Efficient Indexing Structure for Scalable Processing of Spatial Alarms

Myungcheol Doo◇ Ling Liu◇ Nitya Narasimhan♠ Venu Vasudevan♠

◇ School of Computer Science, Georgia Institute of Technology, Atlanta, GA

♠ Applied Research Center, Motorola, Schaumburg, IL

{mcdoo, lingliu}@cc.gatech.edu, {nitya, venu.vasudevan}@motorola.com

ABSTRACT

Spatial Alarms are reminders for mobile users upon their arrival of certain spatial location of interest. Spatial alarm processing requires meeting two demanding objectives: high accuracy, which ensures zero or very low alarm misses, and high scalability, which requires highly efficient and optimal processing of spatial alarms. Existing techniques for processing spatial alarms cannot solve these two problems at the same time. In this paper we present the design and implementation of a new indexing technique, Mondrian tree. The Mondrian tree indexing method partitions the entire universe of discourse into spatial alarm monitoring regions and alarm-free regions. This enables us to reduce the number of on-demand alarm-free region computations, significant saving of both server load and client to server communication cost. We evaluate the efficiency of the Mondrian tree indexing approach using a road network simulator and show that the Mondrian tree offers significant performance enhancements on spatial alarm processing at both the server side and the client side.

1. INTRODUCTION

Spatial alarms extend the idea of time-based alarms to the spatial dimension. They serve as personal reminders to the mobile users upon their arrival of a specified location of interest. An example of a spatial alarm is *Locale* [1] which enables one of the pre-defined cellphone ring settings upon arrival of future reference location of interest. For example, Alice sets a spatial alarm on her office in Georgia Tech campus. When she arrives on campus or within two miles of her office, *Locale* changes the ringer setting to the pre-defined setting such as silence.

A spatial alarm consists of four main components: a *target*, an alarm monitoring region, a spatial *event*, and an *action*. The target component specifies the future reference point of interest (Alices office in our example). The alarm monitoring region contains the alarm target and is defined by the distance threshold to the alarm target (two miles from Alices office in the campus). The spatial event component specifies the moving location of the mobile client, which is of interest to the spatial alarm (Only when Alice is close to the Georgia Tech campus, the event occurs). If the user moves to the current location and the distance to the alarm target is smaller than the spatial threshold, it means that the mobile user enters the alarm region, thus the event occurs. The action component of the spatial alarm defines the information to be disseminated to, or the action to be taken on behalf of, the mobile subscribers of the spatial alarm (e.g., the ringer setting is changed).

Categorization of Spatial Alarms.
According to the publish-subscribe scope, we classify spatial alarms into three categories: *private*, *shared* and *public*. *Private* alarms are installed and used exclusively by the publisher. *Shared* alarms are installed by the publisher with a list of authorized subscribers and the publisher is typically one of the subscribers. *Public* alarms are usually installed with the purpose of sharing them with all mobile users who are entering the spatial regions of the alarms. Mobile users may subscribe to public alarms by topic categories or keywords, such as *"traffic information on highway 85 North in Atlanta"* or *"Zagat survey of top-ranked local restaurants"*.

Challenge of Spatial Trigger Processing.
The two demanding objectives for processing spatial alarms are high accuracy, which ensures zero or very low alarm misses, and high scalability, which requires highly efficient and optimal processing of spatial alarms. In order not to miss a single spatial alarm, either the client or the server needs to check frequently whether the client enters one of her alarm monitoring regions. The alarm check frequency determines the level of accuracy obtained. The higher frequency one performs alarm check, the higher accuracy one can obtain for alarm evaluation but at higher cost of alarm processing at both server and mobile clients. At one extreme, the alarm processing server will handle the check of all alarms for all mobile clients at high frequency, which can be very expensive and not scalable. Furthermore, even in this server centric case, clients still need to be constantly connected in order for the location server to continuously track their current location through signal probing [9], which is known to be energy inefficient compared to running a simple application on the mobile client [18, 10].

Contributions and scope of the paper.
Bearing the above-mentioned problems in mind, in this paper we design and implement a new indexing structure, called Mondrian tree, for scalable processing of spatial alarms. The Mondrian tree indexing method partitions the entire universe of discourse into two types

of spatial regions: alarm monitoring regions and alarm-free regions. At any given time, a mobile client will be residing in either an alarm free region or an alarm monitoring region. This enables us to minimize the cost of dynamically computing alarm-free region for each mobile client as done in existing literature [4, 6, 11, 15]. We evaluate the efficiency of the Mondrian tree indexing approach using a road network simulator. We show that the Mondrian tree offers significant performance enhancements on spatial alarm processing at both the server side and the client side.

2. RELATED WORKS

Three areas of research are directly relevant to spatial alarm processing. First, event-based location reminder systems have been advocated in human computer interaction community [8, 17]. The main focus of this line of work is the usability of location reminder systems. None of these approaches deal with efficient processing of large number of reminders.

The second area of relevant research is the use of safe regions [6, 15] for continuous query processing or spatial alarm processing [4]. The main idea is to compute a safe region for each mobile client in terms of the spatial locality of queries, aiming at reducing the amount of unnecessary evaluations of continuous queries or spatial alarms while the user is moving inside of her safe region. All existing approaches compute safe regions dynamically for mobile clients when they move out of their current safe region or when new queries are added. However, computing safe regions on-demand has three disadvantages. First when two clients a and b are in the same location, the server computes the same safe region twice. Second, the server is burdened with increased safe region computation cost as more users cross their safe regions frequently [4, 6]. Finally, determining a safe region requires intensive computation. For n continuous queries, it takes from $O(n)$ to $O(n \log^3 n)$ to compute a rectangular safe region [15].

Another area related to this work is the information monitoring and event-based systems for delivering relevant information to users on demand. User-defined queries can be initiated when new relevant information which is of personal interest to the user is detected by the system [7].

3. MONDRIAN TREE INDEXING

In this section we present an overview of the Mondrian Tree design, including basic ideas and algorithms for construction, maintenance, and search.

3.1 Basic Ideas and Properties

The Mondrian indexing is a rectangular region partitioning method. It takes as an input the spatial alarms in the given universe of discourse and generates a region partition tree with two types of spatial regions as the output. A unique feature of the Mondrian tree is that it indexes not only spatial alarm monitoring regions but also the remaining regions, called **alarm-free regions (AFR)**, which do not have any spatial alarms. Thus, each leaf node of Mondrian tree is either an alarm monitoring region or an AFR. The goal of such a design is to allow the spatial alarm evaluation engine to easily locate the current region of a mobile client, and more importantly, to evaluate a spatial alarm only when the mobile subscriber of the alarm is inside of the alarm monitoring region.

