The Case for Cloud-Enabled Mobile Sensing Services *

Sougata Sen, Archan Misra, Rajesh Balan
School of Information Systems, Singapore Management University
{sougata.sen.2012,archanm,rajesh}@smu.edu.sg

Lipyeow Lim
Information and Computer Science Department, University of Hawaii at Manoa
lipyeow@hawaii.edu

ABSTRACT

We make the case for cloud-enabled mobile sensing services that support an emerging application class, one which infers near-real time collective context using sensor data obtained continuously from a large set of consumer mobile devices. We present the high-level architecture and functional requirements for such a mobile sensing service, and argue that such a service can significantly improve the scalability and energy-efficiency of large-scale mobile sensing by coordinating the sensing & processing tasks *across* multiple devices. We then focus specifically on the problem of energy-efficiency and provide early exemplars of how optimizing query execution jointly over multiple phones can lead to substantial energy savings.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Cloud Computing*; C.3 [**Special-Purpose and Application-Based Systems**]: [Real-time and embedded systems]

Keywords

Mobile Phone Sensing, Power Management, Query Optimization

1. INTRODUCTION

The popularity of modern well-provisioned smartphones has made the use of cloud resources for computational augmentation (e.g., for real-time speech translation) less compelling. Instead, there has been an emphasis on building cloud services to collect and process sensor and context data produced by these smartphones, to enhance interesting applications such as maps, real time advertisements, etc. However, these services are presently tailored for *individualized* interactions; i.e., they treat each mobile device as an independent entity.

This paper posits that mobile devices are not just infotainment platforms, but are also powerful *distributed sensing* devices. In particular, continuous sensing and processing of a variety of device-embedded sensor streams (such as accelerometers, gyros & microphones) represents a distinct application class, that provides invaluable real-time context for a variety of consumer and enterprise applications. These applications typically require not just individual context, but *collective context* obtained by correlating and filtering the sensor streams from a *group of devices*.

We hypothesize that the ability of a cloud-based service to intelligently *coordinate* the sensing and context extraction tasks across *multiple mobile devices* will be crucial to such applications, and can result in significant energy and/or bandwidth savings. In the rest of this paper we explore the following specific questions:

- What are some of the compelling applications that require cloud-based mobile sensing services? We answer this in Section 2.
- What are the key building blocks of a cloud-based mobile sensing service, and the associated performance, scalability, privacy, and programming API requirements? We describe these in Section 3.
- For a particular application that requires sensor data, how can acquiring the data from multiple phones help to save energy? How does the tradeoff between sensing and communication energy overheads determine the likely benefits of distributing the sensing and context computation tasks across multiple devices, and how does this benefit vary between different commonly-used sensors? We address these questions in Section 4.3.
- What benefits does using a cloud-based service (where there can be a central "coordinator") have over more autonomous query optimization solutions? We explore this issue in Sections 4.4 & 4.5.

2. MOTIVATING APPLICATIONS

Before delving into the proposed architecture, we describe 3 motivating applications (each of these is a real world problem that would benefit from our proposed cloud-based architecture) to illustrate the use of group context to achieve a useful end result. Table 1 summarizes the applications and lists the sensors needed for each application, as well as a few additional attributes that will be explained shortly.

Airport Flight Boarding: A common problem for airlines is identifying how long it will take all passengers to board a particular flight. If the airline underestimates the time, the flight might

*This work 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.

MCC'12, August 17, 2012, Helsinki, Finland.

Application	Sensors Needed	Membership Set	Shareable Sensors?
Airport Flight Boarding	Wi-Fi, Accel, Compass, Microphone	Explicit	No
Dynamic Calendar	GPS, Wi-Fi, Accel, Gyro, Microphone	Explicit & Implicit	Yes
Meeting Status Indicator	Wi-Fi, Microphone, Accel, Light	Explicit	Yes

Table 1: Characterizing Different Applications

leave late, incurring both passenger wrath and possible additional gate charges. If the time is overestimated, the gate might be blocked longer than needed, leading to unnecessarily low utilization. Obtaining an accurate boarding time, however, is quite hard as it requires knowing where all the passengers of the particular flight are currently located and their ETA to the boarding gate.

