Safe Region Techniques for Fast Spatial Alarm Evaluation

Bhuvan Bamba, Ling Liu
College of Computing
Georgia Institute of Technology
{bhuvan, lingliu}@cc.gatech.edu

Arun Iyengar
IBM T.J. Watson Research Center
aruni@us.ibm.com

Philip S. Yu
Department of Computer Science
University of Illinois at Chicago
psyu@cs.uic.edu

***Abstract*— Spatial alarms are personalized location-based triggers installed by mobile users to serve as a reminder of a location of interest to be encountered in their future trips. Unlike continuous spatial queries, spatial alarms do not require immediate processing and periodic reevaluation upon installation. Thus, a critical challenge for efficient processing of spatial alarms is to determine when to evaluate each spatial alarm, while ensuring the demanding requirements of *high accuracy* and *system scalability*. In this paper, we compare alternative approaches for evaluation of spatial alarms: *periodic evaluation*, *safe period-based processing* and *safe region-based processing*. We argue that the safe region-based approach provides highly efficient processing of spatial alarms at the server. Furthermore, it reduces wireless communication costs and energy consumption on the client side by reducing the number of location updates to be transmitted to the server without sacrificing accuracy of spatial alarm evaluation. We develop safe region computation techniques based on different heuristics, namely, *Maximum Perimeter Rectangular Safe Region (MPSR)*, *Largest Component Rectangles Safe Region (LCSR)* and *Bitmap Encoded Safe Region (BSR)* approach, and present an *in-depth* study on trade-offs involved in the selection of an appropriate safe region computation strategy. Our experimental evaluation shows that the best optimization strategy requires an approach which adapts to changing system load conditions and resource constraints, as none of the safe region computation techniques outperforms the others on all relevant evaluation metrics. Experimental evaluation also validates our conjecture that safe region-based processing offers *close to optimal* performance in terms of CPU load on the server and wireless communication costs at the mobile clients.**

I. Introduction

With the advent of mobile communication technology and continued price reduction of location tracking devices, location-based services (LBSs) are widely recognized as an important capability of the future computing environment [6]. Spatial alarms are one of the fundamental functionality for many LBSs. In this paper we present safe region-based optimization techniques for efficient processing of spatial alarms in a client-server based architecture.

Spatial alarms extend the idea of time-based alarms to future events that do not have a definite time of occurrence associated with them but are sensitive to spatial locations which mobile users may travel to in the future. Just as time-based alarms are set to remind us of the arrival of a *future reference time point*, spatial alarms are set to remind us of the arrival of a *spatial location of interest*. However, unlike time-based alarms, the future time instance associated with the occurrence of this event is not definite. Thus, spatial alarms can be modeled as location-based triggers which are fired whenever a mobile user enters the spatial region of the alarms. Spatial alarms provide critical capabilities for many location-based applications ranging from real time personal assistants, inventory tracking, to safety warning systems.

Spatial alarm processing requires meeting two demanding objectives: *high accuracy*, which ensures no alarms are missed, and *system scalability*, which guarantees that the alarm processing system scales to large number of spatial alarms and growing base of mobile users. A simple approach to similar problems involves periodic evaluations at a high frequency. Each spatial alarm evaluation can be conducted by testing whether the user is entering the spatial region of any of her relevant alarms. High frequency is essential to ensure that none of the alarms are missed. Though periodic evaluation is simple, it can be extremely inefficient due to frequent alarm evaluation and the high rate of irrelevant evaluations. This is especially true when the mobile user is traveling in a location that is distant from all her location triggers, or when all her alarms are set on spatial regions that are far apart from one another.

Spatial alarms can be processed using server-based infrastructure, client-based architecture or a cooperative architecture where the server and client share the responsibility of alarm processing. A server-based approach must allow optimizations for processing spatial alarms installed by multiple mobile clients, whereas a client-based approach focuses more on energy-efficient solutions for evaluating a set of spatial alarms installed on a single client. We discuss server-centric approaches for scalable processing of spatial alarms, aimed at optimizing the conventional approach of periodic alarm processing. We show that a cooperative approach where the server computes a safe region for the client and the client monitors its position within this safe region outperforms other server-based processing techniques.

In this paper, we propose the idea of safe region-based processing of spatial alarms and discuss different techniques for safe region computation at the server. Concretely, we propose the *Maximum Perimeter Rectangular Safe Region (MPSR)*, *Largest Component Rectangles Safe Region (LCSR)* and *Bitmap Encoded Safe Region (BSR)* approach. We compare the performance of these techniques with periodic alarm processing, safe period-based processing and display that all safe region-based approaches outperform other processing techniques. On the server side, safe region-based approaches provide scalability by reducing the computational load on the server. This is a direct result of the reduction of number of location updates required by the server to process alarms with high accuracy. On the client side, this reduction in number of transmitted location updates results in significant savings in terms of energy and wireless communication costs. These improvements are obtained at the cost of simple computation by the client to monitor its position within the safe region determined by the server. We provide a detailed study which considers the trade-offs involved in selection of an appropriate safe region computation approach. Our experimental evaluation shows that the best strategy requires a flexible and adaptive approach that can dynamically take into account changing system load conditions and resource constraints in its optimization decision, as none of the developed techniques outperform all other safe region computation approaches for all relevant evaluation metrics. We also show that the performance obtained using safe region-based processing is close to an *optimal solution* in an unconstrained environment where the client has complete knowledge of all spatial alarms installed in the vicinity of its current position. To the best of our knowledge, this is the first work that not only demonstrates the advantage of safe region-based optimization over periodic and safe period-based approaches but also provides a comprehensive study of various alternative safe region-based techniques for efficient processing of spatial alarms.

The rest of the paper is outlined as follows. Section II provides a brief overview of the different alarm processing algorithms. The safe region-based alarm processing techniques are introduced in Section III. We present the algorithms for safe region computation for the MPSR approach, LCSR approach and the BSR approach. Our experimental evaluation is presented in Section IV. The related work is presented in Section V and we conclude the paper in Section VI.

II. Spatial Alarm Processing Algorithms

A spatial alarm expresses a location-based information need of a subscriber $s \in S$ around a *location of interest*. The alarm trigger requires that the subscriber be informed as soon as she enters a spatial region R around the location of interest and the non-spatial constraints associated with the trigger are satisfied. Spatial alarms can be categorized as *private*, *shared* or *public* alarms depending on the scope of subscribership of the alarm. Private alarms are relevant to a single subscriber authorized to install or remove the alarm. A subscriber may install an alarm on the neighborhood grocery store reminding her to purchase groceries when she is within a one mile radius of the store and special discounts are available on her desired items. Shared alarms are installed by a subscriber of the alarm and may be shared with a group of users; for example, in the above scenario the subscriber may wish to share the alarm on the grocery store with other members of her household. Public alarms are relevant to all subscribers in the system; examples of such alarms are warning notifications against hazardous road conditions.

Algorithm 1: Safe Period-based Processing

```
1  forall ($\vec{p_s}(t), s \in S$) do
2      if ($t < sp(s)$) then
3          drop($\vec{p_s(t)}$); //client does not transmit update
4      end
5      else
6          $process(\vec{p_s}(t))$;
7          forall ($a_j \in A_s, j \in [1...|A_s|]$) do
8              $ap(s, aj) = calcApproachPeriod(s, a_j)$;
9          end
10         $sp(s) = \min_{1 \leq j \leq |A_s|}(t + ap(s, a_j))$;
11     end
12 end
```

Different approaches may be deployed in order to process spatial alarms. A *periodic evaluation* approach processes alarms as and when location updates are received from subscribers. We do not consider the update strategy adopted by the system for periodic processing; for example, subscribers may be required to provide an update to the system every five seconds or they may update their location after travelling every 100m using dead reckoning [21]. Irrespective of the update strategy adopted, this approach suffers from two problems. Firstly, the alarm miss rate is unpredictable as it is impossible for the system to determine the ideal location update period. Even with a very high frequency of updates, the system may not be able to achieve 100% success rate. Secondly, this approach processes a large number of unnecessary updates which causes system scalability to suffer.