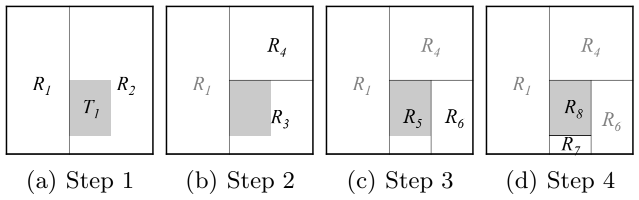


Figure 1: Step-by-step partitioning

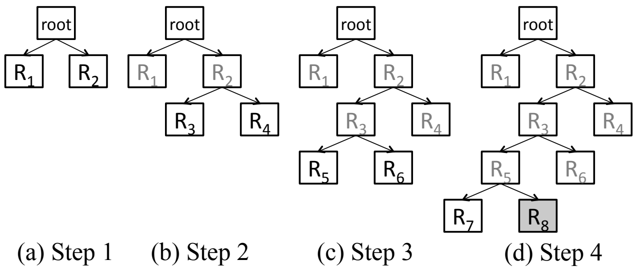
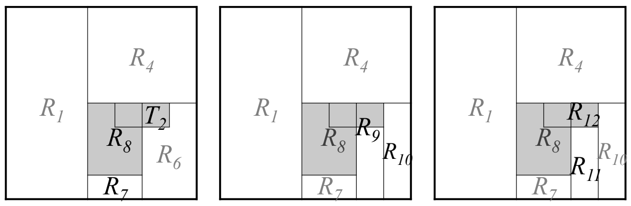


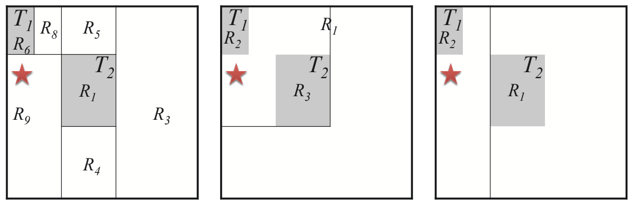
Figure 2: Step-by-step building a tree

Partitioning the Entire Region. Figure 1 shows the steps of partitioning an area with one spatial alarm into 5 disjoint regions (one alarm monitoring region and four AFR regions). The corresponding Mondrian trees for the four steps in Figure 1 are shown in Figure 2. At each step we choose a split dimension which is a perpendicular to one of the 2D coordinate axes. The split dimension may be chosen by alternating between x-axis and y-axis. In step 1, x-axis is selected as a splitting dimension. Now the entire region is divided into smaller regions, R_1 and R_2. The root of the Mondrian tree, which covers the entire region, has two children, each of which points to R_1 and R_2 respectively as shown in Figure 2(a). In step 2, R_2 in Figure 1(a) is chosen to be further partitioned because it contains T_1. We alter the split dimension from x to y-axis, which results in R_3 and R_4 as shown in Figure 1(b) and 2(b). We keep selecting a region and partitioning it until the remaining rectangle has the same region as the spatial alarm monitoring region as shown in Figure 1(d).



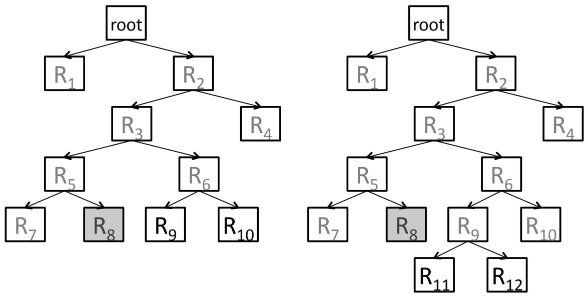
(a) T_2 is inserted (b) Step 5 (c) Step 6

Figure 3: Step-by-step partitioning



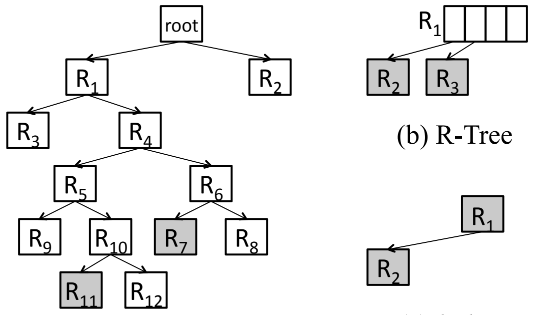
(a) Mondrian Tree (b) R-tree (c) $k-d$ tree

Figure 5: Comparison of Partitioning Scheme



(a) Step 5 (b) Step 6

Figure 4: Step-by-step building a tree



(a) Mondrian Tree (b) R-Tree (c) *k-d* Tree

Figure 6: Comparison of Building Trees

Dealing with Overlapping Spatial Alarms. Mondrian tree deals with a spatial alarm overlapping with another spatial alarm by partitioning the new alarm as two small alarms, one that overlaps with the existing alarm and the other that does not. The overlapping alarm is added into the leaf node that covers the existing alarm. The other small alarm keeps finding the right leaf node. For example, in Figure 1 there is one spatial alarm T_1. Let's add one more alarm T_2 that overlaps with T_1 as shown in Figure 3(a). We continue to partitioning from the step 4 in Figure 1(d) and Figure 2(d). When T_2 is inserted, it goes down to the leaf by selecting a node that contains T_2. When T_2 arrives at R_3 by taking the path $root \rightarrow R_2 \rightarrow R_3$, we stop at R_3 because both R_3's children are intersecting with T_2. We then split T_2 into $T_2.left$ which overlaps with R_5 and $T_2.right$ which overlaps with R_6. $T_2.left$ keeps going down until it reaches R_8. Although R_8 completely covers $T_2.left$'s region and is bigger than $T_2.left$, we do not split R_8 because it already has an alarm T_1. Instead, we add $T_2.left$ into the node R_8. Now R_8 has two spatial alarms T_1 and T_2. On the other hand, $T_2.right$ chooses R_6 for the next node. R_6 is split into R_9 and R_{10} as shown in Figure 3(b) and Figure 4(a). Then R_9 is split again into R_{11} and R_{12} as shown in Figure 3(c) and Figure 4(b).