If our proposed cloud-based sensing service was available, we could imagine building an airport boarding application that provides *situational awareness* of each passenger that needs to board a particular flight. The application is provided with the estimated departure time of the flight, the departure gate, and the list of passengers that need to board. It then continually tracks each passenger and determines a gate ETA for each passenger. However, to do this accurately and efficiently, we need to use sensor data from *both* the passengers themselves and other people in the vicinity of such passengers.

For example, using the Wi-Fi sensor in the phone of the passengers if it is found that the passenger is near the security gate, then the cloud service can use the Wi-Fi, accelerometer, and compass data of nearby phones to determine the security queue length and rate of progress. It can then determine how long it would take the passenger to reach the required gate (from their current location). The application can also use sensor data from multiple phones to estimate how many passengers are sitting or standing (using Wi-Fi and accelerometer data) at the departure gate, and how noisy or chaotic (using microphone data) the gate area is. Using such real-time tracking, the application can, for example, send boarding alerts to passengers who are far from the gate or even dynamically update the plane's "gate pushback" time.

Dynamic Calendar: A common requirement for university students is the need to schedule many impromptu group meetings to discuss class project requirements. However, the dynamic schedule of their group members, coupled with the transient availability of meeting places (study rooms, couches, etc. are rarely empty near major semester deadlines), makes scheduling these impromptu meetings a chore. Hence, students could really benefit from a dynamic calendar application that continuously monitors the movement of people and the availability of meeting locations to dynamically recommend a time and place for project meetings.

More specifically, the application, using our cloud services, can monitor the movement pattern & location of meeting participants (using GPS, Wi-Fi, accelerometer, and gyro data) to dynamically update the "expected meeting start time" (when $>$ x% of the participants might be physically present). It can also monitor the location of other peers (using GPS and Wi-Fi data) coupled with acoustic levels of potential meeting spaces (using microphone data) to also suggest "quiet & available" meeting spaces to the participants.

Meeting Status Indicator: In enterprise environments, an executive assistant may benefit from a proactive "Meeting Status" application to get real-time updates of an executive's schedule. Using our cloud service, we could build an application that monitors the location and current physical activity (sitting, standing) (using Wi-Fi and accelerometer data) of the participants of a particular meeting.

It can then combine this participant data with the ambient light and sound levels of the meeting room (using microphone and light sensor data) to detect when a meeting has truly ended (vs. when a single participant steps out briefly to take a phone call).

The examples are, by no means, exhaustive. They, however, help illustrate a few important points:

- **Query Logic:** The application logic can be viewed as a complex query (consisting of a mix of conjunctive (AND), disjunctive (OR) & negation (NOT) predicates or conditions), defined over *multiple* sensor streams, associated with *multiple* devices. For example, the "Meeting Status" application would use a query along the lines of "If (Person_A = 'standing' AND Person_B = 'standing' AND light_level='high' AND ambient_sound = 'conversation') $\rightarrow$ meeting='ended"'.
- **Membership Set**: In some cases, the set of users (or devices) that needs to be sensed is explicitly defined–e.g., only the checked-in passengers for a specific flight. In some other cases, the set of users may be implicit and non-enumerable. For example, to check whether a specific space is occupied, it may be sufficient to obtain the location of only a small subset of students.
- **Sensor Sharing**: In many cases, the query predicates will require a sensor stream from a *specific* mobile device. For example, to know whether a meeting participant is sitting, we need to sample the participant's accelerometer sensor. When query semantics refer to *ambient context*, the sensor streams may, however, be fungible. For example, to obtain the ambient light/sound levels in a meeting room, the application does not require the microphone and light sensor readings from all participants – it can load-balance this task among the participating devices. (Another example is the problem of estimating the progress rate of the security queue, in our airport flight boarding application.)