In order to deal with the deficiencies associated with the periodic processing approach, we introduce a safe period-based approach. A safe period is computed for each subscriber such that no relevant alarms can be triggered for a subscriber before the expiry of its safe period. The algorithm for safe period-based alarm processing is outlined in Algorithm 1 below. Each location update $\vec{p_s(t)}$ for subscriber s is processed as follows. The timestamp associated with the update t is checked to determine if the safe period $sp(s)$ for the subscriber has expired. All updates received by the system before the expiry of the subscriber's safe period are dropped (lines 2-4). Preferably, the server communicates the safe period to each subscriber which does not report any location updates to the server before the expiry of its safe period in order to conserve energy and bandwidth. As soon as an update from subscriber s with timestamp $t >= sp(s)$ is received, the system processes the location update against the set of stored spatial alarms to determine if any relevant alarms need to be triggered (line 6). The system also computes an *approach period* $ap(s, a_j)$ for each relevant alarm $a_j \in A_s$, where A_s is the set of alarms relevant to the subscriber s (lines 7-9). The approach period is based on a distance measure between the subscriber position and the spatial alarm region $R(a_j)$ and an estimation of the subscriber motion over this time period. The safe period $sp(s)$ for subscriber s is calculated as the minimum time required by the subscriber to approach any of its relevant alarms (line 10). The cost of calculating the safe period for each subscriber is an additional cost associated with safe period-based processing. However, the savings on alarm processing cost far outweigh the additional cost of safe period computation as shown by our experimental evaluation. Further, in order to control this cost the client may calculate the approach period only for relevant alarms in the vicinity of its current position; for example the server may limit the calculation of the approach period for alarms within a one mile radius of the subscriber position. A disadvantage associated with this approach is that it demands pessimistic estimations related to the motion of the subscriber in order to guarantee 100% success rate for alarm triggers. Pessimistic estimations again lead to unnecessary updates being transmitted by the clients and processed at the server. More optimistic motion estimations lead to alarm misses which is unacceptable for our alarm processing system. For more details on safe period-based processing we refer interested readers to our technical report [3].

As discussed above, even the safe period-based approach relies on pessimistic assumptions related to the motion of the subscriber which makes it inefficient. In Section III, we further introduce safe region-based approaches which compute a safe region for each subscriber. As long as the subscriber remains inside its safe region ξ_s, the probability of any relevant alarms being triggered is zero. The server computes a safe region for each subscriber and communicates this safe region to the subscriber. The subscriber is responsible for monitoring its position within the safe region. Once the subscriber moves out of its safe region it provides a location update to the server which performs alarm processing and recomputes the safe region.

III. Safe Region

In this section, we first discuss the basic concepts associated with our safe region computation techniques. We introduce a grid-based framework for limiting the extent of the safe region which controls the computation costs. Next, we present the MPSR algorithm for computing rectangular safe regions around the client location. This approach limits the safe region shape to a rectangular region, thus computing smaller safe regions. However, the downstream bandwidth costs of broadcasting the safe region to the clients and client computational costs for safe region containment detection are low. A simple modification, termed as the LCSR approach, enables us to calculate more complex shaped larger safe regions for the client. This further reduces the wireless communication costs for the client. However, the downstream bandwidth and client computational costs increase due to more complex shaped safe regions. We further introduce BSR techniques for safe region computation which can express larger safe regions using a simple bitmap. This technique is shown to save on communication costs for broadcasting from the server to the clients for low alarm density regions. A simple *Grid Bitmap Encoded Safe Region (GBSR)* approach fails to compute safe regions efficiently and accurately. An extension to this approach using a *pyramid* [18] data structure, termed as *Pyramid Bitmap Encoded Safe Region (PBSR)* approach, allows for more efficient and accurate safe region computation. Last but not the least, this approach provides flexibility by allowing clients to adjust the granularity of their safe region expanse depending on their computing capability.

A. Safe Region Representation

The safe region ξ_s for any subscriber s may be defined as the region within which the probability of any relevant alarms being triggered is zero. In its simplest form safe region for any subscriber comprises of the region covered by the entire Universe of Discourse (or map) $\mathbb{U}$ except the relevant alarm regions. However, such a definition for safe region would amount to communicating information for all relevant alarms to the subscriber which proves to be prohibitively expensive. We now introduce a grid-based framework which allows us to limit the defined safe region to the vicinity of the current subscriber position.

Definition 1: In our framework, we map the Universe of Discourse $\mathbb{U} = Rect(x, y, w, h)$ onto a grid $\mathbb{G}$ of cells. $\{x, y\}$ represents the bottom-left corner and w, h represent the width and height of $\mathbb{U}$. Formally, a grid corresponding to the universe of discourse $\mathbb{U}$ can be defined as $\mathbb{G}(\alpha, \beta) = \{C_{i,j} : 1 \leq i \leq M, 1 \leq j \leq N, C_{i,j} = Rect(x + i \cdot \alpha, y + j \cdot \beta, \alpha, \beta), M = \lceil w/\alpha \rceil, N = \lceil h/\beta \rceil\}$. $C_{i,j}$ is an $\alpha \times \beta$ rectangular area representing the grid cell that is located in the i^{th} column and j^{th} row of the grid $\mathbb{G}$.

Considering our above definition of a grid we can define a mapping from any point $\vec{p} = (p_x, p_y)$ in the Universe of Discourse to the Grid, $f : \mathbb{U} \Rightarrow \mathbb{G}$.

Definition 2: Let $\vec{p} = (p_x, p_y)$ be any point in the Universe of Discourse $\mathbb{U}$. Let $C_{i,j}$ denote a cell in the grid $\mathbb{G}(\alpha, \beta)$. $f(\vec{p})$ is a position to grid cell mapping, defined as $f(\vec{p}) = C_{\lceil \frac{p_x - x}{\alpha} \rceil, \lceil \frac{p_y - y}{\beta} \rceil}$ where (x, y) denotes the bottom-left corner of $\mathbb{U}$.

Our safe region approaches utilize this grid-based framework to efficiently calculate the safe region for each subscriber. The grid-based framework can be used to limit the safe region computation to an area comprising of the current cell of a subscriber s. We define the *monitoring region* ψ_s inside the current grid cell below and proceed to describe our algorithms for safe region computation.

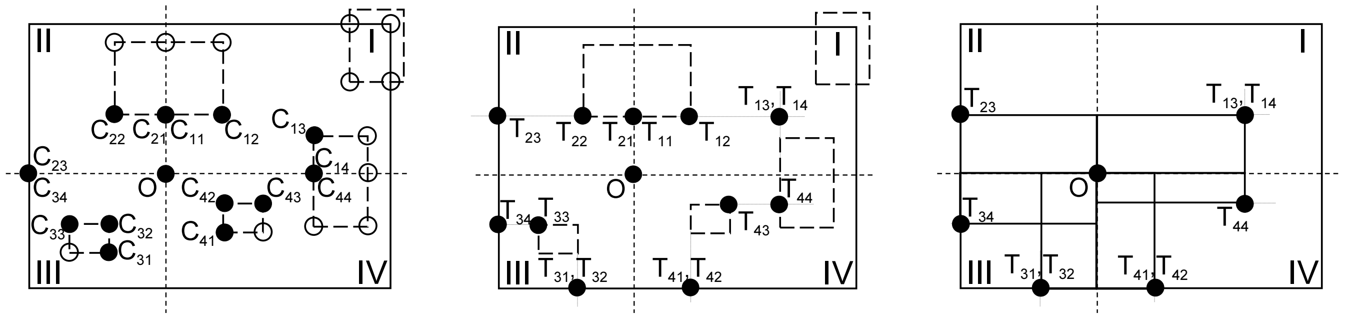
Definition 3: Monitoring Region ψ_s for any subscriber s located in cell $C_{k,l}$ may be calculated as,

$$\psi_s = C_{k,l} - \bigcup_{i=1}^{|A_s^{rel}|} R(s, A_i), \tag{1}$$

where $R(s, A_i)$ defines the spatial alarm regions relevant to subscriber s intersecting the current subscriber cell $C_{k,l}$.