Comparisons with R-tree and *k-d* tree. Mondrian indexing partitions the entire universe of discourse while other multi-dimensional indexing algorithms are interested in regions that have queries. Figure 5 and 6 shows how different Mondrian tree is from R-tree and *k-d* tree. The user's current location is labeled as a star. Mondrian tree returns an AFR for the the current location as shown in Figure 5(a). Therefore, the mobile client can enter a sleep mode during the safe period computed based on the shortest distance from the current location of the mobile client to the border of the AFR region. In contrast, R-tree or *k-d* tree can only locate the spatial alarms nearby a mobile client and the system needs to compute the safe region for each mobile client using the spatial alarms in the vicinity of her current location, because traditional tree-based indexing such as R-tree and *k-d* tree family only index spatial regions containing spatial alarms. Thus, one needs to search k nearest spatial alarms to a mobile client in order to compute the AFR region of the mobile client.

3.2 Challenges in Mondrian Tree Design

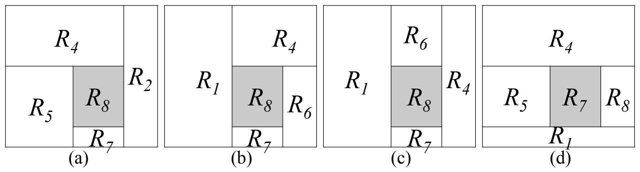
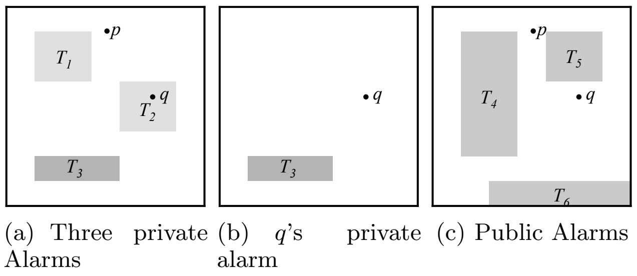


Figure 7: Various partitioning examples for one spatial alarm

We discuss two technical challenges in Mondrian Tree design. First, there are several ways of partitioning the entire universe of discourse of interest. The result of partitioning depends on the order of alarms to be inserted and the order of split dimensions used as shown in Figure 7. Therefore, an immediate question is how to build an optimal Mondrian tree in terms of fast search and low maintenance cost.

The second challenging decision is related to whether we should build one single Mondrian tree for all spatial alarms in the system or we should create one Mondrian tree for spatial alarms of each individual mobile subscriber. We below use an example to illustrate our observation and insights in terms of guidelines that we use to make our final design choice.



(a) Three private Alarms (b) q's private alarm (c) Public Alarms

Figure 8: Public and private Alarms

Consider a simplified scenario represented in Figure 8(a). There are two mobile objects p and q. p has two spatial alarms T_1 and T_2 shaded lighter. q has one alarm T_3 shaded darker.

One Mondrian tree for all spatial alarms: For Figure 8(a), the Mondrian tree for this example displays that p is inside of an alarm-free region (AFR) and q stays in the alarm T_2's monitoring region but q did not subscribe to alarm T_2. In the single Mondrain tree approach, q is either notified that she is inside of an alarm monitoring region regardless of the alarm ownership (subscription based) or the system needs to annotate each alarm region with a list of subscribers in order to detect whether the alarm region is relevant to the mobile client or irrelevant in terms of alarm subscription. In either case, maintaining a single Mondrian Tree for all alarms may incur higher overhead when there are relatively large number of private alarms or shared alarms and fewer public alarms to which every mobile subscribes. **One Mondrian tree for each mobile subscriber**: First, not all users are subscribers of an alarm. Second, not all public alarms are subscribed by all users. Private alarms are subscribed by only one mobile user and shared alarms are subscribed by a group of users. Figure 8(b) shows the entire region of interest from the user q's perspective. Compared to Figure 8(a), Figure 8(b) shows that q is not inside of any alarms and thus not an alarm subscriber. Therefore the movement of q does not trigger any alarm check at the server. It is obvious that building a separate Mondrian tree for each mobile subscriber is much more efficient in terms of both accuracy and efficiency of alarm processing, especially when the number of private alarms and shared alarms is much larger than the total number of public alarms.

On the other hand, this approach might not be efficient in terms of storage cost when a number of mobile users subscribe to the same set of spatial alarms and thus create and maintain the same Mondrian tree multiple times. Figure 8(c) shows that there are three public spatial alarms that both p and q have subscribed to. Thus two of them have the same Mondrian treecreated and maintained in the system. One optimization in this case is to maintain one tree at the server, thus saving the cost of building the same tree multiple times.

In summary, if we have $n + m$ mobile objects in the system and only n of them subscribe to some spatial alarms, then we create and maintain a maximum of n Mondrian trees. Furthermore, among n users, if we have k mobile subscribers who subscribe only public Alarms, then we need to create and maintain upto n - $k + 1$ Mondrian trees, one for public Alarms of k mobile objects and the others for private Alarms of n - k users.

4. MONDRIAN TREE ALGORITHMS

4.1 Incremental Construction

The incremental construction algorithm typically works with an existing Mondrian tree in two steps. On insertion of spatial alarm t, the algorithm frist finds recursively finds a node v such that v's region $v.R$ covers or equals to t's monitoring region $t.R$ and among all the nodes, $v.R$ is the smallest region covering $t.R$. Then the algorithm decides whether to split the region denoted by v or to assign t into node v. Concretely, to the algorithm considers the following three cases:

(a) If v is a leaf without associated with any alarms, $v.R$ completely covers $t.R$ and is bigger than $t.R$, then we split v and insert t into one of its children.

(b) If v is a leaf and $v.R$ is exactly the same as $t.R$ or v has already associated to one or more alarms, then we add t into the set of alarms maintained in the $v.alarms$.

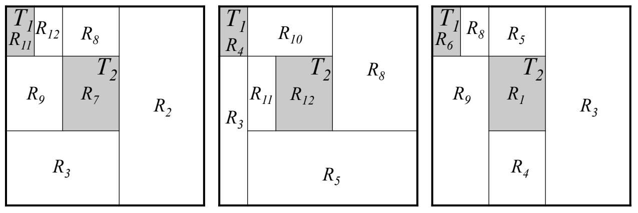
(c) If v is a non-leaf node and $t.R$ overlaps with v's children, we split the alarm t into two parts: $t.left$ and $t.right$ each of which intersects with v's children.

We provide the pseudo code for incremental insertion in Algorithm InsertAlarm below with comments (a), (b), and (c) corresponding to the three cases. SplitNode splits the node into two child nodes along the x or y-axis so that one of them contains t. Similarly, SplitAlarm handles the third case. Due to the space limit, we omit the pseudo code of SplitNode and SplitAlarm .