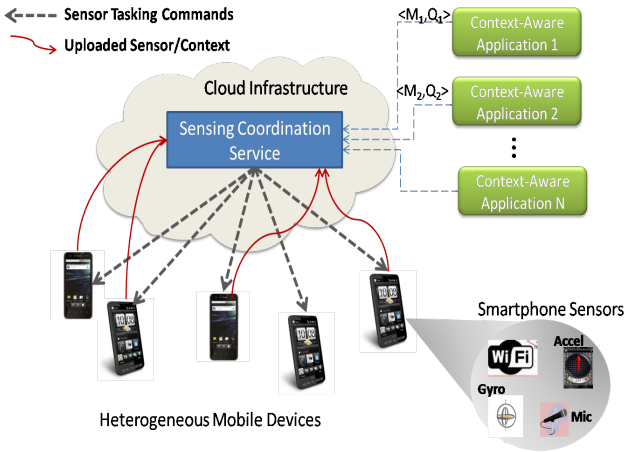


Figure 1: High-Level Architectural Components

3. PROPOSED CLOUD ARCHITECTURE

The motivating applications illustrate our proposition: there are a variety of consumer and enterprise applications (whether running on a backend server or on a consumer mobile device) that can leverage upon mobile sensing, especially when conducted over a group of mobile devices, to infer collective or group context. One naive approach is to have each application directly communicate with all

available mobile phones, retrieve *all the relevant sensor streams* from each of these phones, and then perform the query processing locally. This approach is impractical for two reasons: First, it requires transmission of the same phone sensor data to multiple application end-points, increasing the phone's communication overhead. Second, it fails to harness the significant energy efficiencies that may be achieved (shown in Sections 4.4 & 4.5) by optimizing queries over multiple phones *jointly*.

Our proposal, as illustrated in Figure 1, is a cloud-based *Coordination Service* for large-scale, continuous mobile sensing. Different applications connect to this service and describe their information requirements as a tuple $\langle M, Q \rangle$, where M denotes the set (explicit or implicit) of mobile devices/sensors that are needed to satisfy the query and Q specifies the query processing logic. This coordination service then takes this entire collection of information requests (i.e., the set of $\langle M, Q \rangle$ tuples) and then continuously *optimizes, on a global basis (across all queries), the tasking* of individual sensors on individual phones.

The four key requirements of our architecture are:

1. Significantly Reduce Energy & Bandwidth Costs: The key insight underpinning this approach is that the conventional push-approach, where each mobile phone/sensor simply pushes its sensed value (or derived context) to the Coordination Service, is unnecessarily wasteful. Instead, as we show in Section 4, substantial savings may be achieved, with no impact on query accuracy, by having the Coordination Service adopt a more proactive *pull-based* approach, where it actively and continually adjusts the sensing tasks of different smartphones and the retrieval of sensed data.

2. Scaling Both Sensors and Applications: The architecture needs to scale to thousands of mobile devices providing large amounts of continuous sensor data to thousands of applications. We plan to address this problem by a) carefully partitioning the phones into activity groups (based on locality, temporality, or other attributes) to maximize the data reuse within groups, and b) using the coordination service to reduce the sensor acquisition load on individual mobile devices by combining the requirements of multiple applications.

3. Avoiding Privacy Leaks: Our service short circuits query processing to achieve energy and bandwidth savings across devices. For example, a query of *gps_a = indoors AND gps_b = outdoors* would be stopped if a's GPS sensor is queried and the result is found to be not indoors. An adversary who has access to device b (but not access to the application making the query or the cloud service itself) could potentially infer the GPS state of a by monitoring when the service queries b for a reading (implying that a is indoors). We shall explore various policy-based mechanisms (e.g., the ACF framework [5]) to support scalable protection against such implicit context leakage.

4. Providing an Easy To Use Programming API: A common problem with distributed sensing environments is the complexity required to program applications to use those environments. However, recent work shows promise in developing programmer friendly environments even for complicated sensing tasks. For example, Balan et. al [2] demonstrated a syntax language that allowed even novice programmers to quickly and effectively partition large unfamiliar monolithic applications for use in a distributed dynamic partitioning environment. We plan to explore similar programming abstractions for our cloud service.