B. Maximum Perimeter Rectangular Safe Region Computation

In this section, we devise an approach to compute a rectangular safe region for a subscriber. The goal of the algorithm described here is to compute a rectangular safe region with



II
I
III
IV
O
C_{22} C_{21} C_{11} C_{12} C_{13} C_{14} C_{23} C_{34} C_{42} C_{43} C_{44} C_{33} C_{32} C_{41} C_{31}
T_{23} T_{22} T_{21} T_{11} T_{12} T_{13}, T_{14} T_{34} T_{33} T_{44} T_{43} T_{31}, T_{32} T_{41}, T_{42}

(a) Candidate Point Sets (b) Tension Point Sets (c) Component Rectangles

Fig. 1: Maximum Perimeter Rectangular Safe Region Computation

maximum perimeter; for a convex shaped safe region the *amortized cost* of location updates over time is minimized if perimeter is maximized [9]. A rectangular shape also allows clients to conveniently detect their location with respect to the safe region.

We present our algorithm for computing the maximum perimeter rectangular safe region in algorithm 2 below. The algorithm accepts the current position vector $\vec{p_s}$ for subscriber s and the current grid cell $C(\vec{p_s})$ in which the subscriber resides as inputs. The set of relevant alarms A_s^{rel} for safe region computation is calculated as the set of installed alarms A_s for subscriber s intersecting the grid cell $C(\vec{p_s})$ (line 1). In case no alarms intersecting the grid cell are present, the current subscriber cell is returned as the safe region (lines 2-4). Otherwise, the algorithm partitions the cell $C(\vec{p_s})$ into four quadrants with current subscriber position $\{p_x, p_y\}$ as the origin. We define a set of *candidate points* ($cPtQuads$) and a set of *tension points* ($tPtQuads$) for each quadrant (line 6). The candidate points form the set of points which can potentially form a corner point of the rectangular safe region. Tension points are obtained from candidate points by ensuring that only points that form a rectangular region not overlapping any alarm regions are selected.

Algorithm 2: Maximum Perimeter Rectangular Safe Region Computation

```
   Input: $\vec{p_s}, C(\vec{p_s})$
   Output: $\xi_s$
1  $A_s^{rel} = C(\vec{p_s}) \cap A_s$;
2  if ($A_s^{rel} == \phi$) then
3      return $C(\vec{p_s})$;
4  end
5  else
6      $initialize(cPtQuads, tPtQuads)$;
7      $cPtQuads = getQuadrants(A_s^{rel})$;
8      $cPtQuads = trimCandidatePoints(cPtQuads)$;
9      $tPtQuads = getTensionPoints(cPtQuads)$;
10     $\xi_s = getSRfromCR()$;
11 end
```

The set of candidate points is determined as follows (line 7). Firstly, each alarm corner is assigned as a candidate point in its appropriate quadrant. For alarms which do not completely lie inside the cell $C(\vec{p_s})$ the intersection points of the cell and the alarm are also considered as candidate points. Secondly, for alarms which intersect the x-axis or y-axis of the coordinate system with origin at $\{p_x, p_y\}$, we also consider points of intersection of the alarms with the axes as candidate points. The algorithm trims the set of candidate points in the next step (line 8). Firstly, in case multiple candidate points in a quadrant intersect the x-axis (or y-axis), all candidate points other than the point on the x-axis (or y-axis) closest to the origin are removed from *cPtQuads*. If no intersection points are present on the x-axis, the point of intersection of the x-axis and the cell is added to the candidate points set. Further, we remove points which *dominate* any other point from the candidate set. A point p_1 is said to dominate point p_2 if $p_1.x > p_2.x$ and $p_1.y > p_2.y$. Finally, the points are sorted according to increasing distance of the x-coordinate from the origin. Points with the same x-coordinate are arranged in order of decreasing distance of y-coordinate from origin.

The set of candidate points is then processed in the following manner to obtain the set of tension points (line 9). Each tension point T_{Qi}, where $Q \in \{1, 2, 3, 4\}$ represents the quadrant the point belongs to, has the same x-coordinate as the corresponding candidate point C_{Qi}. The y-coordinate of T_{Qi} is the same as that of C_{Qi-1}, or T_{Qi-1} if T_{Qi} and T_{Qi-1} have the same x-coordinate. The y-coordinate of T_{Q1} is set as either the top bound of the cell or the y-coordinate of a candidate point intersecting the y-axis if any. The set of tension points form the opposite corner (opposite to the origin) of the set of candidate *component rectangles* in each quadrant. The final safe region is composed of the intersection of the component rectangles from each quadrant (line 10). The MPSR approach and the LCSR approach described in the next section differ on the heuristic used to determine the composition of the safe region from the component rectangles. We describe the composition of the safe region for the MPSR approach here. The algorithm adopts a greedy heuristic in which the quadrant with a component rectangle with the largest perimeter is selected first. Quadrants are further selected in a clockwise order, at each step the component rectangle which forms a safe region with the largest perimeter is selected. The algorithm continues until all four quadrants are processed. As opposed to a solution which enumerates every possible combination of component rectangles thus taking quartic time, this approach performs only four greedy decisions.

Figure 1 shows an example of the MPSR computation approach. The candidate point sets for the given scenario are as shown in Figure 1(a). The darkened points represent the candidate points whereas the hollow dots represent points which are trimmed from the candidate point set as explained above. Figure 1(b) displays the set of tension points obtained from the candidate point set as explained in the above algorithm. If one imagines an elastic band laid around the candidate points, the tension points can be obtained by stretching this elastic band to obtain a *rectilinear polygonal* shape which does not overlap any of the alarm regions. Figure 1(c) displays the component rectangles formed by selecting a few of the tension points. The component rectangle in Quadrant I forms the component rectangle with the largest perimeter; thus, Quadrant I is selected as the initial quadrant. The algorithm proceeds to select the next quadrant in a clockwise manner; Quadrant IV is selected. Addition of the component rectangle at tension point T_{44} provides a safe region with larger perimeter compared to the safe region obtained by adding component rectangle with tension point at T_{41}, T_{42}. Finally, the component rectangles with tension points at T_{34} in Quadrant III and T_{23} in Quadrant II are selected. The final safe region composed out of the component rectangles is as shown in Figure 2.

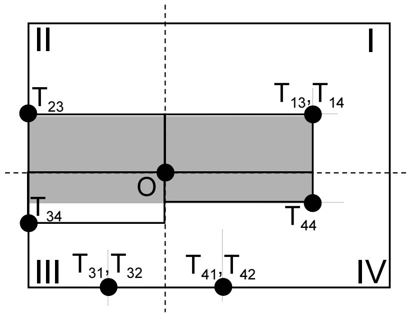


Fig. 2: MPSR from Component Rectangles

The above approach for safe region computation has two advantages: (i) the number of updates sent to the server are drastically reduced compared to a periodic processing approach, and (ii) the subscriber can easily determine her position with respect to the safe region by performing a single computation. However, the procedure for calculating the MPSR for each subscriber proves expensive. Secondly, we show that it is possible for a subscriber to easily monitor her position within a larger safe region if the rectangular shape constraint is removed. The LCSR approach described in the next section provides a complex shaped safe region which further reduces the wireless communication costs for subscribers by requiring even fewer location updates be sent to the server.

C. Largest Component Rectangles Safe Region Computation

The LCSR approach requires determination of candidate points and tension points in a similar manner as described above for the MPSR approach. It uses a different heuristic from the MPSR approach for determination of the safe region from the component rectangles. In stead of using a complicated heuristic to determine a simple rectangular safe region, this approach uses a much simpler heuristic to determine a complex but larger safe region. This results in reduction of safe region computation load on the server. The LCSR approach scans each quadrant to determine the largest component rectangle in each quadrant. The safe region comprises of the union of the largest component rectangles in each quadrant. The region may be represented by a set of five points only, comprising of the origin $\{p_x, p_y\}$ and the four opposite corners of the largest component rectangle in each quadrant. This results in a 150% increase in the downstream bandwidth consumption for communicating a safe region to the client when compared to the MPSR approach. Further, it may require up to four times the computational costs at the client to determine the position of the client within the safe region. It is possible to select even more component rectangles in each quadrant to determine larger safe regions; however, the component rectangles in each quadrant would overlap in such a scenario providing minimal increase in safe region area at the cost of additional load on the downstream bandwidth and the client. Note that the selection of only the largest component rectangle in each quadrant results in non-overlapping regions being selected in each quadrant which significantly increases the area under the safe region with minimal increase in complexity of safe region representation. Figure 3 displays the safe region composed by selecting the largest component rectangle in each quadrant represented by the set of points $\xi_s = \{O, T_{13}, T_{23}, T_{34}, T_{41}\}$.