Algorithm 1 InsertAlarm (node v, alarm t)

```
1: if $v.isLeaf()$ then
2:     if $t.R$ equals $v.R$ or $v.alarms \neq$ null then
3:         $v.alarms \leftarrow t.ID$                         ▷ (b)
4:     else
5:         SplitNode ($v$, $t$)                              ▷ (a)
6:         if $v.left.R$ contains $t.R$ then
7:             InsertAlarm ($v.left$, $t$)
8:         else
9:             InsertAlarm ($v.right$, $t$)
10:        end if
11:    end if
12: else
13:    $(t_{left}, t_{right})$ = SplitAlarm ($v$, $t$)       ▷ (c)
14:    InsertAlarm ($v.left, t_{left}$)
15:    InsertAlarm ($v.right, t_{right}$)
16: end if
```

The incremental insertion is typically used when mobile users subscribe spatial alarms randomly and there is no prior information about spatial alarm distribution. Thus, all spatial alarms will be inserted based on first come first served. For example, Figure 9 shows how the results differ while varying the order of insertion. Figure 9 (a) shows the result of inserting T_2 and then T_1, meanwhile Figure 9 (b) shows the reverse order. Figure 10(a) and Figure 10(b) show corresponding Mondrian trees. Two trees has the same depth, but they have different shapes. We observe that the tree in Figure 10(a) tends to grow faster in left direction than the other. On the other hand, Figure 10(b) is likely to grow in right direction.



(a) T_2 and T_1 (b) T_1 and T_2 (c) Batch insertion

Figure 9: Comparison of Mondrian indexing varying the order of inserting spatial alarms

4.2 Batch Construction

The algorithm for batch construction is typically used to build the initial Mondrian Tree. It can utilize the distribution of spatial alarms to build a more balanced tree if a set of spatial alarms are installed to the system together. We also use the batch algorithm for handling incremental insertion of multiple alarms to the existing Mondrian tee.

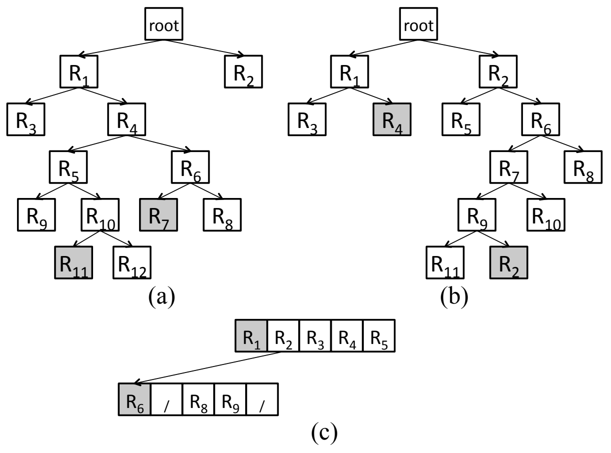


Figure 10: Comparison of Mondrian trees varying the order of inserting spatial alarms

Algorithm 2 BatchInsert (Alarm[] *tList*, Axis axis)

```
1: if tList.size() == 0 then
2:     return
3: end if
4: Sort(tList, axis)
5: int mid = tList.size() / 2
6: InsertAlarm (tList[mid])
7: Alarm[] tLeft ← tList[0..mid-1]
8: Alarm[] tRight ← tList[mid+1..tList.size()]
```

Another possible enhancement towards generating a more balanced Mondrian tree is to use a compact node data structure. In the basic Mondrian tree structure, each node has a region R, two children $left$ and $right$, and a set of alarms $alarms$ as follows:

R	$left$	$right$	$alarms$

We call this data structure as *basic node*. The idea of *compact node* is to represent each node with four child nodes pointing to all surrounding regions: $left$, $right$, *down*, and *up*. The data structure of the compact node is defined as follows:

R	$left$	$right$	$down$	up	$alarms$

Batch construction performs a parallel split. For example, in Figure 9(c), we select x axis for the starting split dimension and cut the entire region along the two vertical lines of R_1 which produce two tall left and right neighbors (R_2 and R_3) and two short lower and upper neighbors (R_3 and R_5).

In general, with n alarms we first sort the set of n alarms in one dimensional coordinate, say x, and choose an alarm such that its x coordinate value is the median of all n alarms' x coordinate values. By taking the alarm as the root, we could insert maximum one half of $n-1$ alarms in the subtree which is pointed to by root's *left* child, and the other half into the subtree pointed by root's *right* field. The same operation continues recursively after sorting remaining alarms in the other dimensional coordinate, say y. The iteration stops when there is no more alarms left to be inserted. Figure 9(c) and 10(c) show the result of partitioning by the batch construction.

4.3 Deletion

The deletion algorithm consists of two steps. The first step is to find nodes that have t using FINDREGIONS. Then for each node v found, we remove $t.ID$ from $v.alarms$. The second step handles the merge operation for different scenarios. For example, when there are no alarms left in $v.alarms$ and four surrounding regions of v are all AFRs, then the merge operation will be invoked to unite the four surrounding AFRs and the node into one bigger AFR. Due to the space limit, we omit the deletion algorithm detail and the other merge scenario analysis in this paper.

Algorithm 3 FINDREGIONS(Node v, Alarm t)

```
1:  Create an empty array $result[]$
2:  if $v.isLeaf()$ then
3:      if $v.R$ contains $t.R$ then
4:          $results[] \leftarrow v$
5:      end if
6:  else
7:      if $v.left.R$ intersects with $t.R$ then
8:          $result[] \leftarrow$ FINDREGIONS($v.left$, $t$)
9:      end if
10:     if $v.right.R$ intersects with $t.R$ then
11:         $result[] \leftarrow$ FINDREGIONS($v.right$, $t$)
12:     end if
13: end if
14: return $result[]$
```

4.4 Search Algorithm

The search for mobile user's current region in Mondrian tree is similar to the search algorithm of binary tree. From the root it compares the position of the mobile object with the left and the right children's' region to determine if the search should proceed in the left subtree or the right subtree of the index. Concretely, the search algorithm finds the *node* which contains the current position of the mobile object. Let the position be represented by $p(x, y)$. The algorithm first takes the root node as an input node v and the current position coordinate of p as $p(x, y)$. If p is inside of one of v's children c, the search continues by setting c to be the next input of FINDREGION. The iteration stops when we arrive at a leaf node of the Mondrian tree. The pseudo code of the algorithm is given in Algorithm FINDREGION. Figure 11 shows the workthrough of FINDREGION.