4. GLOBAL COORDINATION FOR ENERGY-EFFICIENT MOBILE SENSING

For the rest of this paper, we focus on our first requirement: *reducing energy costs for mobile sensing*. We first introduce a previously suggested model of continuous query optimization that helps to save energy by avoiding the unnecessary overheads of transferring data wirelessly from a sensor source to a querying destinatin node. We then explore the various types of enhancements that would be needed, within this framework, to address some of the unique challenges/opportunities for large-scale multi-phone sensing.

4.1 Continuous Query Optimization

Our basic query optimization framework is inspired by ACQUA [7], which optimizes the sequence in which different predicates of a complex continuous query are evaluated, taking into account both *selectivity statistics* and the cost of *data sensing/acquisition*. We shall show that the basic procedure defined in [7] (which considered the case of multiple sensors wirelessly connected to a *single* mobile device) requires significant enhancements to address our problem of large-scale mobile sensing, where multiple queries are being executed simultaneously over multiple mobile devices.

As a brief overview, ACQUA focuses on a complex stream query Q consisting of a set of conjunctive or disjunctive predicates $\mathcal{P}$, where predicate P_i at the leaf of the 'query graph' refers to a particular sensor S_i. For example, consider a query "State⟨Accel⟩= 'walking' and Location⟨Wi-Fi⟩='classroom', where the activity state is deduced from an underlying accelerometer activity stream and location is deduced by using fingerprinting-based techniques applied to Wi-Fi scans. ACQUA's query semantics are defined by two parameters: the *Evaluation Period* ω specifies the periodicity of evaluation, while the 'tumbling window' size of each predicate determines the interval of sensor data that it requires. In the simplest case, each tumbling window is the same size as ω (see Figure 2 for an illustration of such a query).

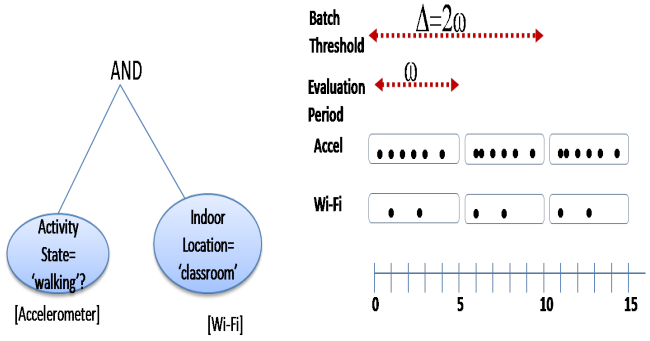
The ACQUA algorithm tries to balance the communication cost (denoted by C_i) of retrieving the data from sensor S_i for evaluating the predicate P_i, with the current selectivity characteristic (S_i) of the predicate; as a high-level overview, it uses a Normalized Acquisition Cost (**NAC**) metric defined by $NAC = \frac{C_i}{P_i}$ and then evaluates the predicates in ascending order of NAC values (in effect, preferring to evaluate cheaper and more selective predicates first, so as to abort the query processing early).

The basic logic in [7] assumes that the sensor streams are acquired and predicates are evaluated every ω secs. However, if the query can tolerate the resulting evaluation latency, better energy efficiency is achieved if the sensor data is batched and transmitted once every *Batch Threshold* (denoted by Δ) secs–i.e., if the evaluation for $\frac{\Delta}{\omega}$ evaluation periods is performed simultaneously (see Figure 2). In Section 4.3), we shall see Δ leads to interesting, non-obvious trade-offs that have not been previously considered and that require us to enhance the query optimization logic.

4.2 Micro-Experiments

In the rest of this section, we shall use simple micro-experiments (on one/two mobile phones) to quantify and illustrate some of the ways in which the mobile context sensing process (consisting of sensor sampling, stream processing and result transmission) can be optimized. The use of a cloud-based coordinator gives rise to two interesting possibilities:

- The sensing and processing steps may occur at different locations –e.g., the sensing may take place on the mobile device, while the actual processing occurs on the cloud-based coordinator.
- In scenarios where collaborative sensing is possible, the sensing task itself may take place on a *surrogate mobile device*.



