015. ACM.

[7] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, Toronto, Canada, 2004.

[8] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal & Ubiquitous Computing*, 10(4):255–268, Mar. 2006.

[9] Y. Lee, Y. Ju, C. Min, S. Kang, I. Hwang, and J. Song. Comon: cooperative ambience monitoring platform with continuity and benefit awareness. In *MobiSys*, Low Wood Bay, UK, June 2012.

[10] Y. Lee, C. Min, C. Hwang, J. Lee, I. Hwang, Y. Ju, C. Yoo, M. Moon, U. Lee, and J. Song. Sociophone: Everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. In *MobiSys*, Taipei, Taiwan, June 2013.

[11] Q. Li and G. Cao. Providing privacy-aware incentives for mobile sensing. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 76–84. IEEE, 2013.

[12] L. Lim, A. Misra, and T. Mo. Adaptive data acquisition strategies for energy-efficient, smartphone-based, continuous processing of sensor streams. *Distributed and Parallel Databases*, 31(2):321–351, May 2012.

[13] A. Madan, M. Cebrian, S. Moturu, K. Farrahi, and A. Pentland. Sensing the 'health state' of a community. *Pervasive Computing*, 11(4):36–45, Oct. 2012.

[14] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell. Darwin phones: the evolution of sensing and inference on mobile phones. In *MobiSys*, San Francisco, CA, June 2010.

[15] S. Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *MobiSys*, Low Wood Bay, UK, June 2012.

[16] J. Pei, M. Hua, Y. Tao, and X. Lin. Query answering techniques on uncertain and probabilistic data: tutorial summary. In *SIGMOD Conference*, Vancouver, Canada, June 2008.

[17] U. Raza, A. Camerra, A. Murphy, T. Palpanas, and G. Picco. What does model-driven data acquisition really achieve in wireless sensor networks? In *PerCom*, Lugano, Switzerland, Mar. 2012.

[18] R. Sen, Y. Lee, K. Jayarajah, A. Misra, and R. K. Balan. Grumon: Fast and accurate group monitoring for heterogeneous urban spaces. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, pages 46–60, New York, NY, USA, 2014. ACM.

[19] S. Sen, A. Misra, R. Balan, and L. Lim. The case for cloud-enabled mobile sensing services. In *Mobile Cloud Sensing Workshop (MCC)*, Helsinki, Finland, Aug. 2012.

[20] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: A new method for fast error estimation in approximate query processing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 277–288, New York, NY, USA, 2014. ACM.