Algorithm 4 FINDREGION(Node v, Alarm t)

```
1:  if $v.isLeaf()$ then
2:      if $v.R$ contains $p$ then
3:          return $v$
4:      else
5:          return $null$
6:      end if
7:  end if
8:  for $c$ : all children in $v$ do do
9:      if $c.R$ intersect with $t.R$ then
10:         return FINDREGION($c$, $p$)
11:     end if
12: end for
```

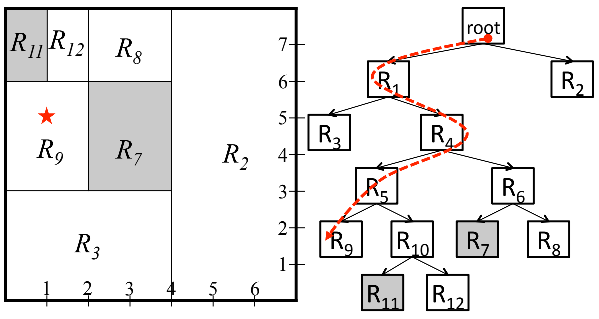


Figure 11: A search path to the current node

5. SPATIAL ALARM PROCESSING

A spatial alarm should be evaluated in three steps. First, we need to determine what type of events should activate the spatial alarm evaluation process. Second, the server needs to find out the list of alarms to be evaluated upon the occurrence of the alarm events. The more relevant this list is, the shorter this list will be and the more efficient the spatial alarm evaluation will be. Third, the server executes the action component of those spatial alarms whose alarm conditions are evaluated to be true. The decision in the first two steps should be made with the objective of minimizing alarm miss-rate to achieve high accuracy.

The nave strategy for alarm processing is periodic evaluation. High frequency is essential to ensure that few or none of the alarms are missed. Periodic evaluation, though simple, can be extremely inefficient due to frequent and often high rate of irrelevant alarm evaluations [4].

A simple and yet straightforward enhancement over periodic evaluation is to use the location update of a mobile user as the alarm evaluation event. Upon receiving a location update at the server (through either server-initiated location tracking service or client location reporting service), the server looks up spatial alarms that are in a close vicinity of the mobile's current location using the Mondrian tree (either general or client-specific). This approach is independent of the concrete location update strategies (periodic, dead-reckoning or others). However, not all location update events are suitable as the alarm firing events. First, not all location updates of a mobile client will lead to a successful evaluation of her spatial alarms. For example, Alice has five alarms installed in downtown Atlanta but her home is 30 miles north from the downtown. When Alice travels within 10 miles of vicinity of her home, all location updates of Alice should not be used as the alarm firing event, since all five alarms will never become true unless Alice travels to downtown Atlanta area. Second, location updates of a mobile client will not lead to the successful evaluation of the spatial alarms that are not subscribed or owned by this mobile. For instance, Bob's spatial alarms are indifferent to the motion behavior of Alice.

In the subsequent sections, we will describe the use of grace period, alarm free regions in conjunction with Mondrian tree for efficient processing of spatial alarms.

5.1 Grace Period

The first basic idea for evaluating spatial alarms efficiently is to incorporate the spatial locality of the alarms and the motion behavior of mobile objects. One approach is to use the concept of grace period.

Given a mobile object p and an alarm T_i ($1 \leq i \leq n$), where n is the total number of alarms installed in the system, the grace period of p with respect to T_i, denoted by $gp(p, T_i)$, can be computed based on the distance between the current position of p and the alarm monitoring region $T_i.R$, denoted as $d(p, T_i.R)$. Given a mobile client and a spatial alarm T_i, two main factors affecting the grace period are (i) the velocity of mobile client p, $f(V_p)$, and (ii) the distance $d(p, T_i.R)$. Thus, the grace period $gp(p, T_i)$ can be computed as follows:

$$gp(p, T_i) = \frac{d(p, T_i.R)}{f(V_p)} \tag{1}$$

The concept of grace period has two performance implications. At the server side, we can skip the check of any spatial alarms owned or subscribed by p during the grace period of $gp(p, T_i)$, where $\{T_i \mid T_i \in p\text{'s subscribing alarms}, 1 \leq i \leq n\}$, regardless whether or not the location update events for p may occur during this period. At the mobile client side, mobile client can enter sleep mode for the spatial alarm evaluation application during the grace period. Our experiments show that the grace period based approach can significantly reduce the amount of unnecessary alarm evaluation without loss of accuracy.

5.2 Alarm-Free Region

Algorithm 5 ComputeGP(Node v, Position p, Velocity v)

```
1: minGP ← 99999999
2: for $d$ : distance to each border of $v$ from $p$ do do
3:     gp ← d / v
4:     if $gp$ < minGP then
5:         minGP ← gp
6:     end if
7: end for
8: return minGP
```

Recall Figure 8(b) we observe that regardless of the total number spatial alarms subscribed or owned by a user, only those alarms that are in her close vicinity have non-zero probability of being fired. The idea of alarm free region (AFR) utilizes this spatial locality of alarms and the motion behavior of mobile clients. In the context of Mondrian tree, an AFR is defined as a rectangular region that does not contain any spatial alarms. As long as the mobile client moves within an AFR, the client is free from the need for alarm check, which significantly reduces the amount of unnecessary evaluations of spatial alarms at the alarm evaluation server. There are two alternative ways to compute AFR: using Mondrian tree or dynamically compute AFR based on the current location of a mobile client and the alarms nearby. We argue that the approach of dynamically computing the AFRs has three disadvantages. First, when two clients a and b are in the same location and subscribe the same alarms, then the server computes the same AFR twice. Second, the server load can grow dramatically due to increased AFR computation as users continue to move or when hot spots of user movement occur. Finally, computing an AFR itself requires intensive computation when the number of nearby spatial alarms is not small. Concretely, computing a rectangular AFR takes from $O(n)$ to $O(n \log^3 n)$, where n is the number of spatial alarms used in the AFR computation.

In summary , Mondrian tree provides an elegant solution to tackle these problems. It partitions the entire universe of discourse into two types of regions: alarm monitoring regions and AFR. Once the tree is built, the server does not need to compute AFR for any mobile clients no matter how they move and where the hotspots are. The Mondrian tree only pays for the maintenance cost when a new spatial alarm is installed (region partition) or existing alarms are expired (region merging). Furthermore, by using Mondrian batch algorithm to handle insertion of new spatial alarms and expiration of existing alarms, we can further reduce or minimize the maintenance cost of the Mondrian trees. Also when several users have the same spatial alarms, the server does not need to compute the same AFR multiple times.