Sample 'conjunctive' query over (Accel, Wi-Fi) sensors. The figure shows how a result is generated every ω secs, and how multiple answers are batched & transferred to the cloud every Δ seconds.

Figure 2: Continuous Query Processing

The specific choices made in query processing can be represented as a tuple $(loc1, loc2)$, where $loc1, loc2$ indicate, respectively, the location where sensing and processing occurs and can assume the value 'L' (local) or 'S' (server/ surrogate). Thus, (L,S), for example, denotes an approach where the sensor stream is generated on the mobile device, but the processing occurs on a cloud server.

To perform our scaled-down conceptual studies, we set up a micro-test environment similar to Figure 1. We used two Google Nexus One phones for our studies and the PowerTutor software [1] for our power measurements.

We used 3 representative sensors: (1) accelerometer (sampled at 30 Hz), with processing to detect {walking, sitting} events over a 'frame' of 5 secs; (2) GPS (sampled at 2 Hz), with processing once every 5 secs to detect 'location change of 5 meters'; (3) Wi-Fi (scanning performed once every 5 secs (0.2 Hz)), with the scanned AP RSSI readings used to compute indoor location based on prior RF fingerprints.

4.3 Characterizing the Computation vs. Communication Tradeoff

We first focus on the single case of a *single query*, running on a single smartphone and investigate the questions:

a) What energy benefits might there be, for different sensors, of offloading the sensing or computation tasks to an alternate device?

b) How are the costs affected by the query evaluation period ω and the batch duration Δ?

In this case, all the energy/power costs were measured on the primary phone, while the peer (surrogate) phone provided sensing data (for the (S,L) and (S,S) cases). All communication was orchestrated through our Coordinating Server; the Wi-Fi radio on the phone was used for communicating with the server (besides being used as a location sensor).

We first study, in Table 2, the energy consumption for the 3 different sensors (Accel, GPS, Wi-Fi) for the 4 different location choices (of sensing & computation locations), for a chosen value of $\omega = \Delta = 5$ (with the Wi-Fi interface being 'always on'). We can observe that the benefit of offloading either the sensing (to a cooperating peer) or the computation (to a cloud node) clearly depends on the energy overheads of the sensor. For GPS (whose power consumption is known to be the highest among sensors & around 250mW), transferring the sensing to a cooperating peer can achieve 87% lower energy overhead, while the benefit of offloading the processing is marginal. Conversely, for the accelerometer, performing both the sensing & computation locally on the mobile node is potentially fine, as any energy savings achieved in offloading computation are negated by the power consumed in transferring the larger volume of data via Wi-Fi (each $\omega = 5$sec corresponds to 150 accelerometer samples).

Sensor	Avg. Power (mW)			
	(L,L)	(L,S)	(S,L)	(S,S)
Accel.	41.4	40.1	36.6	36.5
GPS.	308.2	305.7	37.5	36.7
Wi-Fi	50.1	73.8	39.3	37 .1

Table 2: Sensor Energy Characteristics

To further understand the trade-offs involved, we considered a 'smarter' processing strategy, where the Wi-Fi radio was normally turned 'off' and turned on (for communicating with the server) only at the end of a batch duration Δ. After the query processing and communication is completed, the Wi-Fi radio is turned off again. Figure 3 plots the average power consumed (measured by dividing the total energy by the experiment duration of 30 mins) for each of the 3 sensors, for $\Delta = \{5, 15, 60, 300\}$ secs. Note that, in our experiments, the Wi-Fi radio takes ∼8-10 secs after turning on to establish an IP link and then transmit the raw or processed context predicates to the Coordination Server. Accordingly $\Delta = 5$ secs corresponds to the case where the Wi-Fi radio is 'always on'.