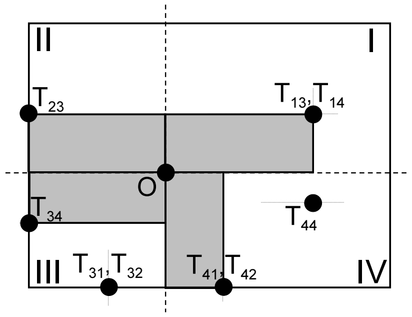
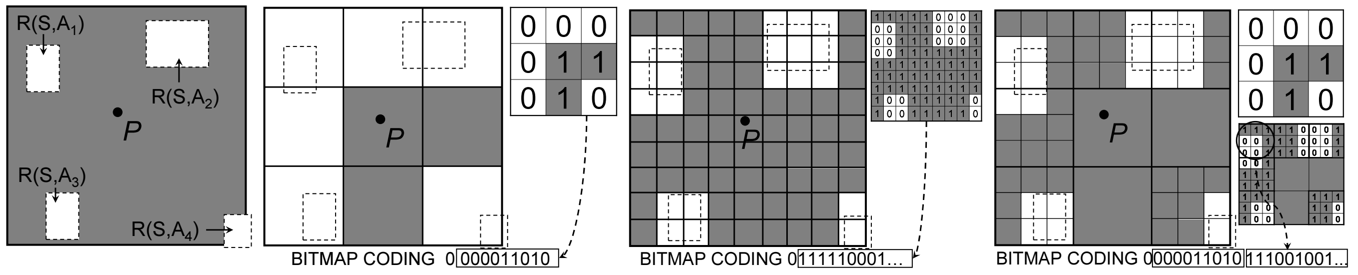


Fig. 3: LCSR from Component Rectangles

The above two approaches for safe region computation lack the flexibility to support heterogeneity among client computational capabilities. The same amount of computation is required on the part of each client to determine its position within the safe region. Our BSR approaches introduced below support client heterogeneity by computing more complex safe regions for clients with higher computational capacity. Each client may specify its level of computational capability to the server and the server *personalizes* safe region computation in accordance with the client capability.

D. Bitmap Encoded Safe Region Computation

In this section, we introduce the BSR computation approach which provides flexibility in safe region computation by providing larger, complex safe regions for clients with higher computational capacity. The safe region computation for each client can be personalized according to its capability. Figure 4(a) displays the monitoring region for subscriber at point P with four relevant alarm regions intersecting the grid cell. The server may compute the safe region for the client as the monitoring region and communicate this region to the client. Note that the server is virtually pushing the relevant alarms onto the client in this scenario by providing this safe


R(S,A$_1$)
R(S,A$_2$)
R(S,A$_3$)
R(S,A$_4$)
P
BITMAP CODING 0 000011010
BITMAP CODING 0 111110001…
BITMAP CODING 0 000011010 111001001…

(a) Monitoring Region (b) Grid Bitmap Encoding (3×3) (c) Grid Bitmap Encoding (9×9) (d) Pyramid Bitmap Encoding

Fig. 4: Bitmap Encoded Safe Region Computation

region. Each alarm region may be represented by the bottom-left and top-right corner point locations. We consider this as an *optimal approach* from the client perspective as the client has complete knowledge of all alarms in its vicinity in such a scenario. However, this approach may not be feasible from the point of view of communication costs incurred while broadcasting safe region to the clients. Additionally, for areas with high density the server may push a large number of alarms onto the client. For weak clients it may not be possible to handle a large number of alarms. To counter this problem, we now develop the concept of *Bitmap Encoded Safe Regions* which provides an estimation of the actual safe region using a bitmap.

Definition 4: A bitmap encoded safe region represents a safe region ξ_s for subscriber s using a bitmap B of length n. A bit value of 1 indicates that a predefined region (cell) belongs to the safe region; whereas a 0 bit indicates the negation.

We first describe a *$\underline{G}$rid $\underline{B}$itmap Encoded $\underline{S}$afe $\underline{R}$egion (GBSR)* computation technique and exhibit its inability to accurately and efficiently represent safe regions. An extension to this approach using a *$\underline{P}$yramid $\underline{B}$itmap Encoded $\underline{S}$afe $\underline{R}$egion (PBSR)* approach allows us to represent safe regions accurately as well as efficiently. BSR techniques exhibit the following advantages: (i) for low alarm density regions, it allows for reduction of location updates from the clients to the server when compared to the MPSR and LCSR approach, (ii) it supports different granularity of safe region computations for different subscribers thus supporting heterogeneity among client capabilities, and (iii) clients can determine their position with respect to the safe region using a predefined (worst case) number of computations.

E. Grid Bitmap Encoded Safe Region Computation

The safe region for a subscriber s can be represented by the set of grid cells as shown in Figure 4(b).

Proposition 5: We use a grid bitmap scheme to represent the safe region within the monitoring region shown in Figure 4(a). The cell $C_{k,l}$ is represented by a single bit $B(C_{k,l})$. If $C_{k,l} \bigcap \Sigma_{m=1}^{|A_s|} R(s, A_m) = \emptyset$ we set $B(C_{k,l}) = 1$ denoting that the entire cell $C_{k,l}$ belongs to the safe region ξ_s, else we set $B(C_{k,l}) = 0$ and split $C_{k,l}$ into $U \times V$ smaller equi-sized cells. The same encoding procedure is used for each smaller cell. This bitmap encoding technique provides a compact representation for safe region ξ_s.

Figure 4(b) shows the safe region representation for the safe region of Figure 4(a) using a bitmap encoding scheme. No alarm regions intersect the three darkened cells which are represented by 1's; other cells intersecting with alarm regions are represented by 0's. The safe region is represented using a simple bitmap $B = 0000011010$ which represents the cell bit values in a raster scan fashion. The first zero bit corresponds to the entire cell, indicating that the cell does not belong to the safe region and has spatial alarms intersecting with it. As visible from Figure 4(b), this safe region representation is able to represent only a small portion of the monitoring region thus providing a poor estimate of the actual safe region. Figure 4(c) presents a 9×9 split of the cell at a finer resolution which allows for more accurate representation of the safe region. However, this approach is inefficient in representing safe regions for the following two reasons: (i) it unnecessarily uses a much larger bitmap than required to represent the safe region, and (ii) different regions will have different alarm densities thus making it difficult to select a uniform grid cell size. Our PBSR approach allows for more accurate representations of the safe region while keeping the bitmap size small.

F. Pyramid Bitmap Encoded Safe Region Computation

The pyramid representation splits cells in the *base* grid (level L=0) with $B(C^0_{i,j}) = 0$ *only* into $U \times V$ smaller cells, where U,V are system defined parameters. The process may be further repeated for several iterations to form smaller cells at each level thus forming a pyramid data structure of height h. As shown using a pyramid structure with h=2 in Figure 4(d), by further splitting cells with $B(C^0_{i,j}) = 0$ into a 3×3 grid we obtain a much more accurate representation for the safe region. Compared to the grid-based approach which either does not represent the safe region accurately (3×3 grid in Figure 4(b)) or computes a much larger bitmap (9×9 grid in Figure 4(c)), the PBSR approach provides flexibility in computation of the safe region. For example, the GBSR approach requires 82 bits, 1 bit for the entire cell and 81 bits for the 9×9 grid, to represent the safe region in Figure 4(c). In comparison the PBSR approach requires only 64 bits, 1 bit for the entire cell, 9 bits for the cells at level 1 and only 54 bits for the cells at level 2, to represent the same safe region as shown in Figure 4(d).