5.3 Mondrian Tree with Grace Period

The goal of using Mondrian tree index with grace period is to combine AFR and grace period in a most effective manner such that the spatial alarm processing can be achieved with high accuracy and high efficiency.

Concretely, a mobile client staying in the AFR can obtain its current grace period either locally or through the server, depending on the capacity of the mobile client. This is a configuration parameter when a client registers with the spatial alarm server. When the grace period expires, the mobile client wakes up and checks if she moves out of its current AFR. If not, the client obtains a new grace period, and the spatial alarm application running on the client enters the sleep mode again during the new grace period. Otherwise, the mobile client communicates with the server to obtain its current residing region. If the region is an AFR, then the client computes the grace period using the same process. Otherwise, the mobile client is inside of an alarm monitoring region. The server invokes the alarm evaluation for this client.

The strategy of using the Mondrian tree index in conjunction with grace period prevails over the strategy of using grace period computation without Mondrian tree for a number of reasons. First, the computation for grace periods of mobile clients are done once during Mondrian tree creation and maintenance, resulting in significant saving in term of sever loads and client energy consumption. Second, Mondrian tree structure facilitates the utilization of distributed client-server architecture. For example, when a mobile client is in an AFR, the server may need to compute the grace period for this client multiple times, especially when the mobile client moves frequently but in the close vicinity. By shipping the AFR of each mobile subscriber to the client side, the mobile client can compute the grace period locally. This approach reduces the amount of communication from client to server for grace period computation within the same AFR, and thus provides significant saving on energy consumption at the client. At the same time, it also reduces the server load for spatial alarm evaluation, allowing better scalability with respect to growing number of clients and large number of spatial alarms.

5.4 Distributed Client-Server Architecture

It is widely recognized that server-centric, distributed client-server, and client-centric are three alternative architectures for mobile computing applications. Considering that client-centric architecture is only applicable for supporting private spatial alarms [12], in this paper we focus on server-centric architecture and distributed client-server architecture.

In the server-centric architecture, spatial alarms will be installed and processed at the server. Mobile clients do not contribute directly to the spatial alarm processing task. The server obtains the current location of its mobile clients through localization services [14, 9] which are orthogonal to the spatial alarm processing service.

In the distributed client-server architecture, the spatial alarm processing can be carefully partitioned into server-side processing and client-side processing. For instance, the Mondrian tree indexing approach promotes the idea of creating and maintaining one Mondrian tree for each mobile subscriber. This enables the system to minimize the overhead of searching for relevant alarms of a given mobile client and reduce the overhead of maintaining a large Mondrian tree. With this design principle in mind, we consider three possible strategies for implementing the Mondrian tree based spatial alarm evaluation. The choice of which strategy to use depends primarily on the capacity of mobile clients.

The first strategy will have the sever perform the following four tasks: (1) construct and maintain n Mondrian tree for n mobile subscribers; (2) search the Mondrian tree index to find the current region in which a mobile resides; (3) compute the grace period based on the current AFR of the mobile; and (4) send to each mobile subscriber her current grace period during which the mobile client can sleep. In this scenario, the client checks if its grace period expires and sends a new grace period request message to the server whenever its grace period expires.

The second strategy will have the server perform only the first two tasks and a modified version of the 4th task. Concretely, the server sends the current region of a mobile instead of the grace period to each client. Now the client can compute the grace period locally using its current AFR (task 3). The client only reports to the server when it moves outside of the current AFR or alarm monitoring region.

The third strategy will have each client build a Mondrian tree. Each client lookups the index locally, finds its current residing region. If it is an AFR, it computes the grace period accordingly. A client only reports to the server if it enters an alarm monitoring region.

6. EXPERIMENTAL EVALUATION

In this section we report our experimental results that evaluate the performance and effectiveness of the Mondrian tree approach for processing spatial alarms. Due to the space limit, we focus our experiments that compare the performance of Mondrian trees with two well-known spatial index structures: Grid index and R-tree. We show that Mondrian tree is more effective for spatial alarm processing, though Grid index and R-tree are known to be efficient for spatial query processing.

(1) Experiment Setup

All the experiments presented in this paper are conducted by extending the GTMobiSim mobility simulator [?, ?]. The simulator generates a set of traces of moving vehicles on a real world road network. Maps used in the simulator are obtained from the National Mapping Division of the U.S. Geological Survey[3] in the form of Spatial Data Transfer Format[2]. Simulation in this paper uses the map of Metro Atlanta, which covers an area larger than 1,000 km^2. We initially place vehicles randomly on the road segments according to traffic densities determined from the traffic volume data [5]. The trace was generated by simulating 2,000 vehicles moving on the roads of metro Atlanta for a period of ten minutes. At each intersection, a vehicle will choose a direction and a road segment to travel randomly from the set of available choices.

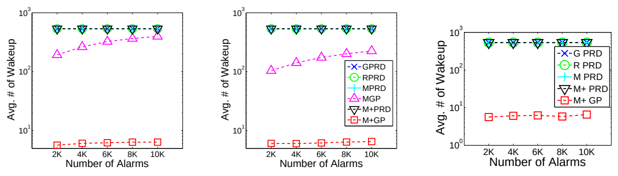
In real world, the travel patterns of mobile users exhibit hot spots. For instance, the mobile density in shopping malls or the football stadium is high during weekends or game seasons. Thus more spatial alarms may be installed around the hot spots. In order to simulate real-world environment, the spatial alarms used in our experiments are generated in two ways: uniform distribution (UNI) and skewed distribution. For skewed distribution, we use the term SKEW80 to denote that 80 percent of spatial alarms are distributed near hotspots and rest of them are uniformly distributed. Thus SKEW100 denotes the extreme case where 100 percent of spatial alarms are installed near hotspots. In our experiments, the number of spatial alarms varies from 2,000 to 10,000. The default number of alarms is 10,000.

All the experiments were conducted on an Intel Core 2 Duo CPU 2.8GHz with 4GB RAM running Windows 7. To show the advantage of Mondrian tree index, we compare the Mondrian tree with two most popular spatial indexing structures: in-memory Grid index and R-tree index.

We evaluate the performance of the Mondrian tree index under the client-server architecture with periodic evaluation approach (M_PRD), the distributed architecture with periodic evaluation (M+PRD), and the distributed architecture with grace period approach (M+GP). We compare Mondrian tree algorithms with Grid index (GPRD) and R-tree index (RPD) using periodic evaluation and dynamic AFR computation at the server.