We note that average power is markedly higher (for the accelerometer & GPS sensors) when $\Delta = 15$ secs. This occurs because, due to the 'turning on' overheads, the Wi-Fi radio effectively runs at an $\approx 100\%$ duty cycle, but also additionally incurs the cost of 'IP connectivity' establishment once every 15 secs. For larger values of Δ, this start-up cost is amortized by the savings achieved by turning the Wi-Fi radio off, resulting in *dramatically lower* average power consumption. For Wi-Fi and Accelerometer, large values of Δ can result in energy savings of $\approx 50-75\%$.

Note that the query processing logic (namely, the order in which different sensor/predicates are conditionally evaluated) depends critically on this 'average power' (this defines the term C_i in Section 4.1). Also note that our results are scenario-dependent: clearly, the cost characteristics will vary, for example, if 3G/LTE is used instead of Wi-F or if the radio interface is already 'on' for other communication tasks. Accordingly, our key takeaway is that:

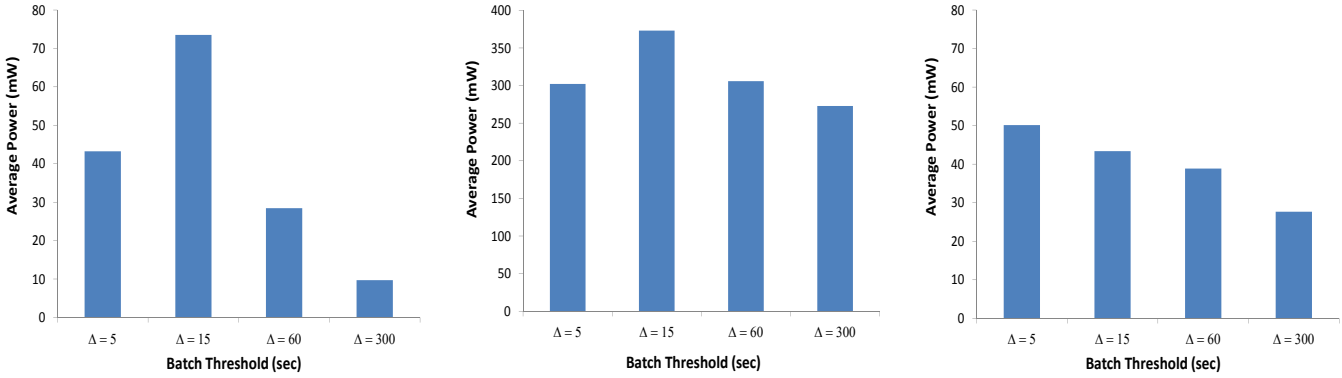
- Computing the 'acquisition cost' term C_i requires careful understanding of the sensor sensing vs. computation cost, as well as the characteristics of the radio interface.
- To the extent permitted by the query latency tolerance limit and the smartphone's 'normal' traffic load, using a large value of Δ allows us to turn the radio off and significantly reduce the energy overheads.

The use of large Δ values (multiples of the evaluation period ω) has implications for an enhanced version of ACQUA. For one, while different evaluation instants could have different selectivity properties (P_i), a larger ω implies a batched transfer process; hence, the query optimization logic must now consider *aggregate* selectivity and cost metrics.

4.4 Single Smartphone Query Optimization

To now understand the benefits of query optimization, we next explore the case of *multiple executing queries, but pertaining to the same smartphone.* We consider two very simple, intuitive queries (one conjunctive, one disjunctive):

- *'User is walking* $\langle Accel \rangle$ *& Location in Lab* $\langle Wi-Fi \rangle$*'*



a) Accelerometer (L,L) b) GPS (L,L) c) Wi-Fi (L,L)

Experiments where the Wi-Fi is turned on (for data transmission) every Δ secs. However, due to power on & off transients, the Wi-Fi radio stays on all the time for $\Delta = \{5, 15\}$secs.