The algorithm for safe region estimation using PBSR is given in algorithm 3 below. It accepts as inputs the base grid cell of the subscriber's current position $C^0_{k,l}$, maximum height

Algorithm 3: Pyramid Bitmap Encoded Safe Region Computation

```
    Input: $C^0_{k,l}$, $h$, $U$, $V$, $A^{rel}_s$
    Output: $B$
 1  $C^0 \leftarrow \{C^0_{k,l}\}$;
 2  $B = null$; $L = 0$;
 3  while ($L < h$) do
 4      $C^{L+1} \leftarrow SPLIT(C^L, U, V)$;
 5      $L = L + 1$;
 6  end
 7  $L = 0$;
 8  while ($L \leq h$) do
 9      for ($i = (k-1) \cdot U^L + 1$; $i \leq k \cdot U^L$; $i++$) do
10          for ($j = (l-1) \cdot V^L + 1$; $j \leq l \cdot V^L$; $j++$) do
11              if (($L = 0$) || (($L \neq 0$) && ($B(C^{L-1}_{\lfloor i/U \rfloor, \lfloor j/V \rfloor}) = 0$))) then
12                  if ($C^L_{i,j} \bigcap A^{rel}_s = \emptyset$) then
13                      $B(C^L_{i,j}) = 1$;
14                  else
15                      $B(C^L_{i,j}) = 0$;
16                  end
17                  $B = B \mid\mid B(C^L_{i,j})$;
18              end
19          end
20      end
21      $L = L + 1$;
22  end
```

h of the pyramid, splitting parameters U, V of the base grid cell and the set of alarms A^{rel}_s relevant to the subscriber within the grid cell.

The bitmap B is initially assigned a *null* value and current level L of the pyramid is set to zero (line 2). The pyramid representation of the base cells is constructed for height h by splitting cells iteratively into $U \times V$ cells (lines 3-6). This step can be performed offline by the server thus providing a precomputed pyramid representation for safe region computation. Next, starting from the base cells (level $L = 0$) we determine if each cell intersects any relevant alarms A^{rel}_s (line 12). Cells not intersecting with any relevant alarm regions are assigned a bit value $B(C^L_{i,j}) = 1$ indicating that they are a part of the safe region; else a cell is assigned bit value 0 (lines 12-16). For cells at each level $L-1$ ($L \leq h$) which have an assigned bit value 0, we consider the relevant $U \times V$ *children* cells at Level L and assign a bit value 0 or 1 considering intersection of the cell with relevant alarms at each level of the pyramid (lines 8-22).

Proposition 6: The PBSR approach for safe region representation allows us to represent the safe region ξ_s in terms of a bitmap of size $|B|$. The height of the pyramid h allows us to control the accuracy of representation of the safe region at the cost of computing a larger bitmap for more accurate representations.

We now define *Coverage* and *Bitmap Size* which allow us to control the quality of the safe region representation for our BSR computation techniques.

Definition 7: The coverage of a safe region representation ξ_s, denoted by $\eta(\xi_S)$, is defined as the ratio of area of the safe region using the BSR representation to the area of the monitoring region.

$$\eta(\xi_s) = \frac{\sum_{L=0}^{h} \sum_{i=(k-1)\cdot U^L+1}^{k\cdot U^L} \sum_{j=(l-1)\cdot V^L+1}^{l\cdot V^L} \frac{\alpha \cdot \beta}{(U \cdot V)^L} \cdot B(C^L_{i,j})}{\psi_s}, \tag{2}$$

where α, β defines the size of a base grid cell $C^0_{k,l}$ of the pyramid.

Definition 8: The bitmap size for safe region ξ_s, denoted by $\vartheta(\xi_s)$, is defined as the number of bits in the BSR representation of the safe region.

$$\vartheta(\xi_s) = 1 + \sum_{L=0}^{h-1} \sum_{i=(k-1)\cdot U^L+1}^{k\cdot U^L} \sum_{j=(l-1)\cdot V^L+1}^{l\cdot V^L} (1 - B(C^L_{i,j})) \cdot U \cdot V \tag{3}$$

In practice, we want to achieve high coverage with as small bitmap size as possible. Each client may specify the maximum height of the pyramid used for the PBSR representation of its safe region. In the worst case scenario, the client may need to determine its position relative to the safe region at each level of the pyramid data structure.

For the BSR approach, safe region for a client needs to be recomputed only when the client moves out of the grid cell. Note that a client may move out of its safe region without triggering any relevant alarms even while it is inside the grid cell. No recomputation of safe regions needs to be performed in such situations for the BSR approach. In case the client triggers an alarm on moving outside its safe region but stays within the cell $C^0_{k,l}$ corresponding to the safe region, the safe region can be quickly updated by considering the triggered alarm to be a part of the safe region. Additionally, BSR approaches can be optimized by precomputing the bitmap at each pyramid level for public alarms. Our experimental results do not consider this optimization for the BSR approaches, in stead performing bitmap computations on the fly.

1) Client Safe Region Containment Detection: The MPSR and LCSR approaches demand that the client monitor its position within rectangular shape safe region(s) which requires simple computations on part of the client. For the PBSR region approach, the client needs to determine its position with respect to the safe region from the bitmap $|B|$. The client determines its position at each level of the pyramid in order to determine if it is within the safe region or not. In the worst case scenario, the client needs to perform h computations, one at each level of a pyramid of height h; on an average it will perform much fewer than h computations. Algorithm 4 outlines the client safe region containment logic required for the PBSR approach.

The algorithm accepts as input the bitmap B and the position vector $\vec{p_s}$ for subscriber s. The algorithm returns the containment detection result CDR indicating a value true if client lies inside safe region or false if client lies outside safe region. Initially the level L of the pyramid is set to zero. The algorithm also identifies the start index $LStartIndex$ and end index $LEndIndex$ for bitmap values in B related to level L.

Algorithm 4: PBSR Client Safe Region Containment Detection

```
    Input: $B$, $\vec{p_s}$
    Output: $CDR \in \{true, false\}$
1   $L = 0; LStartIndex = 0; LEndIndex = 0; posIndex = 0;$
2   $numFalsePrevL = 0; numFalsePrevLPosIndex = 0;$
3   while $(LStartIndex \neq |B|)$ do
4       if $(B[posIndex] == true)$ then
5           return $true$;
6       else
7           $L = L + 1;$
8           for $(i = LStartIndex; i \leq LEndIndex; i++)$ do
9               if $(B[i] == false)$ then
10                  if $(i < posIndex)$ then
11                      $numFalsePrevLPosIndex++;$
12                  end
13                  $numFalsePrevL++;$
14              end
15          end
16          $cellid = getRelCellPos(\vec{p_s}, L);$
17          $LStartIndex = LEndIndex + 1;$
18          $LEndIndex += (numFalsePrevL \cdot U \cdot V);$
19          $posIndex = LStartIndex + cellid +$
            $(numFalsePrevLPosIndex \cdot U \cdot V) - 1;$
20          $numFalsePrevL = 0; numFalsePrevLPosIndex = 0;$
21      end
22  end
23  return $false$;
```

We define the set of bitmap values from start to end index for any level L as a $block$. The bitmap index concurrent to the cell the client currently belongs to is indicated by $posIndex$ (line 1). Additionally, the algorithm needs to keep a count of the number of $false$ bits $numFalsePrevL$ in a block and the number of false bits $numFalsePrevLPosIndex$ before the $posIndex$ in a block (line 2). The algorithm checks for each level L, if the bitmap value is $true$ indicating that client location lies within the safe region (lines 4-6). Otherwise the algorithm increments the level L and maintains the count of number of false values in previous block and the number of false values before $posIndex$ in previous block of the bitmap (lines 8-15). These values are used to determine the $LStartIndex$, $LEndIndex$ and $posIndex$ in the new block corresponding to the next level of the pyramid in the bitmap (lines 16-19). This computation is repeated for each level L of the pyramid to determine if the subscriber lies within the safe region (lines 6-21).

In order to facilitate installation of new alarms, the server maintains a main memory grid index on the safe region of all clients. Location updates are required of all clients whose safe region intersects the new spatial alarm region. Spatial alarm information is indexed using a disk resident R-tree structure. We use a 3-dimensional R-tree which indexes the subscriber relevance information for private, shared and public alarms as well as the bottom-left and top-right points of the safe region *minimum bounding rectangles (MBRs)*.