The main factor affecting the processing of spatial alarms using Grid indexing is the size of grid cell. The smaller cells the Grid has, the more cells each alarm will intersect with. The larger cells the Grid has, the more alarms will be within a single cell, which reduces the server-side evaluation cost. To provide a fair comparison with Grid index against R-tree and Mondrian tree, we choose the size of the cell as 2,730m × 2,730m in the experiments reported in this section.

(2) Client-size Performance Evaluation



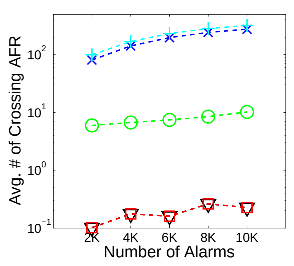
(a) UNI (b) SKEW80 (c) SKEW100

Figure 12: Number of Wake-up

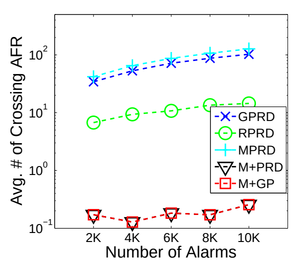
Figure 6 shows the average number of wake-up for each mobile client. In all approaches except M+GP, clients have the same number of wake-up because clients periodically wake up wait for server to inform them if they are in an AFR or an alarm monitoring region. These results show that periodic evaluation has two orders of magnitude larger number of wakeups compared to Mondrian grace period approach (M+GP) because clients in M+GP wake up only when their grace period expires.

Now we examine the size of AFR and its relationship to the number of AFR crossings. Figure 13(a) and 13(b) show the average number of AFR crossings when varying the total number of alarms. The average size of AFR under the varying number of alarms is measured and shown in Figure 13(c) and 13(d). We observe that as the number of alarms is increasing, the average size of AFR decreases for both uniform and skewed alarm distributions. The average size of AFR in M+PRD and M+GP is up to five orders of magnitude bigger than that of MPRD, GPRD, and RPRG approaches. In M+PRD and M+GP, each client only creates and maintains its own Mondrian tree, which has only the number of alarms that the client subscribes to, and thus very small compared to the centralized server-side Mondrian tree (MPRD). Therefore the average AFR size is much bigger for distributed Mondrian tree approach. In MPRD the average size of AFR varies depending on the type of alarm distribution. In the uniform distribution, the average AFR size is smaller than that of GPRD or RPRD because rectangles are distributed evenly and Mondrian tree partitions the entire region into smaller rectangles. In skewed distribution, the higher skewedness of the alarms is, the larger average AFR size MPRD will have and the more alarms are placed near hot spots. show the reverse trend of Figure 13(c) and 13(d).

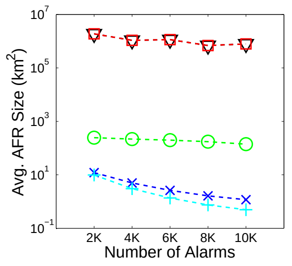
In summary, when using periodic evaluation approaches, clients wake up periodically and wait for the server to inform them if they enter an alarm monitoring region or send them the new AFR if they cross the existing AFR. In contrast, with grace period approach the spatial alarm application on the client side enters the



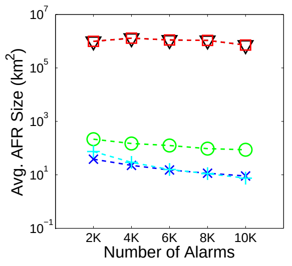
(a) Crossing AFRs(UNI)



(b) Crossing AFRs(SKEW80)

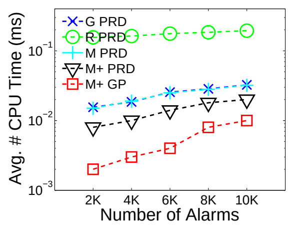


(c) AFR Size(UNI)

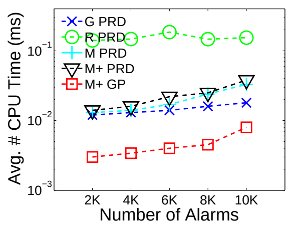


(d) AFR Size(SKEW80)

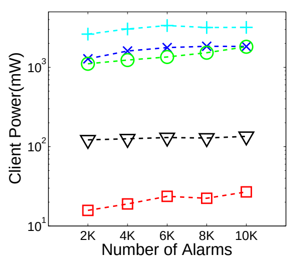
Figure 13: Client Evaluation: AFR crossing vs AFR size



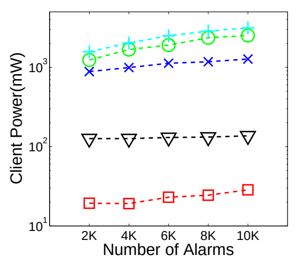
(a) Client CPU(UNI)



(b) Client CPU(SKEW80)



(c) Client Power(UNI)



(d) Client Power(SKEW80)

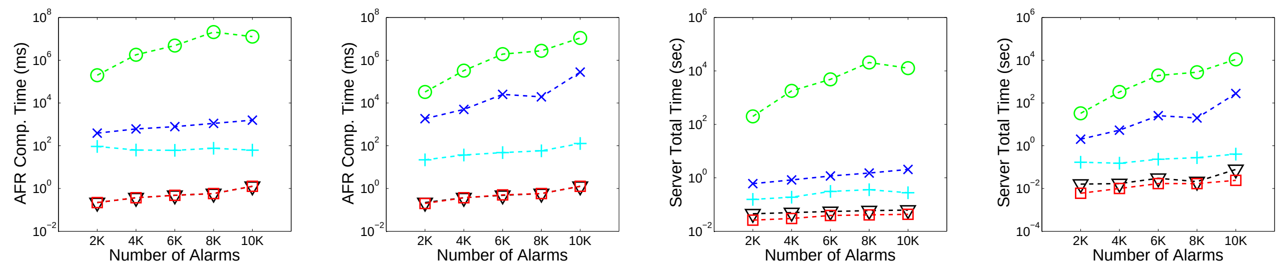
Figure 14: Client Evaluation: CPU (ms) and Power (mW)

sleep-mode longer than periodic approaches. Therefore M+GP has the shortest CPU time as shown in Figure 14(a) and 14(b). RPRD has the longest CPU time because the server takes the longest time for computing AFR as shown in Figure 15(a) and 15(b).