Figure 3: Energy Costs with Different Δ (Dynamic Activation of Network Interface)

Avg. Power (mW)	Naive	'OR' Query	'AND' Query
$\Delta = 5$ secs	74.5	65.8	38.8

Table 3: Energy Savings from Query Optimization

Avg. Power (mW)	Naive	Optimized (independently)	Optimized (with NAC_{mod})
$\Delta = 5$ secs	136.5	136	99.34

Table 4: Multi-Query Optimization on Multiple Phones

- *'User is walking $\langle Accel \rangle$ || Location in Lab $\langle Wi-Fi \rangle$'*

We restrict our results to the case (L,L) with $\Delta = 5$ secs (results for other cases are qualitatively similar). To study the impact of using our query optimization strategy, we assumed that *Pr*(user walking)= 0.2 & *Pr*(user in Lab)= 0.8 throughout the experiment and computed the *NAC* values for each sensor accordingly. (In reality, these probabilities would be predicted using models built on historical context data). We then ran the query optimization logic for these queries and compared the total energy consumed vs. a 'naive' approach, where both sensor predicates were always computed and transmitted to the cloud. For the 'AND' query, we expect the Accel sensor to be queried first (as it has a higher probability of resulting in a 'false' predicate); the selection is reversed for the 'OR' query. Table 3 lists the observed average power consumption values. We can see that a cloud-based query optimization algorithm (for a single phone in isolation) results in about 60% energy savings for the 'AND' query (where the cheaper accelerometer sensor is evaluated first) vs. only about 12% for the 'OR' query. These numbers would obviously vary, depending on the selectivity probabilities for each of the predicates (which, in turn, would depend on an individual's activity statistics).

4.5 Multiple Queries on Multiple Phones

Having established the facts that *a)* different sensors have different communication vs. sensing overheads (and that these overheads depend on the choice of where sensing and computation is performed), and *b)* using such different costs can help optimize queries related to a single smartphone, we now consider the optimization issues that arise in the Coordination Service for the case of multiple queries running on multiple phones. We especially look at the situation when each individual query involves predicates (sensors) from different phones, *with the same sensor (on a specific phone) being involved in multiple queries*.

To illustrate the possibilities, consider two queries (defined over phones A & B) as follows (with Pr(A walking) =0.4 & Pr(A in Lab) = Pr(B in Lab) =0.2).

- **Q1**: *'User A is walking $\langle Accel(A) \rangle$ & User A's Location in Lab $\langle Wi-Fi(A) \rangle$'*
- **Q2**: *'User A is walking $\langle Accel(A) \rangle$ & User B's Location in Lab $\langle Wi-Fi(B) \rangle$'*

In this case, if each query is viewed *independently*, we may end up evaluating the 'Wi-Fi' predicates in Q1 &Q2 first. However, it is easy to see that evaluating user A's 'Accelerometer' predicate may, in certain cases, be more effective–if this predicate is false, it helps to abort *two queries* simultaneously.

While we do not dwell on the precise optimization enhancements in this paper, one heuristic for tackling this problem is to modify each predicates' NAC value by dividing it by the number of queries in which it appears: i.e., $NAC_{mod} = \frac{NAC}{N_Q}$ where N_Q is the number of independent queries having the same predicate. This effectively biases the optimization logic to consider the overlap of predicates among multiple independent, concurrently running queries. We ran the basic and enhanced query optimization logic on two phones (corresponding to the users A & B) for the conjunctive queries specified above. Table 4 plots the comparative avg. power consumed by this enhanced logic vs. that consumed by independent query optimization and the baseline 'naive' approach.

Our results show that optimizing each query independently actually results in almost no savings compared to the baseline approach (even though the independent optimization previously suggested energy savings of as much as 60%, when all queries pertained to a single phone). On the contrary, optimizing the queries jointly results in almost 30% savings in energy. The reason for the poor performance of an independent optimization strategy lies in its failure to consider the statistical correlation *across queries*: in the independent approach, where the Wi-Fi sensors for A &B are sensed first, the Accel sensor for A will end up being sensed and evaluated if the corresponding predicate fails to terminate *either* query. The result illustrates the unique query optimization challenges and opportunities that must be considered by our proposed cloud-based Coordination Service.