IV. Experimental Evaluation

In this section, we evaluate the performance of our safe region computation techniques using four different sets of experiments. The first experiment is aimed at gaining an understanding of the functioning of the MPSR and LCSR approaches. The second set of experiments performs an evaluation of the two BSR approaches: GBSR and PBSR. The third set of experiments compares the performance of our safe region approaches. The final set of experiments provides an evaluation of the safe region techniques compared to periodic processing (PRD), safe period-based (SP) computation and an optimal (OPT) approach. The optimal approach does not consider any restrictions on resource availability and assumes all relevant alarms within the monitoring region are pushed to the client, which implies the client is fully aware of all relevant alarms in its vicinity. We measure the performance of all approaches based on four different evaluation metrics:

CPU Load/Capacity: This factor measures the scalability of the system. It is measured as the ratio of the amount of CPU time used by the system to perform alarm processing and safe region or safe period computations to the amount of time available to the system to perform this processing. CPU load/capacity of $>$ 100% indicates the failure of the system to scale to the desired configuration.

Wireless Communication Cost: This is measured by the number of updates sent to the system by the mobile clients. We measure this parameter as a ratio of the communication costs required by a particular approach to the communication costs incurred by periodic alarm processing at a frequency high enough to trigger all relevant alarms.

Bandwidth: This is the downstream bandwidth (in Mbps) required by the system to communicate the safe region (or alarm information) to the clients for the safe region (or optimal) approaches.

Client Computation Cost: This metric indicates the cost incurred by clients to check their position relative to the safe region in terms of average number of computations performed per client per second.

We do not measure alarm trigger accuracy as the parameters adopted for each processing approach ensure 100% of the alarms are triggered in all scenarios. The sequence of alarms to be triggered is determined by a very high frequency trace of the motion pattern of the vehicles. We briefly describe the experimental setup used to evaluate our system below.

A. Experimental Setup

Our simulator generates a trace of vehicles moving on a real-world road network using maps available from the National Mapping Division of the U.S. Geological Survey (USGS [2]) in Spatial Data Transfer Format (SDTS [1]). Vehicles are randomly placed on the road network according to traffic densities determined from the traffic volume data in [8]. The simulator simulates the motion of vehicles on roads with appropriate velocity information; at intersections, vehicles may move in any direction with

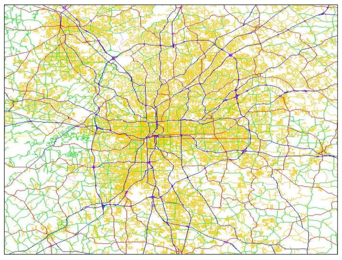
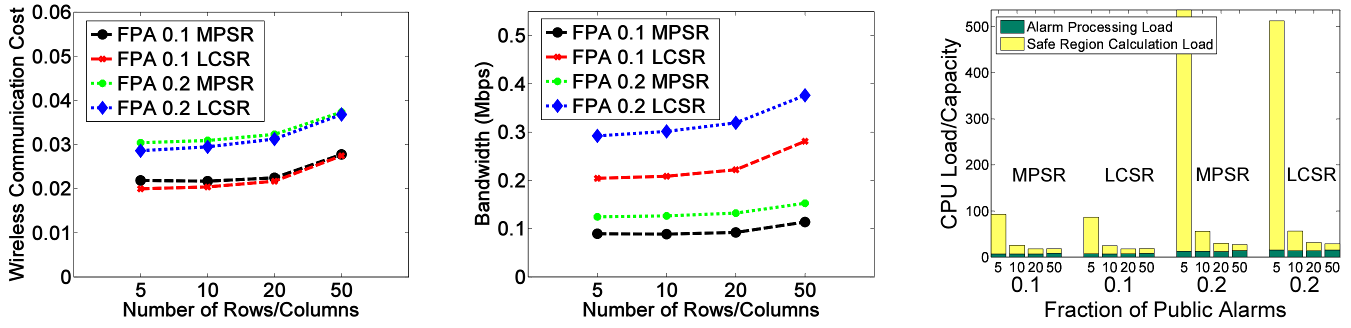


Fig. 6: Road Network for Atlanta and Surrounding Areas



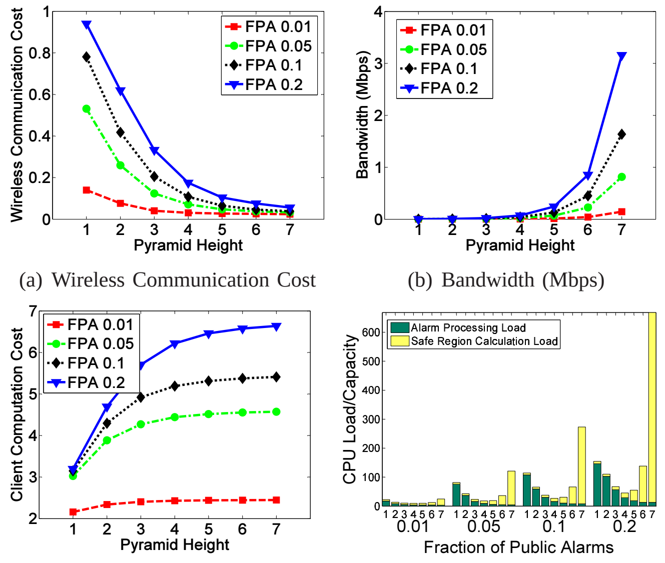
(a) Wireless Communication Cost (b) Bandwidth (Mbps) (c) CPU Load/Capacity

Fig. 5: Performance of MPSR and LCSR Approach

attached probability values. We use a map of Atlanta and surrounding region as shown in Figure 6, which covers an area around 1000 km^2 in expanse, to generate the trace. Our experiments use traces generated by simulating vehicle movement for a period of one hour, results are averaged over a number of such traces. Default traffic volume values allow us to simulate the movement of a set of 10,000 vehicles. Each vehicle generates a set of position parameters during the simulation which are evaluated against the generated spatial alarm information. Default values require each vehicle to generate updates with a period of less than a second for periodic processing. The default spatial alarm information consists of a set of 10,000 spatial alarms installed uniformly over the entire map region. We vary the fraction of private, shared and public alarms installed in the system to vary the number of alarms relevant to each client. This simulator setup allows us to the test the robustness of our framework under realistic mobility patterns.

B. Experimental Results

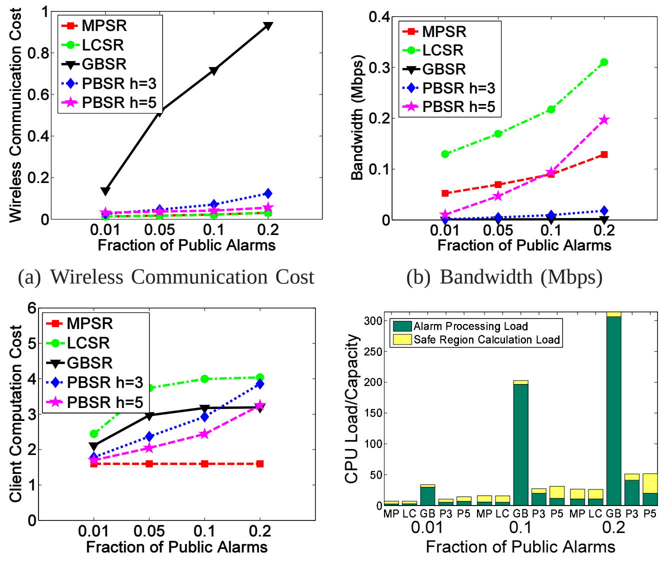
1) Performance of MPSR and LCSR Approach: The first set of experiments compares the performance of the MPSR and LCSR approach. The experimental setup uses default values as defined in Section IV-A. We vary the *fraction of public alarms* (abbreviated as FPA in Figure 5) in the system from 0.01 to 0.2, thus increasing the number of alarms relevant to each subscriber. Private and shared alarms are installed in the system in the ratio 2:1. Note that with increasing fraction of public alarms, the effective density of relevant alarms for each subscriber in the system increases. The size of the monitoring region is varied by varying the number of rows/columns in the grid-based framework. Results are displayed for FPA values of 0.1 and 0.2 to avoid clutter. Figure 5(a) displays the wireless communication cost as a fraction of the communication costs incurred for corresponding periodic alarm processing. Three trends are to be observed from Figure 5(a). Firstly, as we decrease the size of grid cells comprising the monitoring region (increase number of rows/columns) the wireless communication costs increase. Smaller monitoring regions imply smaller safe regions; hence, a client moves out of its safe region more frequently and provides more frequent updates to the server. Secondly, as we increase the density of relevant alarms by increasing the fraction of public alarms wireless communication costs increase. Again, in presence of larger number of relevant alarms smaller safe regions are computed and the client needs to update its position more frequently in this scenario. However, the increase in communication costs is non-linear; from FPA 0.01 to FPA 0.2, when the average number of relevant alarms for each client increases by almost 20 times, the increase in wireless communication costs is only three times or lower. Last but not the least, the LCSR approach requires around 10% lower communication costs compared to MPSR approach for larger grid cell sizes (5 or 10 rows/columns in the figure). For smaller grid cell sizes (20 or 50 rows/columns in the figure) the gap between the two approaches reduces; across different FPA values similar trends are observed. The downstream bandwidth for broadcasting safe region from the server to the clients also increases with decreasing grid cell sizes as can be observed from Figure 5(b). This is a direct result of the increase in number of updates being processed at the server resulting in more frequent safe region computations. The bandwidth consumption is reasonably low, not larger than 400 Kbps for 10,000 clients for the LCSR approach in the worst case scenario. As expected the LCSR approach has around 2.5 times the bandwidth consumption incurred by the MPSR approach due to larger safe region representation size. The CPU load/capacity numbers in Figure 5(c) display that the system load is reasonably low except for the largest grid cell size which implies infeasibility of using such large grid cells. As expected, with higher FPA values the computational load on the server increases due to higher number of relevant alarms being processed for each client. Secondly, we observe that the CPU load decreases with decreasing grid cell sizes. As we decrease the size of the grid cell for a particular FPA value, the safe region computation costs decrease. As fewer alarms are considered for each computation, the cost of performing a single safe region computation decreases. Even though safe region computations will be performed more frequently with decreasing grid cell size, the net effect results in reduction in safe region computation costs. However, the alarm processing costs rise with decreasing grid cell size due to larger number of location updates being processed by the alarm processing server. The appropriate number of rows/columns can be observed to be around 20 to 50, as the total CPU load is lowest with this setting. Another trend that