The third set of experiments on the client size aims at measuring CPU and energy efficiency. We compute the power consumption of client device for CPU usage and network usage. According to [16] even in idle mode in which there is no data transfer via a wireless network device, the network device consumes power, and the activity for data transfer via network devices consumes more power than the computing activity. Due to the server processing time as shown in Figure 15(a) and 15(b), clients in GPRD and RPRD wait longer and more frequently for AFR information from the server and thus consume more battery power. This is evident from Figure 14(c) and 14(d). Clients in M+GP consumes the least power and wakeups the least. M+PRD approach, relatively speaking, consumes less power than Grid, R-tree, and server-side (centralized) Mondrian tree approaches. However, the M+PRD approach consumes more power than M+GP because M+GP requires clients to wake up only when their grace period is expired instead of periodically as in M+PRD. RPRD and MPRD approaches consume most power. In MPRD, clients cross their AFR frequently due to the small size of AFRs. In RPRD, clients have to wait longer due to the longer response time compared to the G_PRD approach.

(3) Performance Evaluation in Server-side
Mondrian tree approach indexes both spatial alarms and AFRs at the same time. Therefore, it does not require on-demand AFR computation as Grid index (GPRD) and R-tree index (RPRD) approaches. Figure 15(a) and 15(b) show AFR computation for GPRD and RPRD approaches. RPRD takes more time than GPRD due to the higher search cost for finding the nearby alarms.

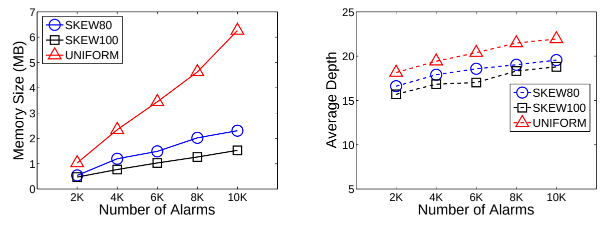
Figure 15(c) and 15(d) show the total server time, which consists of computing AFRs (in R-tree and Grid index) or searching AFRs (in Mondrian tree) and processing spatial alarms. The main factor affecting the server time is the number of crossings either reported or detected depending on whether the location of clients is tracked and whether the distributed architecture is employed. The more AFR crossings a client experiences, the more time the server spends in computing or searching AFR in addition to processes alarms. Given that Mondrian distributed approach (M+GP) has the largest AFR size on average, there is fewer AFR crossings experienced at the client side. Therefore, M+GP has the least server time for spatial alarm processing, while RPRD consumes the largest server time.



(a) AFR Comp.(UNI) (b) AFR Comp.(SKEW80) (c) Server Time(UNI) (d) Server Time(SKEW80)

Figure 15: Server Side Alarm Evaluation

(4) Cost of Mondrian Tree in Size and Depth In this set of experiments, we vary the number of spatial



(a) Memory Size (b) Average Depth

Figure 16: Cost of Mondrian Tree

alarms from 2,000 to 10,000 and evaluate the cost of constructing Mondrian tree in terms of memory size and the average tree depth.

The fact that a Mondrian tree with more alarms requires more storage is self-evident as shown in Figure 16(a). A leaf node that already has a set of alarms will add a new alarm instead of splitting the node. Thus when alarms are distributed with higher skewedness, a Mondrian tree has less number of nodes than the one with uniform distribution. Thus, UNI requires more storage (Figure 16(a)) and has higher tree depth (Figure 16(b)).

7. CONCLUSIONS

We have described the design and development of the Mondrian tree index for efficient processing of spatial alarms. The main distinguishing feature of Mondrian tree compared with conventional spatial indexes such as R-tree family, Grid file family is that Mondrian tee approach indexes not only spatial alarms but also AFRs, enabling fast lookup of AFRs instead of on-demand computation of AFRs. Another novelty of the Mondrian indexing framework is its ability to utilize the characteristics of spatial alarms to create and maintain one Mondrian tree for each mobile subscriber, which is particularly effective when there is relatively small number of public alarms compared to the private and shared alarms in the system. Our experiments show that the distributed grace-period based Mondrian tree approach (M+GP) can dramatically minimize the amount of unnecessary spatial alarm processing compared to sever side (centralized) Mondrian tree or R-tree and Grid indexing structures under periodic evaluation with on-demand AFR computation.

8. REFERENCES

[1] Locale. http://www.twofortyfouram.com/.
[2] Spatial data transfer format. `http://www.mcmcweb.er.usgs.gov/sdts/`.
[3] Us geological survey. `http://www.usgs.gov/`.
[4] B. Bamba, L. Liu, , A. Iyengar, and P. S. Yu. Distributed processing of spatial alarms: A safe region-based approach. ICDCS, 2009.
[5] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *Proc. IEEE ICDCS*, 2005.
[6] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. SIGMOD, 2005.
[7] L. Liu, C. Pu, and W. Tang. Webcq - detecting and delivering information changes on the web. In *Proc. CIKM*, 2000.
[8] P. Ludford, D. Frankowski, K. Reily, K. Wilms, and L. Terveen. Because i carry my cell phone anyway: Functional location-based reminder applications. In *SIGCHI*, 2006.
[9] E. Martin, L. Liu, M. Weber, P. Pesti, and M. Woodward. Unified analytical models for location management costs and optimum design of location areas. In *Proc. CollaborateCom*, 2009.
[10] R. N. Mayo and P. Ranganathan. Energy consumption in mobile devices: Why future systems need requierement-aware energy scale-down. In *Proc. Power-Aware Computing Systems*, 2003.
[11] M. Mokbel, X. Xiong, and W. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. ACM SIGMOD*, 2004.
[12] A. Murugappan and L. Liu. Energy-efficient processing of spatial alarms on mobile clients. In *Proc. ICSDE*, 2008.
[13] P. Pesti, B. Bamba, M. Doo, L. Liu, B. Palanisamy, and M. Weber. Gtmobisim: A mobile trace generator for road networks. In *Technical Report, College of Computing, Georgia Tech*, 2009.
[14] P. Pesti, L. Liu, B. Bamba, A. Iyengar, and M. Weber. Roadtrack: Scaling location updates for mobiles on road networks with query awareness. In *VLDB*, 2010.
[15] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. In *IEEE Transactions on Computers*, 2002.
[16] V. Raghunathan, T. Pering, R. Want, A. Nguyen, and P. Jensen. Experience with a low power wireless mobile computing platform. In *Proc. of International Symposium on Low power Electronics and Design*, 2004.
[17] T. Sohn, K. Li, G. Lee, I. Smith, J. Scott, and W. Griswold. Place-its: A study of location-based reminders on mobile phones. In *UbiComp*, 2005.
[18] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-hel devices. In *IEICE Trans. on Communications*, 1997.