5. RELATED WORK

Academic work that aligns most closely to our cloud-assisted mobile sensing framework lies principally in two areas: *continuous sensing* and *collaborative processing*.

Continuous sensing approaches improve the energy efficiency of continuous sensing on a single device, primarily by dynamically adapting individual sensing parameters (e.g., Jigsaw [4]), activating more expensive sensors only if less energy-hungry sensors satisfy certain triggers (e.g., EEMS [9]) or by shifting computational tasks between the device and the cloud (e.g., Kobe [3] or SociableSense [10]). The CasCap framework [10] suggests the use of a device clone on the cloud to better coordinate the sensing and transmission activities of a mobile device, by using appropriate global context. All these approaches focus on a single mobile device and not on optimizing across *multiple* mobile devices.

A limited set of recent approaches have explored the *collaborative processing* paradigm, where multiple mobile devices share or coordinate their sensing activity. The Darwin mobile sensing framework [6] utilizes the sensing capability of multiple proximate phones by building and exploiting distributed classifiers to overcome the inaccuracies and errors of a single mobile device. The ErDos framework [8] exploits collaboration among multiple nearby devices to load-share the sensing burden (e.g., round-robin sharing of GPS sensors) of *shared ambient context*. In contrast to these approaches that focus primarily on ambient context, we aim to optimize more complex queries that are not just about ambient context and that are not limited to a set of proximate mobile devices. Moreover, we propose a centralized cloud-coordinated model that optimizes multiple queries jointly.

6. CONCLUSION AND FUTURE WORK

We have articulated a vision of better coordination of 'large-scale mobile sensing', driven by the observation that context is often most useful when composed from the activity/ environmental state of multiple individuals. Using a couple of simple examples, we showed that a cloud-based Mobile Sensing Service may deliver appreciable reduction in energy overheads (up to $\approx 30\%$ in our studied cases), by intelligently optimizing the query processing *across* multiple devices. Our results also shows why a *stovepiped* architecture (where each phone's query is optimized separately) may not provide the savings that are apparent at first glance.

In the near future, we plan to: (1) develop more sophisticated query optimization logic (taking into consideration more involved situations with multiple predicates) and (2) build and test out a working version of our proposal, including the use of past history to generate accurate estimates for the predicate selectivity probabilities ($P(\ldots)$).

7. REFERENCES

[1] PowerTutor: A Power Monitor for Android-Based Mobile Platforms. *http://ziyang.eecs.umich.edu/projects/powertutor/*.

[2] Balan, R. K., Gergle, D., Satyanarayanan, M., and Herbsleb, J. D. Simplifying cyber foraging for mobile devices. *ACM MobiSys*, 2007.

[3] Chu et. al, D. Balancing Energy, Latency and Accuracy for Mobile Sensor Data Classification. *ACM Sensys'11*, pages 54–67, 2011.

[4] Lu, H., Yang, J., Liu, Z., Lane, N. D., Choudhury, T., and Campbell, A. T. The jigsaw continuous sensing engine for mobile phone applications. *ACM Sensys*, 2010.

[5] McDaniel, P. On context in authorization policy. *ACM SACMAT*, 2003.

[6] Miluzzo, E., Cornelius, C., Ramaswamy, A., Choudhury, T., Liu, Z., and Campbell, A. T. Darwin phones: the evolution of sensing and inference on mobile phones. *ACM MobiSys*, 2010.

[7] Misra, A. and Lim, L. Optimizing Sensor Data Acquisition for Energy-Efficient Smartphone-based Continuous Event Processing. *IEEE MDM*, 2011.

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

[9] Wang, Y., Lin, J., Annavaram, M., Jacobson, Q., Hong, J. I., Krishnamachari, B., and Sadeh, N. M. A framework of energy efficient mobile sensing for automatic user state recognition. *ACM MobiSys*, 2009.

[10] Xiao, Y., Hui, P., and Savolainen, P. Cascap: Cloud-assisted context aware power management for mobile devices. *ACM MCS*, 2011.