(a) Wireless Communication Cost (b) Bandwidth (Mbps)

(c) Client Computation Cost (d) CPU Load/Capacity

Fig. 7: Performance of BSR Approach



(a) Wireless Communication Cost (b) Bandwidth (Mbps)

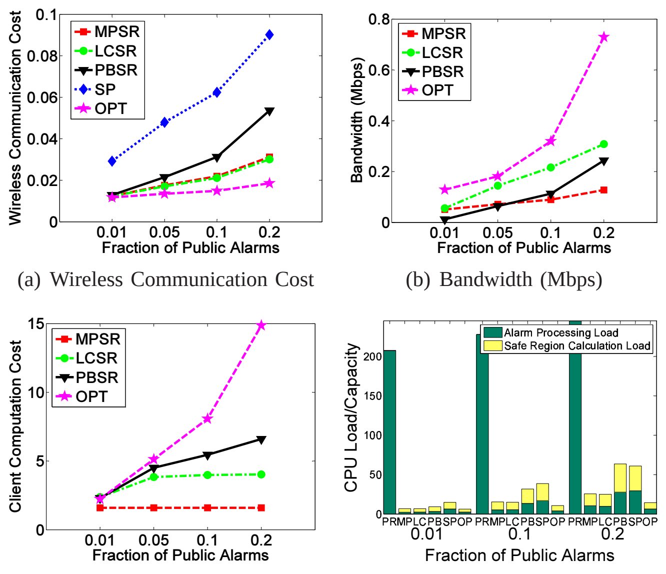
(c) Client Computation Cost (d) CPU Load/Capacity

Fig. 8: Performance Comparison of Safe Region Approaches

can be observed here is that the LCSR approach incurs around 5% lower costs compared to the MPSR approach due to lower number of location updates being processed and a relatively simpler safe region computation approach.

2) Performance of BSR Approach: This set of experiments is designed to evaluate the performance of the BSR approach. We vary the height of the pyramid from $h = 1$ (for GBSR) to $h = 7$ and observe the performance as shown in Figure 7. Figure 7(a) displays the wireless communication costs incurred as we increase the pyramid height from $h = 1$ to $h = 7$. It can be observed that the GBSR approach is highly inefficient as it limits safe region computation to a very high granularity. The safe region computed using this approach provides a very coarse representation of the actual safe region forcing the clients to frequently update their location as a result of which GBSR approach incurs high communication costs. As we increase the pyramid height, more accurate safe region representations can be computed and consequently wireless communication costs experience a sharp drop. Another observation is that BSR approaches display high sensitivity to alarm density levels; the performance deteriorates sharply for higher FPA values. On the other hand, the bandwidth required by the server to broadcast the safe regions to the clients increases with pyramid height (Figure 7(b)). For higher level pyramids, larger bitmaps are required to represent the safe region and hence higher bandwidth is required. For pyramid height $h = 7$, with high alarm density the downstream bandwidth requirement goes up to 3.2 Mbps, but for $h = 5$ this value remains below 250 Kbps even when fraction of public alarms is increased to 0.2. Figure 7(c) displays the average number of computations performed per client per second to determine its position within the safe region. Clients need to perform the safe region containment detection check as described in Section III-F.1. For the GBSR approach the clients need to perform an average of 2-3 computations per second. This cost does not experience a significant increase with pyramid height for low FPA values. For higher FPA values the costs rise to 6-7 computations per second for a pyramid of height $h = 7$. As seen from Figure 7(d), for low pyramid height, safe region computation costs are low as relatively simpler computations are involved. On the other hand, alarm processing costs are high as a large number of updates are received from clients. On increasing pyramid height alarm processing costs drop due to fewer client position updates. The safe region computation costs increase due to high complexity of safe region computation. Even despite the fewer number of safe region computations being performed at higher pyramid height, the increase in cost of a single safe region computation is such that a net increase in safe region computation load is experienced. However, this cost can be significantly offset by using precomputed bitmaps for public alarms as described earlier. For $h = 4$ or $h = 5$, the overall CPU load is at its lowest point. With increasing FPA values the system experiences an increase in CPU load.

3) Performance Comparison of Safe Region Approaches: This section provides a performance comparison of the various safe region computation techniques developed in this work, MPSR, LCSR, GBSR and PBSR (for $h = 3$ and $h = 5$), with varying alarm density levels. As can be observed from Figure 8(a), the GBSR approach incurs heavy communication costs due to the inaccurate nature of computed safe regions. The MPSR, LCSR and PBSR approaches perform well even at higher alarm density levels. For low FPA values the PBSR approach outperforms both the MPSR and LCSR approach; whereas for higher FPA values the MPSR and LCSR approaches perform better. As far as the downstream bandwidth consumption is concerned, we observe from Figure 8(b) that the PBSR approach for $h = 3$ performs better than the MPSR approach at all alarm density levels. The PBSR approach with $h = 5$ performs better than the MPSR



(a) Wireless Communication Cost (b) Bandwidth (Mbps)

(c) Client Computation Cost (d) CPU Load/Capacity

Fig. 9: Performance Comparison of Safe Region with Other Approaches

approach for all alarm density levels except for FPA value of 0.2. The GBSR approach performs the best on this metric; however, the approach is not competitive from point of view of wireless communication costs as well as CPU load/capacity. Figure 8(c) displays the client computation costs incurred by the different approaches. The MPSR approach requires the lowest computation costs for client containment detection and the cost does not vary with the alarm density. Other approaches experience an increase in this cost at higher alarm density levels. The PBSR approach experiences significant increase in computational costs with increase in the fraction of public alarms. The client computation cost for this approach are higher than MPSR approach but lower than the LCSR approach. Figure 8(d) displays the CPU load experienced by each of the safe region approaches. The MPSR (MP in figure), LCSR (LC in figure) and the PBSR approach with $h = 3$ and $h = 5$ (denoted as P3 and P5 in the figure) have low CPU load whereas the GBSR approach (GB in the figure) fails to scale to this configuration. The alarm processing costs with the GBSR region approach are prohibitively high as alarm processing has to be performed on a large number of client location updates.

4) Performance Comparison of Safe Region with Periodic, Safe Period and Optimal Approach: Now we compare the performance of the safe region approaches with periodic processing, safe period-based processing and the optimal approach. As can be seen from Figure 9(a), the safe region approaches incur very low wireless communication costs. Periodic processing requires clients to transmit each location update to the server incurring a wireless cost of 1 and is not shown in the figure. The safe period approach experiences significantly higher communication costs, approximately 2-3 times the cost incurred by the safe region approaches. This is largely due to the pessimistic assumptions required to ensure that the safe period approach triggers all alarms with a 100% success rate. The MPSR and LCSR approach incur around 40% higher communication costs compared to an optimal approach even for FPA value 0.2. For lower alarm density levels the gap between the optimal and safe region approaches is much lower. The optimal approach would require clients to transmit updates only when the spatial constraints for one or more relevant alarms are met. Figure 9(b) displays the downstream bandwidth consumed by the system to broadcast safe regions or relevant alarms (in case of OPT approach) to the clients. Safe period approach would also require that a computed safe period be broadcast to each client; however, we exclude the bandwidth incurred for this approach from these results. As expected the safe region approaches incur much lower bandwidth expense when compared with an optimal solution. PBSR ($h = 5$) performs the best for low FPA values (low relevant alarm density); for higher FPA values the performance of the PBSR approach declines and is worse than that of the MPSR approach. However, the PBSR approach always performs better than the LCSR approach on this metric. Not surprisingly, client computational costs for the optimal approach are significantly higher than the safe region approaches (Figure 9(c)) as the optimal solution is based on the assumption that clients have high computational capability. PBSR, MPSR and LCSR approaches require lower client computational costs especially at higher alarm density levels. The CPU load experienced by each approach is as shown in Figure 9(d). Periodic approach (PR) has much higher alarm processing costs as each update needs to be processed by the client and the CPU load does not scale. The processing load does not rise much at higher alarm densities as each update is processed by this approach for all FPA values. The MPSR and LCSR approaches experience lower CPU load due to much lower alarm processing load. With increasing FPA values, the safe region computation as well as the alarm processing load rises; however, the total load incurred by the system is much lower than the periodic approach for all configurations. The PBSR approach again shows similar trends as the MPSR and LCSR approaches; however, the CPU load incurred by this approach at higher FPA values are higher than MPSR and LCSR approaches. The safe period (SP) approach experiences much higher CPU load compared to the safe region approaches. This is a direct result of the larger number of updates that need to be processed by the safe period approach. Results for the optimal approach are plotted to show that the safe region approaches do not incur much higher CPU load except for the highest FPA values.

V. Related Work

An event-based location reminder system has been advocated by many human computer interaction projects [14], [19], [7], [15], [12]. In the realm of information monitoring, event-based systems have been developed to deliver relevant information to users on demand [13], [4]. In addition to monitoring continuously changing user information needs, spatial alarm processing systems also need to deal with the complexity of monitoring user location data in order to trigger relevant alerts in a non-intrusive manner.

A large body of work exists on monitoring continuous queries assuming known movement trajectories [10], [20]. A second category does not make any assumptions on movement patterns. [23], [25] propose the idea of returning moving query results with a validity scope. Periodic reevaluation approach is commonly used for continuous monitoring of moving objects [11], [16], [17], [24]. Incremental reevaluation for range and kNN queries was also proposed in [16], [22]. Spatial alarms differ from this work as they do not demand periodic evaluation or reevaluation like continuous queries; in stead they require one shot evaluation which should result in a trigger when the alarm trigger conditions are satisfied. Much of our work is focussed on determining the opportune moment for evaluating spatial alarms relevant to a client.

Safe Region computation techniques have been developed for continuous queries in [17], [5], [9]. None of the previous work except [9] presents clear algorithms for safe region computation. Our work differs from [9] as we present safe region computation for spatial alarms. Further, the algorithms presented in [9] do not consider scenarios with overlapping query regions, query regions overlapping multiple grid cells and query regions intersecting the axes of the coordinate system with the client position at the origin. Our MPSR and LCSR computation algorithms are able to handle such scenarios. Algorithms presented in previous work also fail to consider an environment supporting heterogeneous client capabilities. Our BSR techniques explore possibilities for supporting client heterogeneity and present flexibility of computing more complex safe regions for clients with higher computational power.

VI. Conclusion

We have presented a safe region-based processing technique and different safe region computation algorithms for ensuring accurate, scalable processing of spatial alarms. This paper makes three important contributions towards supporting efficient processing of spatial alarms. First, we introduce the concept of safe region-based alarm processing to enhance the scalability of the system. We develop three techniques for safe region computation, namely the MPSR approach, LCSR approach and the BSR approach. Second, we provide an in-depth study that evaluates the various heuristics behind our safe region computation techniques. A detailed experimental evaluation is conducted for the proposed safe region computation algorithms. We obtain two important insights. First, none of the safe region computation algorithms can outperform all others over all evaluation metrics. It is essential to take into account dynamically changing load conditions and resource constraints for a region when deciding on an appropriate safe region computation technique. Last but not the least, our framework supports heterogeneous environments with varying server load and resource conditions and heterogeneity of client capabilities. For instance, the Bitmap Encoded Safe Region (BSR) approach supports client heterogeneity by computing different granularity safe regions for each client according to their computational capacity. Our experimental evaluation shows that the safe region techniques outperform other existing spatial alarm processing techniques like periodic evaluation, safe period-based approach, and offer close to optimal performance for different alarm distribution scenarios.

References

[1] Spatial Data Transfer Format. `http://www.mcmcweb.er.usgs.gov/sdts/`.

[2] U.S. Geological Survey. `http://www.usgs.gov`.

[3] B. Bamba, L. Liu, and P. S. Yu. Scalable Processing of Spatial Alarms. Technical report, Georgia Institute of Technology, 2008.

[4] V. Bazinette, N. Cohen, M. Ebling, G. Hunt, H. Lei, A. Purakayastha, G. Stewart, L. Wong, and D. Yeh. An Intelligent Notification System. *IBM Research Report RC 22089 (99042)*, 2001.

[5] Y. Cai and K. Hua. An Adaptive Query Management Technique for Efficient Real-Time Monitoring of Spatial Regions in Mobile Database Systems. In *IEEE IPCCC*, pages 259–266, 2002.

[6] Computer Science and Telecommunications Board. IT Roadmap to a Geospatial Future. *The National Academics Press*, 2003.

[7] A. Dey and G. Abowd. CybreMinder: A Context-Aware System for Supporting Reminders. In *Second International Symposium on Handheld and Ubiquitous Computing*, pages 172–186, 2000.

[8] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *MobiSys*, 2003.

[9] H. Hu, J. Xu, and D. Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *ACM SIGMOD*, pages 479–490, 2005.

[10] G. Iwerks, H. Samet, and K. Smith. Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *VLDB*, pages 512–523, 2003.

[11] C. Jensen, D. Lin, and B. Ooi. Query and Update Efficient B+-Tree based Indexing of Moving Objects. In *VLDB*, pages 768–779, 2004.

[12] S. Kim, M. Kim, S. Park, Y. Jin, and W. Choi. Gate Reminder: A Design Case of a Smart Reminder. In *Conference on Designing Interactive Systems*, pages 81–90, 2004.

[13] L. Liu, C. Pu, and W. Tang. WebCQ - Detecting and Delivering Information Changes on the Web. In *CIKM*, pages 512–519, 2000.

[14] 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 Conference on Human Factors in Computing Systems*, pages 889–898, 2006.

[15] N. Marmasse and C. Schmandt. Location-Aware Information Delivery with ComMotion. In *HUC*, pages 157–171, 2000.

[16] M. Mokbel, X. Xiong, and W. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases. In *ACM SIGMOD*, pages 623–634, 2004.

[17] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers*.

[18] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Reading, Mass, 1990.

[19] 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.

[20] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, pages 287–298, 2002.

[21] O. Wolfson, A. Sistla, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.

[22] X. Xiong, M. Mokbel, and W. Aref. SEA-CNN: Scalable Processing of Continuous k-Nearest Neighbor Queries in Spatio-Temporal Databases. In *ICDE*, pages 643–654, 2005.

[23] J. Xu, X. Tang, and D. Lee. Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments. *IEEE TKDE*, pages 474–488, 2003.

[24] X. Yu, K. Pu, and N. Koudas. Monitoring k-Nearest Neighbor Queries over Moving Objects. In *ICDE*, pages 631–642, 2005.

[25] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. Lee. Location-based Spatial Queries. In *ACM SIGMOD*, pages 443–454, 2003.